1 Einleitung

DRIVE/WINDOWS ermöglicht mit SQL-Anweisungen den Zugriff auf das Datenbanksystem SESAM/SQL-Server V2. Dieses Lexikon beschreibt in Kurzform die genaue Syntax der DRIVE-SQL-Anweisungen für SESAM/SQL V2 in der DRIVE/WINDOWS-Version 2.1 für BS2000, MS-Windows und SINIX.

Eine ausführliche Beschreibung der SQL-Anweisungen für SESAM/SQL V2 finden Sie im Handbuch SESAM/SQL-Server (BS2000/OSD), SQL-Sprachbeschreibung Teil 1: SQL-Anweisungen [18].

1.1 Aufbau des Handbuchs

Das Kapitel DRIVE-SQL-Anweisungen enthält einen alphabetischen Nachschlageteil über alle SQL-Anweisungen.

Komplexe Anweisungsteile, die sowohl in DRIVE- als auch in SQL-Anweisungen enthalten sind, werden gesondert beschrieben im Kapitel „Metavariablen“ des DRIVE- Lexikons [3]. Davon abweichende DRIVE-SQL-Metavariablen werden in diesem Lexikon in einem eigenen Kapitel „Metavariablen“ beschrieben. Die Beispiele zu Metavariablen beschreiben z.T. nur die Metavariable und nicht den gesamten syntaktischen Kontext, der die folgende Datensatzausgabe ermöglicht.

Der Zugriff auf das Datenbanksystem SESAM/SQL V2 wird von allen drei Plattformen (BS2000, SINIX, MS-Windows) unterstützt, auf denen DRIVE/WINDOWS abläuft. Bitte beachten Sie, daß es auf der Plattform MS-Windows keinen Dialog-Modus gibt und keinen Betriebsmodus mit TP-Monitor (im BS2000 UTM-Betrieb). Außerdem werden auf dieser Plattform die DISPLAY- und DISPATCH-Anweisungen nicht unterstützt. Entsprechende Aussagen beziehen sich auf DRIVE-Anwendungen, die auf SINIX und BS2000 ablaufen.

Die Einteilung von SQL-Fehlern in DRIVE-Fehlerklassen (Systemvariable &DML_STATE) erfolgt anhand des SQLCODEs (Systemvariable &SQL_CODE) kompatibel zu SESAM/SQL V1. Ein SQL-Fehler mit SQLSTATE XXXXX, SQLCODE -xxx oder -1xxx oder -2xxx und Meldungstext „SEWXXXX < Meldungstext>“ wird nicht mehr als Fehlermeldung der Form DRI9xxx ausgegeben, sondern als Fehlermeldung DRI0536 in der Form „DRI0536 XXXXX xxxx < Meldungstext>“. DRIVE/WINDOWS unterstützt den SQLSTATE von SESAM/SQL V2 durch die neue Systemvariable &SQL_STATE (siehe DRIVE-Sprachbeschreibung [2], Kapitel „Variablen und Konstanten einsetzen“). Die SQLSTATEs einschließ-

lich einer Gegenüberstellung aller SQLCODEs finden Sie im SESAM-Handbuch Meldungen [24], eine kurze Übersicht im Abschnitt „Abbildung der SQLCODEs von SESAM auf &DML_STATE“ auf Seite 257 dieses Lexikons.

Das Kapitel „Syntaxübersicht“ ist ein alphabetischer Nachschlageteil über alle im Handbuch verwendeten Anweisungen und Metavariablen.

Die Anweisungen sind alphabetisch angeordnet. Pro Anweisung gibt es einen Eintrag, der den Namen der Anweisung als Kopfzeile hat und eine Kurzbeschreibung.

Anweisungsname - Kurzbeschreibung

Nach der Überschrift wird die Wirkungsweise der Anweisung beschrieben.

Dieser Abschnitt beschreibt auch die Voraussetzungen, die für die erfolgreiche Ausführung der Anweisung erfüllt sein müssen. Insbesondere sind die notwendigen Zugriffsrechte zusammengestellt.

ANWEISUNGSNAME KLAUSEL parameter ...

KLAUSEL
Erklärungen zur Klausel.

parameter
Erklärungen zum Parameter.

Die Klauseln und Parameter sind in der Reihenfolge beschrieben, in der sie in der Syntaxdefinition vorkommen.

1.2 Aufbau von DRIVE-SQL-Anweisungen

DRIVE-SQL-Anweisungen bestehen aus folgenden Elementen:

- Schlüsselwörtern
- Namen
- Literalen
- Metavariablen
- Trennzeichen
- Kommentaren

Beispiel

```
CYCLE cursorname INTO &variablenname.*;' /*lies cursorname satzweise nach */
                    /* variablenname */

CYCLE WHILE &SQL_CLASS <> 02  AND variable <= 1000;
                      /* Schleife solange bis keine Sätze */
                      /* mehr kommen, aber höchstens */
                      /* eintausendmal */
```

Schlüsselwörter:	CYCLE, INTO, WHILE, AND, SQL_CLASS (Name einer DRIVE-Systemvariablen)
Namen:	cursorname, variablenname
Literale:	'02', 1000
Metavariablen:	variable, .*
Trennzeichen:	Leerzeichen, Vergleichsoperatoren <>, <=, Semikolon
Kommentar:	/* lies cursorname satzweise nach */ /* variablenname */

Schlüsselwörter

Schlüsselwörter sind Wörter, die so angegeben werden müssen, wie sie im Handbuch dargestellt sind. Eine Aufstellung aller DRIVE-Schlüsselwörter befindet sich im Anhang des DRIVE-Lexikons [3].

Namenskonventionen

Namen kennzeichnen variable Werte, die bei der Eingabe vom Anwender durch aktuelle Werte ersetzt werden müssen.

Namen dürfen Buchstaben, Ziffern und Sonderzeichen enthalten, wenn keine weitergehenden Einschränkungen beschrieben sind.

Namen, die Buchstaben, Ziffern und Unterstriche (_) enthalten, müssen nicht besonders gekennzeichnet werden. Namen, die darüber hinaus weitere Sonderzeichen enthalten, müssen in Anführungszeichen (") gesetzt werden.

Ein Katalogname darf bei SESAM V2 nicht länger als 18 Zeichen sein, ein (einfacher) Schemaname nicht länger als 31 Zeichen. Der Name eines Berechtigungsschlüssels (SQL-User-Name) darf nicht länger als 18 Zeichen sein. Die weiteren SESAM-Regeln für Katalognamen und Berechtigungsschlüssel sind zu beachten.

Ist ein Name nicht in die Sonderzeichen " eingeschlossen (regulärer Name), muß er mit einem Buchstaben beginnen, gefolgt von weiteren Buchstaben, Ziffern oder Unterstrichen. Er darf kein DRIVE-Schlüsselwort sein.

Ist ein Name in die Sonderzeichen " eingeschlossen (spezieller Name), darf er nicht mit einem Unterstrich beginnen und jedes abdruckbare Zeichen enthalten. Das Sonderzeichen " ist zu verdoppeln. Die äußeren Sonderzeichen " zählen bei der Namenslänge nicht mit, die inneren zählen nur einfach.

Bei qualifizierten speziellen Schema-, Tabellen- oder Spaltennamen wird empfohlen, den Qualifizierungspunkt (das Sonderzeichen .) außerhalb der Sonderzeichen " zu verwenden, d.h. die Spezialisierung auf einfache Namen zu beschränken. DRIVE/WINDOWS behandelt jedenfalls spezielle Namen als einfache Namen.

Es gibt DRIVE-Schlüsselwörter, die keine SESAM-Schlüsselwörter sind (z.B. KEY). Soll ein solches Schlüsselwort in einer SQL-Anweisung als Name eines SQL-Objekts (z.B. einer Spalte) verwendet werden, so muß es, im Unterschied zu SESAM, als spezieller Name angegeben werden.

Literale

Literale sind Konstanten, die in der angegebenen Form vom Sprachübersetzer übernommen werden.

Numerische Literale werden direkt angegeben und hexadezimale Literale (nur in DRIVE-Anweisungen) mit X'*literal*' angegeben.
Alphanumerische Literale müssen in Hochkommas gesetzt werden.
Für Datumzeit-Literale müssen Sie angeben, ob das Literal ein Datum, eine Uhrzeit oder einen Zeitstempel enthält. Für Intervall-Literale (nur in DRIVE-Anweisungen) müssen Sie eine Einheit für die Zeitspanne angeben.

Hochkommas (') in Literalen müssen durch Hochkommas entwertet werden.

Beispiel

Das Literal „So geht's:“ wird folgendermaßen geschrieben:

```
'So geht''s:'
```

Metavariablen

Metavariablen sind komplexe Anweisungsteile, die zur Erleichterung der Übersichtlichkeit aus einer Anweisung ausgegliedert werden. Sie werden in einem eigenen Kapitel beschrieben.

Trennzeichen

Trennzeichen müssen zwischen Schlüsselwörtern, Namen, Literalen und Metavariablen angegeben werden, um sie eindeutig identifizieren zu können. Als Trennzeichen dienen:
– das Semikolon (Anweisungstrenner)
– das Leerzeichen
– das Tabulatorzeichen
– das Komma (,)
– der Verknüpfungsoperator ||
– alle Vergleichsoperatoren = < > <= >= <>
– alle Rechenoperatoren + – * / % **

Außerhalb von Zeichenfolgen, die in Hochkomma (') oder Anführungszeichen (") eingeschlossen sind, wirkt auch ein Kommentar oder ein Zeilenende wie ein Trennzeichen.

Kommentare

Ein DRIVE-Kommentar wird eingeleitet durch die Zeichenfolge /* und abgeschlossen durch die Zeichenfolge */. Zwischen diesen Kommentarzeichen kann beliebiger Text stehen, der auch über mehrere Zeilen gehen kann.

Die Zeichenfolgen /* und */ kennzeichnen keine Kommentare, wenn sie in Anführungszeichen (") oder Hochkommas (') stehen.

SQL-Kommentare (vgl. SESAM-Handbuch [18], Kapitel 3.2.5) sind in DRIVE/WINDOWS nicht erlaubt.

1.3 Verwendete Darstellungsmittel

In diesem Handbuch verwenden wir folgende Darstellungsmittel:

—————— ——————	Syntaxdefinitionen
GROSS	SQL-Schlüsselwörter
fett	Hervorhebung im Fließtext
kursiv	Frei wählbare Namen und Metavariablen in Syntaxdefinitionen und im Fließtext
Schreibmaschinenschrift	Feststehende Namen (z. B. Kommandos auf Betriebssytemebene, Dateinamen) und Fehlermeldungen im Fließtext, Programmtext in Beispielen und feste Namen von Beispieltabellen im Fließtext

::=	Definitionszeichen Die Angabe auf der rechten Seite von ::= definiert die Syntax für das Element auf der linken Seite.
[ ]	Optionale Angaben Die eckigen Klammern sind Metazeichen, die in einer SQL-Anweisung nicht angegeben werden.
{ | }	Alternative Angaben in Syntaxdefinitionen. Eine der Alternativen in geschweiften Klammern muß angegeben werden. Die geschweiften Klammern sind Metazeichen, die in einer SQL-Anweisung nicht angegeben werden.
{ } ...	Zusammenfassung von Klauseln in Syntaxdefinitionen, die gemeinsam wiederholt werden können. Die geschweiften Klammern sind Metazeichen, die in einer SQL-Anweisung nicht angegeben werden.
...	In Syntaxdefinitionen bedeuten die Punkte, daß die vorausgehende Angabe beliebig oft wiederholt werden kann. In Beispielen bedeuten die Punkte, daß die restlichen Teile für das Beispiel ohne Bedeutung sind. Die Punkte sind Metazeichen, die in einer SQL-Anweisung nicht angegeben werden.
	Dieses Zeichen weist Sie auf eine sehr wichtige Information hin, die Sie unbedingt beachten müssen.

2 Arbeiten mit SESAM/SQL V2

Dieses Kapitel beschreibt:

– die Organisation einer SESAM V2-Datenbank und die Migration einer SESAM V1-Datenbank am Beispiel (Seite 10)

– Regeln und Empfehlungen für den Zugriff auf SQL-Objekte von SESAM V2 durch DRIVE-Programme (Seite 12)

– die Einstellung der Datenbank-Umgebung (aktueller SQL-Benutzer, Standardkatalog und Standardschema) (Seite 25)

– die Syntax- und Semantikregeln der DRIVE-Anweisungen PARAMETER DYNAMIC und OPTION zur Einstellung der Datenbank-Umgebung sowie Umgehungsmöglichkeiten für die zur Zeit beim SESAM V2-Zugriff nicht unterstützte Anweisung SHOW, mit der Informationen über Metadaten von persistenten SQL-Objekten und temporären Dialog-Objekten ausgegeben werden (ab Seite 36)

– Beispiele und Beispiel-Datenbank (Seite 42)

2.1 Organisation einer SESAM V2-Datenbank am Beispiel

Dieser Abschnitt stellt anhand eines Beispiels die wichtigsten Objekte einer SESAM V2-Datenbank vor sowie die Mittel, um die Objekte zu erzeugen, zu verwalten und zu bearbeiten. Dabei wird eine Tabelle durch Migration einer SESAM V1-Datenbank erzeugt. Eine vollständige Beschreibung des Beispiels und der Migration finden Sie ab Seite 42.

2.1.1 Begriffe der logischen Organisation

Eine SESAM V2-Datenbank besteht logisch aus den folgenden SQL-Objekten:

– Katalog

  Der Katalog ist eine BS2000-Datei, die Systemtabellen zur Verwaltung der Datenbank enthält. Er wird vom Universal User als dem obersten Datenbank-Verwalter über die Utility-Anweisung CREATE CATALOG ... generiert. Ein Katalog kann mehrere Sche-

mas mit mehreren Basistabellen enthalten. Im Beispiel besteht der Katalog „PERSONALVERWALTUNG“ aus den beiden Schemas „STAMMDATEN“ und „PROJEKTDATEN“.

Eine Anwendung kann nur über einen DBH (Database Handler) auf Kataloge zugreifen (vgl. DRIVE-Programmiersprache [2], Abschnitt „Besonderheiten bei SESAM V2“). In der Konfigurationsdatei des DBH werden mit ADD-SQL-DATABASE-CATALOG-LIST die logischen und physikalischen Namen der Kataloge vereinbart, die von diesem DBH verwaltet werden sollen. Die Datenbanken müssen alle auf dem Rechner liegen, auf dem der DBH läuft. Insofern können die physikalischen Katalognamen durch einen BS2000-Kennungsnamen präfigiert werden. Die logischen Katalognamen sind einfach (max. 18 Zeichen) und müssen alle unterschiedlich sein.

– Schema

  Schema ist der Name für eine Menge von Tabellen, die logisch zusammengehören. Das Schema ist genau einem SQL-User zugeordnet, der Eigentümer des Schemas ist und damit verantwortlich für die Tabellen. Im Beispiel bilden die beiden Tabellen „MITARBEITER“ und „ABTEILUNG“ das Schema „STAMMDATEN“, die Tabelle „PROJEKT“ das Schema „PROJEKTDATEN“.

  Der Schemaname muß innerhalb eines Katalogs eindeutig sein.

– Tabelle und Spalten

  Eine Tabelle kann als eine Menge von Spalten aufgefaßt werden, die die Anwenderdaten darstellen. Auf eine Tabelle können der Eigentümer zugreifen und die SQL-User, die der Eigentümer dazu berechtigt hat (siehe GRANT-Anweisung).

  Der Tabellenname muß innerhalb eines Schemas eindeutig sein. Der vollqualifizierte Name einer Tabelle besteht aus dem Catalogname.Schemaname.Tabellenname. Außerhalb der DRIVE-Anwendung kann die Namensqualifizierung in DRIVE/WINDOWS erfolgen durch
  – Compileroptionen für statische Programme (siehe Abschnitt „OPTION CATALOG/SCHEMA/AUTHORIZATION“ auf Seite 28)
  – dynamische SET-Anweisungen für dynamische Anweisungen zur Defaulteinstellung für das Laufzeitsystems (siehe Abschnitt „SET CATALOG/SCHEMA/SESSION AUTHORIZATION“ auf Seite 26).

– Constraint

  Ein Constraint (Integritätsbedingung) besteht aus 1-n Spalten einer oder mehrerer Tabellen, die alle einer Bedingung genügen. Ein Constraint kann vom Eigentümer der Tabelle oder einem dazu privilegierten SQL-User festgelegt werden. Es gibt die Constraints:

- UNIQUE (im Beispiel „ABTEILUNG_NR“, „COMP_PERS_NR“ und „PROJEKT_NR“)
- PRIMARY KEY (siehe UNIQUE)
- CHECK überprüft, ob der Spaltenwert in einem bestimmten Wertebereich liegt oder der NULL-Wert ist.
- FOREIGN KEY stellt die referentielle Integrität zwischen Ausgangs- und Verweistabelle sicher (im Beispiel „ABT_MIT_NR“, „PROJ_MIT“, „ABT_LEITER“). Die Ausgangstabelle muß einen Constraint vom Typ UNIQUE oder PRIMARY KEY enthalten. Beim Laden der Verweistabelle (über DDL oder Utility-Monitor) wird überprüft, ob die 1-n Spalten des FOREIGN KEY Werte enthalten, die bereits in der Ausgangstabelle vorhanden sind. Hierfür braucht der SQL-User, der die Verweistabelle lädt, das Recht, die Daten des referenzierten Schemas zu lesen (GRANT ... REFERENCES).

Der Constraintname muß innerhalb eines Schemas eindeutig sein. Der vollqualifizierte Constraintname besteht aus: Catalogname.Schemaname.Constraintname.

– Index

  Zur Performancesteigerung können Indizes über 1-n Spalten einer Tabelle definiert werden. Für SESAM V2 unterstützt DRIVE/WINDOWS z.Zt. Indizes nicht in der Syntax. Indizes von Basistabellen, die z.B. im Utility-Monitor spezifiziert wurden, können jedoch in DRIVE/WINDOWS benutzt werden.

  Die Anweisung CREATE INDEX ... setzt voraus, daß das Schema festgelegt wurde und Tabellenfelder existieren. Der vollqualifizierte Indexname besteht aus: Catalogname.Schemaname.Indexname.

2.1.2 Begriffe der physikalischen Organisation

Physikalisch wird eine Datenbank durch Speichergruppen und Spaces beschrieben und mit SSL-Sprachmitteln (Storage Structure Language) verwaltet. Ein Space ist eine BS2000-Datei, in der die Datensätze und Indizes in Form von Tabellen abgelegt werden. Eine Speichergruppe faßt mehrere Spaces logisch zusammen.

Jeder SQL-User hat einen Default-Space D0user (11 Zeichen), der mit dem Katalognamen qualifiziert ist. Außerdem kann jeder SQL-User weitere Spaces einrichten, die er als Eigentümer verwaltet (catalogname.spacename).

Beispiel

Ein privilegierter User (z.B. Universal User) muß zunächst mit der GRANT-Anweisung einem SQL-User das Recht geben, eine Speichergruppe einzurichten. Nach der Anweisung CREATE STOGROUP kann dieser SQL-User dann weiteren SQL-Usern das Recht geben,

diese Speichergruppe zu benutzen (GRANT USAGE ON STOGROUP). Die folgende Tabelle zeigt einen möglichen Transaktionsablauf im Utility- Monitor zum Einrichten der physikalischen Datenbank-Organisation:

Transaktion	SQL-Anweisung
TA1	GRANT CREATE STOGROUP ON CATALOG *catalog* TO *user1*
TA2: User *user1*	CREATE STOGROUP *stogroup* ...
TA3	GRANT USAGE ON STOGROUP *stogroup* TO USER *user2*
TA4: User *user2*	CREATE SPACE ... USING STOGROUP *stogroup*

Die SSL-Sprachmittel werden von DRIVE/WINDOWS nicht unterstützt.

2.2 Datenbankaufbau und Migration von SESAM V1-Tabellen

Dieser Abschnitt zeigt Ihnen am Beispiel der SESAM V1-Tabellen „MITARBEITER“, „ABTEILUNG“ und „PROJEKT“ (siehe Abschnitt „Beispiele und Beispiel-Datenbank“ auf Seite 42), wie Sie mit dem Utility-Monitor eine SESAM V2-Datenbank aufbauen und SESAM V1-Tabellen migrieren können und wie Sie mit DRIVE-Programmen neue SESAM V2-Tabellen erzeugen und Rechte vergeben können.

Jeder Zugriff (Lesen, Schreiben) auf ein SQL-Objekt benötigt einen Berechtigungsschlüssel. Dieser Schlüssel entspricht dem SQL-User-Namen. Jeder SQL-User darf nur auf bestimmte SQL-Objekte zugreifen. Dies sind zunächst alle Objekte, deren Eigentümer er ist, weiterhin alle Objekte, deren Eigentümer ihm Lese- oder Schreibrecht gewährt haben.

V2-Datenbank einrichten und V1-Datenbanken migrieren (Utility-Monitor)

Um eine Datenbank aufzubauen, muß der Universal User (im Beispiel „SYSTEMVERWALTER“), der alle Rechte hat, daher die folgenden Arbeitsschritte im Utility-Monitor ausführen (vgl. SESAM-Handbuch Utility-Monitor [21], Kapitel 5):

1. im Startmenü die Funktion Konfiguration auswählen (CNF) und die SQL-Umgebung mit folgenden Angaben festlegen:

   SEE-AUTHID : SYSTEMVERWALTER

   SEE-CATALOG : PERSONALVERWALTUNG

   SEE-SCHEMA : STAMMDATEN

2. im Startmenü die Funktion Anweisungsdatei (IFP) auswählen und die folgenden Anweisungsdateien (siehe Seite 47) angeben zum Einrichten des Katalogs, der Systemzugänge, der Schemas und zum Definieren der SQL-User:

UTI.PERSONALVERWALTUNG.CATALOG

UTI.PERSONALVERWALTUNG.USER

UTI.PERSONALVERWALTUNG.SYSTEMUSER

UTI.PERSONALVERWALTUNG.SCHEMA

3. im Startmenü die Funktion Migrieren (MIG) auswählen und die folgenden Angaben zur Migration der SESAM V1-Datenbank in eine SESAM V2-Tabelle machen:

   MIGRATE DATABASE : $Kennung.MITARBEITER.DB-SIB.0006

   WITH INDEX (y/n) : Y

   TO TABLE : MITARBEITER

   Die Datensicherung (DB-SIB) der SESAM V1-Datenbank wird als SESAM V2-Tabelle im Schema „STAMMDATEN“ abgelegt.

   Um die Datensicherung MITARBEITER.DB-SIB.0006 zu migirieren, müssen über den Utility-Monitor der SQL-User, der Katalog und das Schema eingestellt werden (siehe Arbeitsschritt 1).

   Der SQL-User „SYSTEMVERWALTER“, der das Migrate-Recht hat, migriert die Datensicherung „MITARBEITER“ in das Schema PERSONALVERWALTUNG.STAMMDATEN.

SQL-Tabelle „ABTEILUNG“ erzeugen (DRIVE-Programm)

Das DRIVE-Programm „DRI.TABLE.ABTEILUNG“ erzeugt im Schema „STAMMDATEN“, dessen Eigentümer „PERSONALLEITER“ ist, die SQL-Tabelle „ABTEILUNG“ mit dem Primärschlüssel KEY_ABT_NR (siehe Seite 50).

Zugriffsrechte vergeben und Fremdschlüssel definieren (DRIVE-Programm)

Der SQL-User „PERSONALLEITER“ kann als Eigentümer Privilegien an andere SQL-User vergeben (GRANT-Anweisung), um ihnen den Zugriff auf die Daten zu ermöglichen. Im Beispiel muß der SQL-User „PERSONALLEITER“ dem SQL-User „PROJEKTLEITER“ die Zugriffsrechte auf die Daten des Schemas „STAMMDATEN“ mit den Tabellen „MITARBEITER“ und „ABTEILUNG“ geben. Der SQL-User „PROJEKTLEITER“ vergibt die Zugriffsrechte auf die Tabelle „PROJEKT“ im Schema „PROJEKTDATEN“, dessen Eigentümer er ist. Insbesondere hat damit „PERSONALLEITER“ für „PROJEKT“ das Referenzrecht (Privileg REFERENCES). „PERSONALLEITER“ definiert einen Fremdschlüsselverweis von „MITARBEITER“ auf den Primärschlüssel von „PROJEKT“ und einen von „MITARBEITER“ auf den Primärschlüssel von „ABTEILUNG“ (siehe Seite 52).

2.2.1 DRIVE-Anforderungen an SESAM-Migration

Damit bestehende DRIVE-New-Style-Programme und DRIVE-Old-Style-Prozeduren, die auf SESAM-Datenbanken der SQL V1 oder der V14 zugreifen, nach der Datenbank-Migration zur SESAM V2 möglichst ohne Änderungen ablaufen können, müssen folgende Regeln beachtet werden:

– Alle Datenbanksicherungen (DB-SIBs), auf die die DRIVE-Anwendung zugreift, werden in dasselbe Schema migriert.

– Datenbanken, die von DRIVE-Programmen im New-Style mit SQL-DML-Anweisungen bearbeitet werden, werden als SQL-Tabellen migriert. Datenbanken, die von DRIVE-Prozeduren im Old-Style oder Mischbetrieb mit CALL-DML-Anweisungen bearbeitet werden, werden als CALL-DML-Tabellen migriert. Hat eine Datenbank einen Kennwortkatalog (PK-SIB) und soll sie in eine CALL-DML-Tabelle migriert werden, so muß der Kennwortkatalog mitmigriert werden.

– Es werden Systemzugänge für die DRIVE-Anwendung eingerichtet (z.B. UTM-Betrieb oder TIAM-Betrieb).

– Wenn der DRIVE-SQL-User nicht Eigentümer des Schemas ist, müssen ihm mit der GRANT-Anweisung alle erforderlichen Privilegien zugewiesen werden .

– Der Katalog oder die CALL-DML-Tabellen werden in die Konfigurationsdatei des SESAM-DBH eingetragen (DBH-Startanweisungen ADD-SQL-DATABASE-CATALOG-LIST und ADD-OLD-TABLE-CATALOG-LIST).

Werden die o.g. Regeln bei der Migration beachtet, brauchen bestehende DRIVE-Programme im New-Style, Old-Style oder Mischbtrieb nicht geändert zu werden. Die SQL-Tabellen werden mit der Anweisung PARAMETER DYNAMIC CATALOG/SCHEMA/ AUTHORIZATION bekanntgemacht. Für Old-Style-Programme oder Mischbetrieb mit Zugriff auf CALL-DML-Tabellen sind keine Einstellungen der Datenbank-Umgebung mit PARAMETER DYNAMIC nötig. Eventuell benötigte Kennwörter können mit der PERMIT-Anweisung aus dem New-Style in den Old-Style oder Mischbetrieb weitergereicht werden.

2.3 DRIVE-Programmzugriff auf SESAM

DRIVE/WINDOWS unterstützt die SQL-Norm (ISO/IEC 9075:1992), und zwar vollständig den Entry Level, weitgehend den Intermediate Level und wichtige Teile des Full Level (siehe Database Language SQL [47]). Der Grad der Unterstützung wird im wesentlichen durch den Funktionsumfang der SESAM/SQL-Version 2.x bestimmt.

Der Zugriff auf SESAM-Datenbanken ist möglich von

- **Old-Style-Prozeduren**,

  deren SQL-Sprachumfang identisch ist mit dem der DRIVE-Version 5.1 (vgl. Handbuch DRIVE V5.1 - Teil 2: Lexikon [15]). Sie greifen über die CALL-DML-Schnittstelle auf CALL-DML-Tabellen zu (vgl. die Anweisungen MIGRATE und CREATE TABLE).

- **New-Style-Programmen**,

  deren SQL-Sprachumfang wesentlich umfangreicher ist, nämlich

  – die normkonformen Anweisungen von SESAM V2 nahezu vollständig abdeckt (siehe nächster Abschnitt),

  – bereits die wichtigsten Erweiterungen von SESAM V2 zur SQL-Norm umfaßt (siehe nächster Abschnitt),

  – komfortable DRIVE-spezifische Erweiterungen zur SQL-Norm beinhaltet (vgl. DRIVE- Programmiersprache [2], Kapitel „Datenbank-Unterstützung“)

  – über die EXECUTE-Anweisung eine transparente Möglichkeit zur Formulierung dynamischer SQL-Anweisungen anbietet.

  Sie greifen über die statische SQL-Schnittstelle (ICSQLE) oder über die dynamische SQL-Schnittstelle (ESQL/COBOL) auf CALL-DML- und SQL-Tabellen zu.

Die Einsatzmöglichkeit dieser Programmstile hängt von der jeweiligen DRIVE/WINDOWS-Betriebsart ab:

– im Old-Style-Betrieb können nur Old-Style-Prozeduren (mit der DO-Anweisung) ausgeführt werden

– im New-Style-Betrieb können nur New-Style-Programme (mit den Anweisungen DO, CALL und ENTER) ausgeführt werden

– im Mischbetrieb können Old-Style-Prozeduren und New-Style-Programme ausgeführt werden, und zwar kann ein New-Style- Programm eine Old-Style-Prozedur mit DO oder CALL aufrufen (vgl. DRIVE- Programmiersprache [2], Kapitel „Integration von Old-Style-Prozeduren“)

Eine **SQL-Anweisung** in einem New-Style-Programm heißt

– **statisch**, wenn sie explizit in der Source codiert und damit übersetzbar ist, und

– **dynamisch**, wenn sie erst im Zuge der Ausführung einer EXECUTE-Anweisung generiert, übersetzt und ausgeführt wird (vgl. DRIVE-Programmiersprache [2], Kapitel „Programmierlogik“ und DRIVE-Lexikon [3], EXECUTE-Anweisung).

Die deklarative DRIVE-Anweisung DECLARE VARIABLE ... LIKE TABLE/CURSOR ... gilt auch als statische SQL-Anweisung. Cursoranweisungen, die sich auf variable Cursor beziehen, gelten als dynamisch. Die ausführbare DRIVE-Anweisung CYCLE *cursor* INTO... gilt als statische SQL-Anweisung, wenn *cursor* statisch ist, und als dynamische SQL-Anweisung, wenn *cursor* variabel ist.

Ein **New-Style-Programm** heißt

- **statisch**, wenn es mindestens eine ausführbare statische SQL-Anweisung außer COMMIT WORK und ROLLBACK WORK und keine dynamischen SQL-Anweisungen enthält, und
- **dynamisch**, wenn es keine ausführbare statischen SQL-Anweisungen außer COMMIT WORK und ROLLBACK WORK enthält.
- Alle anderen New-Style-Programme heißen **allgemein**.

Im folgenden werden Regeln und Empfehlungen für statische und dynamische SQL-Anweisungen und Programme gegeben.

Im **Dialog-Modus** eingegebene SQL-Anweisungen werden von DRIVE/WINDOWS wie dynamische Programm-Anweisungen verarbeitet, so daß für sie auch die entsprechenden Regeln und Empfehlungen gelten, soweit sinnvoll.

Während beim SESAM V1-Zugriff von DRIVE/WINDOWS statische und dynamische SQL-Anweisungen über dieselbe Schnittstelle auf SESAM zugreifen, verwendet DRIVE/WINDOWS beim SESAM V2-Zugriff aus Performancegründen zwei verschiedene Schnittstellen (s.o.).

2.3.1 SQL-Sprachumfang im New-Style

DRIVE/WINDOWS unterstützt im New-Style für den DB-Server SESAM/SQL V2 die folgenden Schnittstellen (vgl. die SESAM-Handbücher [18] und [20] zur verwendeten Klassifikation):

- Utility-Anweisungen (z.B. CREATE CATALOG und MIGRATE) über den Utility-Monitor von SESAM (vgl. SESAM-Handbücher [19] und [21])
- UDL (User Definition Language) zur Verwaltung von Benutzereinträgen (z.B. CREATE USER und CREATE SYSTEM_USER) ebenfalls über den Utility-Monitor
- DDL (Data Definition Language) zur Schemadefinition und -verwaltung (z.B. CREATE SCHEMA und CREATE TABLE), deren sämtliche Anweisungen bereits im DRIVE-Sprachumfang enthalten sind mit nur geringen Einschränkungen im Funktionsumfang (z.B. noch keine CALL-DML-Tabellen und kein kaskadierendes Löschen)
- SSL (Storage Structure Language) zur Verwaltung der Speicherstruktur (z.B. CREATE INDEX) über den Utility-Monitor

– alle Anweisungen zur Transaktionsverwaltung
– alle Anweisungen zur Sessionsteuerung
– DML (Data Manipulation Language) zum Abfragen und Ändern von Daten (insbesondere INSERT, UPDATE, DELETE und DECLARE CURSOR), deren sämtliche Anweisungen bereits im DRIVE-Sprachumfang enthalten sind, ohne jegliche Einschränkung gegenüber dem Funktionsumfang von SESAM/SQL V2.0 und mit nur geringen Einschränkungen gegenüber neuen Funktionen von SESAM/SQL V2.1 (z.B. bei SQL-Ausdrücken und Tabellenangaben)
– CALL-DML für CALL-DML-Tabellen über den Mischbetrieb (vgl. DRIVE V5.1-Handbücher [14] und [15]).

Eine New-Style-Transaktion darf entweder nur DML-Anweisungen oder keine DML-Anweisungen enthalten. Eine SQL-Anweisung wird von SESAM nur dann ausgeführt, wenn die SQL-Umgebung ihrer Transaktion dies erlaubt. Diese SQL-Umgebung besteht aus einem SQL-User, dessen Name Berechtigungsschlüssel heißt, aus einem zugeordneten Standardkatalog und aus einem zugeordneten Standardschema. Der SQL-User ist berechtigt, auf die Daten bzw. Metadaten der Transaktion zuzugreifen, wenn er entweder ihr Eigentümer ist oder vom jeweiligen Eigentümer befugt wurde (siehe GRANT-Anweisung).

Ein New-Style-Programm kann eine Old-Style-Transaktion initiieren, indem es eine Old-Style-Prozedur (mit DO oder CALL) aufruft. Dies ist nur außerhalb von New-Style-Transaktionen erlaubt. Die Old-Style-Transaktion benötigt zur Ausführung ihrer CALL-DML-Anweisungen keine (New-Style-) SQL-Umgebung, sondern nur ggf. ein Paßwort, das im New-Style mit PERMIT eingegeben werden muß und von DRIVE/WINDOWS an die Old-Style-Transaktion weitergereicht wird.

2.3.2 Statische Programme

Die SQL-Anweisungen eines statischen (New-Style-) Programms entsprechen in der SESAM-Terminologie den vorübersetzbaren Anweisungen. Jede DRIVE-Übersetzung eines statischen Programms erfolgt als SESAM-Vorübersetzung mit Datenbankkontakt. Folglich muß SESAM für den Compilevorgang zur Verfügung stehen, d.h. der DRIVE/WINDOWS zugeordnete DBH muß gestartet sein. Außerdem müssen Angaben vorliegen zu
– AUTHORIZATION (Berechtigungsschlüssel),
– CATALOG (Standardkatalog) und
– SCHEMA (Standardschema).

Diese Angaben können folgendermaßen vorgenommen werden:
– DRIVE-übersetzungsoptionen (OPTION-Anweisung im Programm oder, höherprior, OPTION-Klausel der COMPILE-Anweisung) oder
– DRIVE-Parametereinstellungen (Anweisung PARAMETER DYNAMIC, vgl. Abschnitt „Datenbank-Umgebung einstellen“ auf Seite 25).

Hieraus ergeben sich folgende Eigenschaften für das Transaktionsprofil (im folgenden abgekürzt TA-Profil) eines statischen Programms ohne CALL-Anweisungen:

– enthält es n COMMIT-Anweisungen, so führt es zum Ablaufzeitpunkt bis zu n Transaktionen aus; ist n = 0, so kann es nur an einer Transaktion eines rufenden Programms teilhaben (vgl. Abschnitt „Programmkommunikation“ auf Seite 17)

– alle diese Transaktionen sind genau einem SQL-User zugeordnet, nämlich dem über OPTION AUTHORIZATION angegebenen oder dem zum Übersetzungszeitpunkt über PARAMETER DYNAMIC voreingestellten

– der Zugriff auf alle nicht voll qualifizierten Datenfelder und Metadaten (z.B. Integritätsbedingungen) erfolgt über die Programmqualifikation, d.h. die über SCHEMA und CATALOG wirksamen Standardwerte für Schema- und Katalogname

Die wirksamen Werte von AUTHORIZATION, CATALOG und SCHEMA können der Übersetzungsliste des Programms entnommen werden.

2.3.3 Dynamische Programme

Die SQL-Anweisungen eines dynamischen (New-Style-) Programms entsprechen in der SESAM-Terminologie den präparierbaren Anweisungen. Die Präparierung erfolgt zum Ausführungszeitpunkt. Zum Übersetzungszeitpunkt gibt es daher keinen SESAM-Kontakt. Die aktuelle SQL-Umgebung, d.h. aktueller Berechtigungsschlüssel, Standardkatalogname und Standardschemaname, wird dynamisch während des Ablaufs durch SET SESSION AUTHORIZATION/CATALOG/SCHEMA-Anweisungen eingestellt. Falls das Programm keine solchen SET-Anweisungen enthält, gilt die DRIVE-Standardwert-Einstellung gemäß der letzten entsprechenden PARAMETER DYNAMIC-Anweisung (vgl. Abschnitt „Datenbank-Umgebung einstellen“ auf Seite 25).

SET SESSION AUTHORIZATION und PARAMETER DYNAMIC AUTHORIZATION sind nur außerhalb von Transaktionen erlaubt. Der Wechsel des Standardkatalogs oder des Standardschemas ist immer möglich. Der SQL-User kann daher pro Transaktion gewechselt werden, Standardkatalog und Standardschema können sogar pro Anweisung gewechselt werden. Jede Transaktion hat daher „ihren“ SQL-User und kann auf genau die Kataloge und SQL-Objekte zugreifen, auf die „ihr“ SQL-User zugreifen darf.

Hieraus ergeben sich folgende Eigenschaften für das TA-Profil eines dynamischen Programms ohne CALL-Anweisungen:

– enthält es n COMMIT-Anweisungen, so führt es zum Ablaufzeitpunkt bis zu n Transaktionen aus; ist n = 0, so kann es nur an einer Transaktion eines rufenden Programms teilhaben (vgl. Abschnitt „Programmkommunikation“ auf Seite 17)

– jede dieser Transaktionen ist genau einem SQL-User zugeordnet, nämlich dem zuletzt über SET SESSION AUTHORIZATION angegebenen oder zuletzt (zum Ausführungszeitpunkt) über PARAMETER DYNAMIC voreingestellten; dieser Berechtigungsschlüssel kann vom rufenden Programm oder aus dem Dialog-Modus vererbt sein

– der Zugriff auf alle nicht voll qualifizierten Datenfelder und Metadaten erfolgt über die aktuell wirksamen Standardwerte für SCHEMA und CATALOG, d.h. entsprechend der letzten jeweiligen SET- bzw. PARAMETER DYNAMIC-Anweisung. Jeder Standardwert kann vom rufenden Programm oder aus dem Dialog-Modus vererbt sein.

Die jeweils wirksamen Werte von AUTHORIZATION, CATALOG und SCHEMA können zum Entwicklungszeitpunkt im Debug-Modus durch Ablaufverfolgung kontrolliert und durch Debugaktionen in der Debugliste mitgeschnitten werden (Programm mit Ablaufverfolgung starten und Debugaktionen TRACE und DISPLAY LIST an jeder entsprechenden SET- und PARAMETER DYNAMIC-Anweisung sowie an jeder Transaktionsanweisung hinterlegen (vgl. DRIVE-Lexikon [3], Anweisungen DEBUG, AT und TRACE).

2.3.4 Programmkommunikation

DRIVE-Programme kommunizieren miteinander über die CALL-Anweisung. Hierbei sind folgende Fälle zu unterscheiden:

1. statisches (New-Style-)Programm ruft statisches Programm
2. dynamisches (New-Style-)Programm ruft dynamisches Programm
3. statisches Programm ruft dynamisches Programm
4. dynamisches Programm ruft statisches Programm
5. allgemeines (New-Style-)Programm ruft allgemeines Programm
6. New-Style-Programm ruft Old-Style-Prozedur

Diese Fälle werden im folgenden anhand von Beispielen erläutert.

Die Fallunterscheidung ist für alle Plattformen (BS2000, SINIX, MS-Windows) vollständig. Auf der Plattform SINIX heißen die im BS2000 möglichen Programme DRIVE-Alpha-Programme. Diese kommunizieren miteinander ebenfalls nur über die CALL-Anweisung. Dagegen kommunizieren DRIVE-Window-Programme, d.h. Programme mit grafischer Bedienoberfläche, zusätzlich über ADD/NEXT/NEW WINDOW-Anweisungen und Ereignisse miteinander. Während bei CALL der Ablaufteil (Body) eines DRIVE-Window-Programms abgearbeitet wird, werden bei ADD/NEW/NEXT WINDOW der Initialisierungsblock und bei einem Ereignis der entsprechende Ereignisblock des zugehörigen Skripts abgearbeitet (siehe ON-Anweisung im DRIVE-Lexikon [3]).

Auf der Plattform MS-Windows gibt es nur DRIVE-Window-Programme. Hinsichtlich der TA-Profile verhalten sich jedoch DRIVE-Window-Programme genauso wie DRIVE-Alpha-Programme. In den folgenden Beispielen kann jedoch jede CALL-Anweisung auch eine ADD/NEW/NEXT WINDOW-Anweisung oder eine ON-Anweisung sein und entsprechend jeder Abschnitt nicht nur ein Ablaufteil (Body), sondern auch ein Initialisierungsblock oder ein Ereignisblock.

Eine (New-Style-)Transaktion werde durch die drei DRIVE-Programme P1, P2 und P3 realisiert, wobei P2 von P1 und P3 von P2 mit CALL gerufen werden. P2 und P3 dürfen keine Transaktionsanweisung enthalten, P1 enthält nach dem Aufruf von P2 ein COMMIT WORK. Durch diese CALL-Aufrufkette ergeben sich die folgenden fünf Programmabschnitte, von denen jeder SQL-Anweisungen enthält:

```
PROC P1;
...
Abschnitt 1 (enthält keine TA-Anweisung)
...
CALL P2;
   PROC P2;
   ...
   Abschnitt 2 (enthält keine TA-Anweisung)
   ...
   CALL P3;
      PROC P3;
      ...
      Abschnitt 3 (enthält keine TA-Anweisung)
      ...
      END PROC;
   ...
   Abschnitt 4 (enthält keine TA-Anweisung)
   ...
   END PROC;
...
```

```
Abschnitt 5 (enthält eine COMMIT WORK-Anweisung)
...
END PROC;
```

Fall 1 (statisch ruft statisch)

P1, P2 und P3 sind statische Programme.

CALL P2 und CALL P3 werden nur ausgeführt, wenn P2 und P3 schon als Zwischencode oder Objektcode vorliegen. Ist P1 selbst Teil einer CALL-Kette (also nicht mit DO oder ENTER aufgerufen), so gilt dies auch für P1, falls bei CALL P1 eine Transaktion offen ist.

Für jedes der Programme P1, P2 und P3 gelten die im vorletzten Abschnitt „Statische Programme“ formulierten Regeln. Insbesondere hat jedes Programm „seinen“ SQL-User, „seinen“ Standardkatalog und „sein“ Standardschema gemäß Übersetzungsliste.

1. Wird P1 selbst nicht mit CALL gerufen, so gilt bei der Ausführung von P1 folgendes: P1 startet im Abschnitt 1 eine neue Transaktion und beendet sie im Abschnitt 5 jeweils mit „seiner“ SQL-Umgebung; P2 setzt diese Transaktion in den Abschnitten 2 und 4 jeweils mit „seiner“ SQL-Umgebung fort; P3 setzt diese Transaktion im Abschnitt 3 mit „seiner“ SQL-Umgebung fort. Dasselbe gilt, wenn P1 zwar mit CALL gerufen wird, bei CALL P1 aber keine Transaktion offen ist.
2. Wird dagegen P1 mit CALL gerufen und ist zu diesem Zeitpunkt eine Transaktion offen, so setzt P1 diese Transaktion mit „seiner“ SQL-Umgebung fort, d.h. P1 spielt dann die Rolle von P2, aber mit der zusätzlichen Eigenschaft, daß P1 die vererbte Transaktion beendet.

Hieraus ergeben sich folgende Eigenschaften für das TA-Profil bei Ausführung von P1:

– wird P1 nicht mit CALL gerufen, sondern mit DO oder ENTER, so liegt eine Transaktion mit bis zu drei SQL-Usern, bis zu drei Standardkatalogen und bis zu drei Standardschemas vor

– ist P1 selbst Teil einer CALL-Kette, so hängt das TA-Profil davon ab, ob beim Aufruf von P1 eine Transaktion offen ist oder nicht.

  Ist keine Transaktion offen, so gilt das für DO P1 Gesagte. Ist dagegen eine Transaktion offen, so setzt P1 diese „fremde“ Transaktion im Abschnitt 1 fort und beendet sie im Abschnitt 5, jeweils mit „seiner“ SQL-Umgebung; diese fortgesetzte Transaktion kann mehr als drei SQL-Umgebungen haben

– kehrt P1 zu seinem Rufer zurück (nicht bei Folge-DO und ENTER), so nimmt es keine SQL-Umgebung mit; beim Rufer von P1 gilt weiter die SQL-Umgebung, die vor dem Aufruf von P1 aktuell war

Fall 2 (dynamisch ruft dynamisch)

P1, P2 und P3 sind dynamische Programme.

P1, P2 und P3 brauchen nicht als Zwischencode oder Objektcode vorzuliegen. Da dynamische Anweisungen ohnehin erst zum Ablaufzeitpunkt generiert und übersetzt werden können, hat das Vorliegen von Zwischencode oder Objektcode keine Auswirkungen auf die Performance der SQL-Anweisungen (Ausnahme: variable Cursor).

Für jedes der Programme P1, P2 und P3 gelten die im Abschnitt „Dynamische Programme“ formulierten Regeln. Insbesondere kann der SQL-User nicht innerhalb einer Transaktion gewechselt werden. Hingegen können Standardkatalog und Standardschema von Anweisung zu Anweisung (soweit sinnvoll) gewechselt werden.

1. Wird P1 selbst nicht mit CALL gerufen, so gilt bei der Ausführung von P1 folgendes:

   P1 startet im Abschnitt 1 eine neue Transaktion mit einem SQL-User, den es selbst mit SET SESSION AUTHORIZATION einstellt oder der gemäß der aktuellen PARAMETER DYNAMIC AUTHORIZATION-Voreinstellung eingestellt ist. P1 beendet diese Transaktion im Abschnitt 5 mit demselben SQL-User; P2 bzw. P3 setzen diese Transaktion in den Abschnitten 2 und 4 bzw. im Abschnitt 3 jeweils mit demselben SQL-User fort. Dasselbe gilt, wenn P1 zwar mit CALL gerufen wird, bei CALL P1 aber keine Transaktion offen ist (siehe Fall 3).

2. Wird dagegen P1 mit CALL gerufen und ist zu diesem Zeitpunkt eine Transaktion offen, so setzt P1 diese Transaktion mit dem aktuellen Berechtigungsschlüssel fort, d.h. P1 spielt dann die Rolle von P2, aber mit der zusätzlichen Eigenschaft, daß P1 die vererbte Transaktion beendet.

Hieraus ergeben sich folgende Eigenschaften für das TA-Profil bei Ausführung von P1:

– wird P1 nicht mit CALL gerufen, so liegt eine Transaktion mit einem SQL-User und „vielen“ Standardkatalogen und -schemas vor

– ist P1 selbst Teil einer dynamischen CALL-Kette, so hängt das TA-Profil davon ab, ob beim Aufruf von P1 eine Transaktion offen ist oder nicht.

  Ist keine Transaktion offen, so initiiert P1 in Abschnitt 1 eine „eigene“ Transaktion und beendet sie in Abschnitt 5, jeweils mit „seinem“ SQL-User.

  Ist eine Transaktion offen, so setzt P1 diese „fremde“ Transaktion im Abschnitt 5 fort, jeweils mit „ihrem“ SQL-User. Die initiierte und die fortgesetzte Transaktion können jeweils nur einen SQL-User haben

– wurde P1 mit CALL gerufen, so nimmt es bei der Rückkehr zu seinem Rufer die aktuelle SQL-Umgebung mit. Beim Rufer von P1 gilt daher ein anderer aktueller SQL-User, wenn bei CALL P1 keine Transaktion offen war und P1 eine neue SET SESSION AUTHORIZATION-Anweisung abgesetzt hat.

Beim Rufer von P1 gilt ein anderer Standardkatalog bzw. ein anderes Standardschema, wenn P1 oder P2 oder P3 eine neue SET CATALOG- bzw. eine neue SET SCHEMA-Anweisung abgesetzt hat

– wurde P1 mit DO gerufen, so stellt DRIVE/WINDOWS bei der Rückkehr in den Dialog-Modus wieder die SQL-Umgebung gemäß der letzten PARAMETER DYNAMIC AUTHORIZATION/CATALOG/SCHEMA-Anweisung her

Fall 3 (statisch ruft dynamisch)

P1 und P3 sind statisch, und P2 ist dynamisch.

1. Wird P1 selbst nicht mit CALL gerufen, so gilt bei der Ausführung von P1 folgendes:

   P1 startet im Abschnitt 1 eine neue Transaktion und beendet sie im Abschnitt 5 jeweils mit „seiner“ SQL-Umgebung.

   P2 setzt diese Transaktion in den Abschnitten 2 und 4 mit dem SQL-User der letzten SET SESSION AUTHORIZATION- oder PARAMETER DYNAMIC AUTHORIZATION-Anweisung (vor Aufruf von P1) fort.

   P3 setzt diese Transaktion im Abschnitt 3 mit „seiner“ SQL-Umgebung fort.

   Dasselbe gilt, wenn P1 mit CALL gerufen wird, aber bei CALL P1 keine Transaktion offen ist.

2. Wird dagegen P1 mit CALL gerufen und ist zu diesem Zeitpunkt eine Transaktion offen, so setzt P1 diese Transaktion mit „seiner“ SQL-Umgebung fort. Der in P2 gültige SQL-User wurde außerhalb dieser vererbten Transaktion eingestellt.

Hieraus ergeben sich folgende Eigenschaften für das TA-Profil bei Ausführung von P1:

– wird P1 nicht mit CALL gerufen oder ist bei CALL P1 keine Transaktion offen, so liegt eine Transaktion mit bis zu drei SQL-Usern und „vielen“ Standardkatalogen und -schemas vor

– ist bei CALL P1 eine Transaktion offen, so setzt P1 in Abschnitt 1 diese Transaktion fort und beendet sie in Abschnitt 5, jeweils mit „seinem“ SQL-User. Diese Transaktion kann mehr als drei SQL-User und „viele“ Standardkataloge und -schemas haben.

Fall 4 (dynamisch ruft statisch)

P1 und P3 sind dynamisch, und P2 ist statisch.

1. Wird P1 selbst nicht mit CALL gerufen, so gilt bei der Ausführung von P1 folgendes:

   P1 startet im Abschnitt 1 eine neue Transaktion mit einem SQL-User, den es selbst mit SET SESSION AUTHORIZATION einstellt oder der gemäß der aktuellen PARAMETER DYNAMIC AUTHORIZATION-Voreinstellung eingestellt ist. P1 beendet diese Transaktion im Abschnitt 5 mit demselben SQL-User.

   P2 setzt diese Transaktion in den Abschnitten 2 und 4 mit „seinem“ SQL-User fort, und P3 setzt sie im Abschnitt 3 mit dem SQL-User von P1 fort. Dasselbe gilt, wenn P1 zwar mit CALL gerufen wird, bei CALL P1 aber keine Transaktion offen ist.

2. Wird dagegen P1 mit CALL gerufen und ist zu diesem Zeitpunkt eine Transaktion offen, so setzt P1 diese Transaktion mit dem aktuellen Berechtigungsschlüssel fort und beendet die vererbte Transaktion auch mit diesem SQL-User.

Hieraus ergeben sich folgende Eigenschaften für das TA-Profil bei Ausführung von P1:

– wird P1 nicht mit CALL gerufen oder ist bei CALL P1 keine Transaktion offen, so liegt eine Transaktion mit bis zu zwei SQL-Usern und „vielen“ Standardkatalogen und -schemas vor

– ist bei CALL P1 eine Transaktion offen, so setzt P1 in Abschnitt 1 diese Transaktion fort und beendet sie in Abschnitt 5, jeweils mit dem aktuellen Berechtigungsschlüssel. Diese Transaktion kann mehr als zwei SQL-User und „viele“ Standardkataloge und -schemas haben.

Fall 5 (allgemein ruft allgemein)

P1, P2 und P3 sind allgemeine Programme.

Während in den Fällen 1 bis 4 jedem Programm genau ein SQL-User zugeordnet ist, kann nun jedes Programm einen „statischen“ und einen gemeinsamen „dynamischen“ SQL-User haben. Entsprechend allgemeiner und unübersichtlicher wird das TA-Profil.

Fall 6 (New-Style ruft Old-Style)

Ruft ein New-Style-Programm P4 eine Old-Style-Prozedur P5 mit DO oder CALL, so darf keine New-Style-Transaktion offen sein. Bei Rückkehr aus dem Old-Style in den New-Style (rufendes Programm oder Dialog-Modus) darf außerdem keine Old-Style-Transaktion offen sein.

New-Style-Transaktionen ignorieren Old-Style-Paßwörter und Old-Style-Transaktionen ignorieren New-Style-SQL-User. Die TA-Profile von P4 und P5 sind daher disjunkt.

– Das TA-Profil von P4 erhält man, indem man in P4 die CALL-Anweisung durch ein COMMIT WORK ersetzt. Für P4 gilt dann das in den Fällen 1 bis 5 für P1 Gesagte.

– Eine Old-Style-Prozedur kann nur mit einem Paßwort arbeiten. Das TA-Profil einer Old-Style-Prozedur entspricht daher dem eines statischen New-Style-Programms.

2.3.5 Programmierempfehlungen

Die folgenden Empfehlungen erhöhen die Übersichtlichkeit der DRIVE-Programme und der für sie wirksamen SQL-Umgebung:

– möglichst je DRIVE-Sitzung (TIAM-Session oder UTM-Vorgang) nur einen SQL-User verwenden und ihn über PARAMETER DYNAMIC AUTHORIZATION als Standardwert einstellen (Single-SQL-User-Sitzung). DRIVE/WINDOWS kann entsprechend konfiguriert werden (vgl. DRIVE-Programmiersystem [1], Kapitel „Datenschutz“)

– möglichst viele statische und dynamische Programme, möglichst wenige allgemeine Programme entwickeln

– dynamische Programme ohne PARAMETER DYNAMIC AUTHORIZATION/CATALOG/SCHEMA-Anweisungen entwickeln, und umgekehrt Programme mit diesen PARAMETER DYNAMIC-Anweisungen ohne SQL-Anweisungen entwickeln

– dynamische Programme als Single-SQL-User-Programme konzipieren, d.h. keine SET SESSION AUTHORIZATION-Anweisung (Vererbung des SQL-Users durch den Rufer) oder nur eine solche Anweisung und zwar vor der logisch ersten SQL-Anweisung

– das TA-Profil von allgemeinen Programmen festlegen durch eine OPTION AUTHORIZATION-Anweisung und dynamische SET SESSION AUTHORIZATION-Anweisungen (Multi-SQL-User-Programme)

2.3.6 Inkompatibilitäten

Wenn Sie mit dem Datenbanksystem SESAM/SQL V1.1 arbeiten, ergeben sich keine Unterschiede beim Zugriff von DRIVE/WINDOWS in den verschiedenen Versionen. Beim Zugriff auf das Datenbanksystem SESAM/SQL V2 ergeben sich die folgenden drei Unterschiede zwischen DRIVE/WINDOWS V2.x und früheren DRIVE-Versionen (DRIVE/WINDOWS V1.x oder DRIVE V6.x):

- Statische SQL-Objekte können nicht in dynamischen SQL-Anweisungen referenziert werden:
  - Wird ein Cursor statisch deklariert und dynamisch ausgeführt, z.B.

    ```
    DECLARE C1 CURSOR FOR SELECT SCHLUESSEL FROM FIRMA;
    EXEC 'OPEN C1';
    ```

    so gibt DRIVE/WINDOWS V2.x zur Ausführung die folgende Meldung aus:

    DRI0033 DYNAMISCHE ANWEISUNG UNZULAESSIG.

    Die Systemvariable &ERROR enthält 'SYNTAX ERROR'.
  - Wird ein temporärer View statisch deklariert und dynamisch referenziert, z.B.

    ```
    CREATE TEMPORARY VIEW V1 AS SELECT SCHLUESSEL FROM FIRMA;
    EXEC 'INSERT INTO V1 VALUES (''HUGO01'')';
    ```

    so gibt DRIVE/WINDOWS V2.x zur Ausführung ebenfalls die Meldung DRI0033 aus und belegt &ERROR mit 'SYNTAX ERROR'.
- In Programmen dynamisch definierte temporäre Views sollen vor Verlassen des Programms, in dem sie definiert wurden, mit DROP TEMPORARY VIEW ... oder am Ende der DRIVE-Anwendung mit DROP TEMPORARY VIEWS explizit gelöscht werden. Andernfalls geschieht folgendes:
  - Am Ende der Anwendung, d.h. beim Übergang in den Dialog-Modus oder in die SPU, wird die folgende Meldung ausgegeben:

    DRI0488 NOCH VORHANDENE DYNAMISCHE TEMPORAERE VIEWS WURDEN FREIGEGEBEN
  - Am Ende des Programms durch Aufrufen eines Folgeprogramms mit DO wird das Programm mit der Meldung DRI0488 abgebrochen, d.h., das Folgeprogramm wird nicht ausgeführt.

    i Eigentümer eines dynamisch definierten temporären Views ist derjenige SQL-User, der zum Zeitpunkt der Ausführung der dynamischen CREATE TEMPORARY VIEW-Anweisung aktuell ist (zuletzt ausgeführte SET SESSION AUTHORIZATION- oder PARAMETER DYNAMIC AUTHORIZATION-Anweisung). Ist zum Zeitpunkt der Ausführung der

DROP TEMPORARY VIEW(S)-Anweisung ein anderer SQL-User aktuell, so kann der dynamische temporäre View nicht gelöscht werden, da dies nur der Eigentümer darf.

- Wird in offener Transaktion mit CALL ein Programm gerufen, das statische SQL-Anweisungen außer COMMIT WORK und ROLLBACK WORK enthält, so muß es bereits als Zwischencode oder Objektcode vorliegen. Andernfalls gibt DRIVE/WINDOWS die folgende Meldung aus und bricht die Ausführung ab:

  DRI0490 SESAMSQL UEBERSETZUNG BEI OFFENER TRANSAKTION NICHT MOEGLICH

2.4 Datenbank-Umgebung einstellen

Eine Datenbank-Umgebung für SESAM/SQL V2 besteht aus allen SQL2-Datenbanken, die mit einem SESAM/SQL2-Server (SESAM-DBH) angesprochen werden können. Dies wird in der Server-Konfigurationsdatei mit ADD-SQL-DATABASE-CATALOG-LIST festgelegt.

Um mit SESAM/SQL V2 arbeiten zu können, müssen folgende Angaben gemacht werden:
– voreingestellter Katalogname (vgl. Abschnitt „Datenbank- und Schemawechsel“ auf Seite 28)
– voreingestellter Schemaname (vgl. Seite 28)
– aktueller Berechtigungsschlüssel (SQL-User-Name).

Mit diesen drei Namen wird die aktuelle SQL-Umgebung für die DRIVE-Sitzung (TIAM-Session oder UTM-Vorgang) festgelegt.

Beim Datenzugriff durch ein DRIVE-Programm prüft der Server SESAM/SQL V2 die Merkmale des SQL-Users:

– ist der SQL-User eingetragen (siehe CREATE USER-Anweisung)

– ist für den SQL-User ein Systemzugang vorhanden (siehe CREATE SYSTEM_USER-Anweisung im SESAM-Handbuch [18])

– hat der SQL-User Zugriffsrecht auf die Daten (siehe GRANT-Anweisung).

Für die Übersetzung und für Programme mit statischen SQL-Anweisungen werden diese Größen mit OPTION CATALOG/SCHEMA/AUTHORIZATION versorgt (Seite 28). Für dynamische SQL-Anweisungen werden diese Größen mit PARAMETER DYNAMIC oder SET CATALOG /SCHEMA/ AUTHORIZATION versorgt (siehe Seite 36, Seite 137, Seite 139, Seite 141). SET-Werte gelten nur für das aktuelle DRIVE-Programm, PARAMETER-Werte gelten als zeitlich letzte Einstellung auch über das DRIVE-Programmende hinaus im Dialog-Modus und als Vorbelegung in der Parameter-Maske.

Fehlen bei der Übersetzung OPTION-Anweisungen oder -Klauseln, werden die Parametereinstellungen übernommen. Sind diese nicht vorhanden, bricht DRIVE/WINDOWS im Falle von AUTHORIZATION mit der Fehlermeldung DRI0525 ab und verwendet in den Fällen CATALOG und SCHEMA den Standardwert Leerzeichen.

Dieses Kapitel beschreibt das Zusammenspiel der DRIVE- und SQL-Anweisungen, mit denen Sie die Datenbank-Umgebung einstellen: PARAMETER, OPTION und SET mit den Operanden CATALOG/SCHEMA/AUTHORIZATION.

2.4.1 SET CATALOG/SCHEMA/SESSION AUTHORIZATION

Mit der Anweisung SET legen Sie die SQL-Umgebung fest, mit der Sie arbeiten wollen. Die SET-Anweisung wird erst bei der Ausführung wirksam. SET SCHEMA und SET CATALOG wirken bei DRIVE/WINDOWS wie bei SESAM nur auf dynamische Anweisungen. SET SESSION AUTHORIZATION wirkt bei DRIVE/WINDOWS nur auf dynamische Anweisungen (abweichend von SESAM).

Im einzelnen gilt folgendes:

– Eine SET CATALOG-Anweisung legt den voreingestellten Katalognamen für dynamische SQL-Anweisungen fest. Im Programm-Modus ist sie statisch und dynamisch erlaubt. Im Dialog-Modus ist sie nicht erlaubt.

– Eine SET SCHEMA-Anweisung legt den voreingestellten Schemanamen für dynamische SQL-Anweisungen fest. Im Programm-Modus ist sie statisch und dynamisch erlaubt. Im Dialog-Modus ist sie nicht erlaubt.

– Eine SET SESSION AUTHORIZATION-Anweisung legt den aktuellen Berechtigungsschlüssel (SQL-User-Name) für nachfolgende SQL-Anweisungen fest. Sie ist im Programm-Modus nur dynamisch erlaubt und kann daher nur auf dynamische SQL-Anweisungen wirken; in offener Transaktion wird sie von SESAM mit SQLSTATE 25SA4 abgewiesen. Im Dialog-Modus ist sie nicht erlaubt.

2.4.2 PARAMETER DYNAMIC CATALOG/SCHEMA/AUTHORIZATION

Mit der Anweisung PARAMETER DYNAMIC CATALOG/SCHEMA/AUTHORIZATION legen Sie die SQL-Umgebung für die DRIVE-Sitzung fest. Die angegebenen Werte werden von DRIVE/WINDOWS verwaltet und an das Datenbanksystem SESAM V2 weitergegeben. D.h. DRIVE/WINDOWS hält die DRIVE- und die SQL-Umgebung konsistent. Die Parameter CATALOG/SCHEMA/AUTHORIZATION können eingestellt werden:

– im Dialog-Modus (als Einzelanweisung oder maskenunterstützt im BS2000 und SINIX, menügeführt in der SPU)
– im Programm-Modus

– als Start-Up-Parameter beim Starten von UTM
– im UTM-Vorlauf-Programm über Benutzerkennsätze.

Diese Voreinstellung wird außer bei einem expliziten COMMIT WORK auch in folgenden Situationen festgeschrieben:
– nach ordnungsgemäßem oder fehlerhaftem Programmende (END PROC auf oberster Stufe oder BREAK)
– vor Folge-DO-Anweisungen
– nach ROLLBACK WORK im Dialog-Modus
– nach Beenden der ersten Tansaktion im Programm-Modus durch ROLLBACK WORK
– bei Vorgangsbeginn unter UTM, falls es Start-Up-Parameter für die UTM-Anwendung oder Benutzerkennsätze mit den entsprechenden PARAMETER DYNAMIC-Anweisungen gibt.

Die Wirkung der Anweisung PARAMETER DYNAMIC CATALOG/SCHEMA/ AUTHORIZATION ist für statische und dynamische Anweisungen unterschiedlich.

Bei der Übersetzung von Programmen werden die Werte für CATALOG/SCHEMA/ AUTHORIZATION in der folgenden Priorität gültig:

– sind im Programm mit der Anweisung OPTION Werte für CATALOG/SCHEMA/ AUTHORIZATION gesetzt, gelten diese, falls bei COMPILE keine entsprechenden OPTION-Werte angegeben wurden.

– ansonsten gelten die Werte aus den Compilereinstellungen (OPTION-Klausel der COMPILE-Anweisung)

– sind auch diese nicht vorhanden, gelten die Werte aus der Parametervoreinstellung

– sind auch diese nicht vorhanden, so gilt im Falle von CATALOG und SCHEMA das Leerzeichen als Standardwert, im Falle von AUTHORIZATION wird mit der Fehlermeldung DRI0525 abgebrochen.

  Die drei gültigen Werte werden in die Übersetzungsliste geschrieben und (von SESAM) im abgespeicherten Zwischencode festgeschrieben, so daß sie bei der Ausführung des Programms für alle statischen Anweisungen gelten, unabhängig von aktuellen Werten des Laufzeitsystems.

Für dynamische Anweisungen gilt:

– sind im Programm mit SET-Anweisungen Werte für CATALOG/SCHEMA/ AUTHORIZATION gesetzt, gelten diese unabhängig von anderen Einstellungen.

  Dabei schreibt ein COMMIT WORK im Programm die im Programm mit SET-Anweisungen gesetzten Werte für CATALOG/SCHEMA/AUTHORIZATION fest. Sie gelten bis zum Programmende, falls keine weiteren SET-Anweisungen folgen und die Parametervoreinstellung nicht geändert wird.

Ein ROLLBACK WORK im Programm setzt die im Programm mit SET-Anweisungen gesetzten Werte für CATALOG/SCHEMA/AUTHORIZATION zurück, falls sie nicht zuvor schon mit COMMIT WORK festgeschrieben wurden. D.h. es werden dann die Werte aus der Parametervoreinstellung oder der letzten festgeschriebenen SET-Anweisung gültig.

– enthält das Programm keine SET-Anweisungen zum Setzen der Werte für CATALOG/SCHEMA/AUTHORIZATION, gelten die Werte aus der aktuellen Parametervoreinstellung des Laufzeitsystems.

2.4.3 OPTION CATALOG/SCHEMA/AUTHORIZATION

Mit der Anweisung OPTION steuern Sie auch die Übersetzung eines DRIVE-Programms mit Zugriff auf eine SESAM-Datenbank. Diese Anweisung gilt für die Übersetzung. Außerdem werden die Werte im abgepeicherten Zwischencode festgeschrieben, so daß sie bei der Ausführung des Programms für alle statischen SQL-Anweisungen gelten, unabhängig von aktuellen Werten des Laufzeitsystems.

Bei der Übersetzung werden Optionenwerte nur für statische Anweisungen übernommen. Werden keine Optionen gesetzt, überprüft DRIVE/WINDOWS, ob Parameter-Werte gesetzt sind. Ist dies nicht der Fall, wird im Falle AUTHORIZATION mit der Fehlermeldung DRI0525 abgebrochen. In den Fällen CATALOG und SCHEMA verwendet DRIVE/WINDOWS das Leerzeichen als Standardwert.

Da DRIVE/WINDOWS **immer mit Datenbankkontakt übersetzt**, wird zu Beginn der DRIVE-Übersetzung von SESAM eine Datenbanktransaktion eröffnet („SESAM-Vorübersetzung“), die erst am Ende der DRIVE-Übersetzung von SESAM zurückgesetzt wird. Dies bedeutet insbesondere, daß pro Übersetzung immer nur ein Berechtigungsschlüssel verwendet werden kann. Die Verwendung mehrerer Berechtigungsschlüssel wäre nur bei einer SESAM-Vorübersetzung ohne Datenbankkontakt sinnvoll. Dies ist bei DRIVE/WINDOWS nicht erlaubt.

Die Übersetzung kann nur in TA-losem Zustand beginnen, da bei der ersten (deklarativen oder ausführbaren) statischen SQL-Anweisung bei SESAM eine Transaktion eröffnet wird.

2.4.4 Datenbank- und Schemawechsel

Mit dem Operanden SCHEMA bei PARAMETER DYNAMIC oder OPTION und mit der SET SCHEMA-Anweisung wird der Name eines Schemas voreingestellt, auf das sich in der Folge alle mit einfachen Namen angegebenen Tabellen (Basistabellen oder persistente Views) und Integritätsbedingungen beziehen.

Mit dem Operanden CATALOG bei PARAMETER DYNAMIC oder OPTION und mit der SET CATALOG-Anweisung wird der Name einer Datenbank (SESAM-Katalog) voreingestellt, auf die sich in der Folge alle mit einfachen Namen angegebenen Schemas, Tabellen und Integritätsbedingungen beziehen.

Sie können ein anderes als das voreingestellte Schema durch direkte Qualifizierung mit dem Schemanamen referenzieren und entsprechend eine andere als die voreingestellte Datenbank durch direkte Qualifizierung mit dem Katalognamen. Dieser Zugriff kann z.B. sinnvoll sein, wenn Sie in einer Transaktion zwei Datenbanken referenzieren wollen. In diesem Fall muß es allerdings möglich sein, auf beide Datenbanken mit demselben SQL-User zuzugreifen, da dieser nur außerhalb einer Transaktion geändert werden kann. Die unterschiedlichen Transaktionsprofile von statischen und dynamischen (New-Style-) Programmen werden im Abschnitt „DRIVE-Programmzugriff auf SESAM“ auf Seite 12 beschrieben.

2.4.5 Aktueller Berechtigungsschlüssel zur Übersetzung und Ausführung

Der aktuelle Berechtigungsschlüssel wird folgendermaßen bestimmt:

– Wird OPTION AUTHORIZATION angegeben, gilt der angegebene Berechtigungsschlüssel in der gesamten Übersetzung und bei der Ausführung jeder übersetzten statischen SQL-Anweisung.

– Wird OPTION AUTHORIZATION nicht angegeben, so gilt für die Übersetzung und bei der Ausführung jeder übersetzten statischen SQL-Anweisung der aktuelle Berechtigungsschlüssel aus der Parametereinstellung von DRIVE/WINDOWS zum Übersetzungszeitpunkt.

Dieser aktuelle Berechtigungsschlüssel kann weder durch SET SESSION AUTHORIZATION-Anweisungen noch durch PARAMETER DYNAMIC AUTHORIZATION-Anweisungen geändert werden.

Durch das implizite ROLLBACK WORK von SESAM am Ende der Übersetzung wird der aktuelle Berechtigungschlüssel zurückgesetzt, falls er über OPTION im Programm eingestellt wurde. Es gilt dann wieder die aktuelle Parametervoreinstellung.

Zum Ausführungszeitpunkt werden alle SET SESSION- und PARAMETER DYNAMIC AUTHORIZATION-Anweisungen ausgeführt und folglich jeweils der aktuelle Berechtigungsschlüssel aktualisiert. Diese Änderungen des aktuellen SQL-Users wirken aber nur auf dynamische Anweisungen. Für statische Anweisungen gilt der bei der Übersetzung wie beschrieben eingestellte Berechtigungsschlüssel.

2.4.6 Beispiele zur Datenbank-Umgebung

2.4.6.1 SESAM-Datenbank und DRIVE-Programme

Der SQL-User FLE hat Zugang zum Katalog TLDBSQL2 und ist Eigentümer des Schemas FLE. Das Schema FLE enthält die Tabelle FIRMA mit der Spalte SCHLUESSEL vom Typ CHAR(6).

Der SQL-User HUGO hat keinen Zugang zum Katalog TLDBSQL2, d.h. entweder existiert der User nicht oder er hat keinen Systemzugang für die BS2000-Benutzerkennung, in der DRIVE/WINDOWS läuft, oder für DRIVE/WINDOWS als UTM-Anwendung.

Der SQL-User DOM hat Zugang zum Katalog TLDBSQL2, aber kein Zugriffsrecht auf die Tabelle FIRMA, d.h. der Eigentümer FLE hat für DOM keine entsprechende GRANT-Anweisung abgesetzt.

Die DRIVE-Bibliothek „$YDRI6FLE.FLE.LIB“ (Linkname USEROML) enthält die folgenden DRIVE-Programme:

```
/* DRIVE-Programm ENVIRONMENT */
PROC ENVIRONMENT;
PAR DYN AUTHORIZATION=FLE;
PAR DYN CATALOG=TLDBSQL2;
PAR DYN SCHEMA=FLE;
END PROC;
```

```
/* DRIVE-PROGRAMM BEISPIEL */
PROC BEISPIEL;
DCL VAR &FIRMA LIKE TABLE FIRMA;
EXEC 'SET SESSION AUTHORIZATION "HUGO"';
INSERT INTO FIRMA (SCHLUESSEL) VALUES ('HUGO00');
EXEC 'INSERT INTO FIRMA (SCHLUESSEL) VALUES ("HUGO01")';
ROLLBACK WORK;
END PROC;
```

2.4.6.2 TIAM-Betrieb

Die folgenden Beispiele zeigen das Zusammenspiel der Anweisungen PARAMETER, OPTION und SET zum Einstellen der SQL-Umgebung für den TIAM-Betrieb.

do environment

compile beispiel option listing=lib authorization=hugo

```
    PROGRAMM      : BEISPIEL
    BIBLIOTHEK    : FLE.LIB

ZEILE  QUELLE NEST BEISPIEL              SEITE :     1

1      1   0  PROC BEISPIEL;
2      2   0  DCL VAR &FIRMA LIKE TABLE FIRMA;
***                  *
***          %  DRI0536  42SQG  -550 SYSTEMZUGANG FUER DEN
             BERECHTIGUNGSSCHLUESSEL HUGO IM CATALOG TLDBSQL2 NICHT ZUGREIFBAR
3      3   0  EXEC 'SET SESSION AUTHORIZATION "HUGO"';
4      4   0  INSERT INTO FIRMA (SCHLUESSEL) VALUES ('HUGO00');
***          *
***          %  DRI0536  42SQG  -550 SYSTEMZUGANG FUER DEN
             BERECHTIGUNGSSCHLUESSEL HUGO IM CATALOG TLDBSQL2 NICHT ZUGREIFBAR
5      5   0  EXEC 'INSERT INTO FIRMA (SCHLUESSEL) VALUES ("HUGO01")';
6      6   0  ROLLBACK WORK;
7      7   0  END PROC;

UEBERSETZUNGSOPTIONEN   BEISPIEL     SEITE :     2

OPTION DBSYSTEM     = SESAMSQL  DURCH VOREINSTELLUNG
OPTION LISTING      = LIBRARY   DURCH KOMMANDO
OPTION CODE         = OFF       DURCH VOREINSTELLUNG
OPTION SCHEMA       = FLE       DURCH VOREINSTELLUNG
OPTION CATALOG      = TLDBSQL2  DURCH VOREINSTELLUNG
OPTION AUTHORIZATION = HUGO     DURCH KOMMANDO

ANZAHL FEHLER :         2
```

compile beispiel option listing=lib authorization=dom

```
   PROGRAMM      : BEISPIEL
   BIBLIOTHEK    : FLE.LIB

ZEILE QUELLE NEST BEISPIEL              SEITE :      1

1      1   0  PROC BEISPIEL;
2      2   0  DCL VAR &FIRMA LIKE TABLE FIRMA;
***                  *
***          % DRI0536 42SQK -127 TABELLE TLDBSQL2.FLE.FIRMA FUER BENUTZER DOM
NICHT ZUGREIFBAR
3      3   0  EXEC 'SET SESSION AUTHORIZATION "HUGO"';
4      4   0  INSERT INTO FIRMA (SCHLUESSEL) VALUES ('HUGO00');
***          *
***          % DRI0536 42SQK -127 TABELLE TLDBSQL2.FLE.FIRMA FUER BENUTZER
             DOM NICHT ZUGREIFBAR
5      5   0  EXEC 'INSERT INTO FIRMA (SCHLUESSEL) VALUES ("HUGO01")';
6      6   0  ROLLBACK WORK;
7      7   0  END PROC;

UEBERSETZUNGSOPTIONEN   BEISPIEL              SEITE :      2

OPTION DBSYSTEM     = SESAMSQL  DURCH VOREINSTELLUNG
OPTION LISTING      = LIBRARY   DURCH KOMMANDO
OPTION CODE         = OFF       DURCH VOREINSTELLUNG
OPTION SCHEMA       = FLE       DURCH VOREINSTELLUNG
OPTION CATALOG      = TLDBSQL2  DURCH VOREINSTELLUNG
OPTION AUTHORIZATION = DOM      DURCH KOMMANDO

ANZAHL FEHLER :         2
```

compile beispiel option listing=lib code=on

```
   PROGRAMM      : BEISPIEL
   BIBLIOTHEK    : FLE.LIB

ZEILE  QUELLE NEST BEISPIEL                SEITE :      1

1      1   0  PROC BEISPIEL;
2      2   0  DCL VAR &FIRMA LIKE TABLE FIRMA;
3      3   0  EXEC 'SET SESSION AUTHORIZATION "HUGO"';
4      4   0  INSERT INTO FIRMA (SCHLUESSEL) VALUES ('HUGO00');
5      5   0  EXEC 'INSERT INTO FIRMA (SCHLUESSEL) VALUES ("HUGO01")';
6      6   0  ROLLBACK WORK;
7      7   0  END PROC;

UEBERSETZUNGSOPTIONEN    BEISPIEL                SEITE :      2

OPTION DBSYSTEM      = SESAMSQL  DURCH VOREINSTELLUNG
OPTION LISTING       = LIBRARY   DURCH KOMMANDO
OPTION CODE          = ON        DURCH KOMMANDO
OPTION SCHEMA        = FLE       DURCH VOREINSTELLUNG
OPTION CATALOG       = TLDBSQL2  DURCH VOREINSTELLUNG
OPTION AUTHORIZATION = FLE       DURCH VOREINSTELLUNG

ANZAHL FEHLER :          0
```

debug beispiel

```
********* BIBLIOTHEK  : FLE.LIB
   ZEILE PROGRAMM    : BEISPIEL
      1 PROC BEISPIEL;
% DRI0593  ABLAUFVERFOLGUNG BEENDET: ZEILE     1 IN PROZEDUR 'FLE.LIB(BEISPIEL)
'
% DRI0576  ANFANGS-HALTEPUNKT ERREICHT
*t all
********* BIBLIOTHEK  : FLE.LIB
   ZEILE PROGRAMM    : BEISPIEL
      3 EXEC 'SET SESSION AUTHORIZATION "HUGO"';
      4 INSERT INTO FIRMA (SCHLUESSEL) VALUES ('HUGO00');
      5 EXEC 'INSERT INTO FIRMA (SCHLUESSEL) VALUES ("HUGO01")';
% DRI0536  42SQG  -550 SYSTEMZUGANG FUER DEN BERECHTIGUNGSSCHLUESSEL HUGO IM
CA
TALOG TLDBSQL2 NICHT ZUGREIFBAR
% DRI0579  FEHLER BEI AUSFUEHRUNG DER ANWEISUNG 'EXEC'
% DRI0578  AKTUELLER HALTEPUNKT: ZEILE     5 IN PROZEDUR 'FLE.LIB(BEISPIEL)'
*break debug
```

2.4.6.3 UTM-Betrieb

Die Beispiele des TIAM-Betriebs verhalten sich im UTM-Betrieb genauso. Lediglich die Definition von Systemzugängen für SQL-User ist unterschiedlich (vgl. CREATE SYSTEM_USER-Anweisung im SESAM-Handbuch [18]).

Zusätzlich kann die SQL-Umgebung durch Parameter in der UTM-Startprozedur und durch Benutzerkennsätze mit Verweisen auf UTM-Vorlaufprozeduren eingestellt werden.

Das folgende Beispiel zeigt diese Möglichkeit zur Einstellung der SQL-Umgebung mit PARAMETER-Anweisungen im UTM-Betrieb.

Die UTM-Startprozedur enthält folgende DRIVE-Start-Up-Parameter:

```
.DRIVE PAR DYN LIB="$YDRI6FLE.FLE.LIB";

.DRIVE PAR DYN CATALOG=WRZLPRMPFT;

.DRIVE PAR DYN SCHEMA=HOKUS;

.DRIVE PAR DYN AUTHORIZATION=POKUS;
```

Die DRIVE-Bibliothek „$YDRI6FLE.FLE.LIB“ (Linkname USEROML) enthält ein S-Element mit dem Namen „DRISQL@@FELIX@@@“ oder „DRISQL@@@@@@@@@@“ und der einzigen Zeile

```
"$YDRI6FLE.FLE.LIB"(ENVIRONMENT)
```

als Inhalt (Benutzerkennsatz mit Verweis auf eine Vorlaufprozedur).

Die UTM-Vorlauf-Prozedur ENVIRONMENT wurde oben dargestellt.

Die folgenden Schritte werden beim Starten von DRIVE im UTM-Betrieb ausgeführt:

1. Beim Hochfahren von UTM werden die Parameterwerte WRZLPRMPFT, HOKUS und POKUS festgehalten, ohne daß diese an SESAM geschickt werden.
2. Nach Anmelden des UTM-Users FELIX mit KDCSIGN und Eingabe des ersten Transaktionscodes DRISQL werden die Anweisungen SET CATALOG 'WRZLPRMPFT', SET SCHEMA 'HOKUS' und SET SESSION AUTHORIZATION 'POKUS' an SESAM geschickt.
3. Anschließend wird der Benutzerkennsatz DRISQL@@FELIX@@@ gefunden und daraufhin die Vorlaufprozedur ENVIRONMENT ausgeführt.

Der Übersichtlichkeit halber wird empfohlen, UTM-Vorlaufprozeduren ohne SQL-Anweisungen zu programmieren und gewünschte SQL-Zugriffe in ein Folge-DO- oder CALL-Programm zu verlagern, das unmittelbar vor END PROC aufgerufen wird.

Bei END PROC wird die aktuelle Parametervoreinstellung festgeschrieben. Danach befindet sich der UTM-User FELIX im DRIVE-Dialog-Modus, und es ist der SQL-User FLE voreingestellt.

Für den Fall, daß die Vorlaufprozedur als letzte Anweisung vor END PROC eine Folge-DO-Anweisung enthält, wird vor dem Start des neuen Dialog-Programms ebenfalls die Parametervoreinstellung festgeschrieben. Ist die letzte Anweisung vor END PROC jedoch eine CALL-Anweisung, so wird sie nicht vor dem Start des neuen Programms, sondern erst beim nächsten COMMIT WORK festgeschrieben.

2.5 DRIVE-Anweisungen für SESAM V2

Mit den DRIVE-Anweisungen PARAMETER DYNAMIC und OPTION für SESAM können Sie die SQL-Umgebung einstellen.

Beispiele unter SHOW zeigen, wie Sie mit SELECT-Abfragen auf SESAM-Systemviews Informationen über permanente Datenbankobjekte erhalten.

2.5.1 PARAMETER DYNAMIC - Dynamische Parameter festlegen

PARAMETER DYNAMIC legt die SQL-Umgebung für die DRIVE-Sitzung (TIAM-Session oder UTM-Vorgang) fest. Die Anweisung gilt im Programm- und Dialog-Modus.

Eine vollständige Operandenbeschreibung der Anweisung PARAMETER DYNAMIC finden Sie im DRIVE-Lexikon [3].

Für statische Programme gilt:
Werden für die Operanden AUTHORIZATION, CATALOG und SCHEMA weder in einer OPTION-Anweisung im Programm noch in der OPTION-Klausel der COMPILE-Anweisung Werte gesetzt, so gelten die Werte, die mit der Anweisung PARAMETER DYNAMIC gesetzt wurden oder vorbelegt sind. Fehlen Werte für CATALOG und SCHEMA, wird jeweils ein Leerzeichen als Wert genommen. Ist AUTHORIZATION nicht vorhanden, wird die Übersetzung mit der DRIVE-Fehlermeldung DRI0525 abgebrochen.

Für dynamische Programme gilt:
Werden CATALOG, SCHEMA und AUTHORIZATION nicht über SET-Anweisungen gesetzt, gelten die Werte aus der Parametereinstellung.

Die Anweisung PARAMETER DYNAMIC eröffnet keine Transaktion.

PARAMETER **DYN**AMIC [AUTHORIZATION=berechtigung |
CATALOG=sesdbname |
SCHEMA=schemaname]

AUTHORIZATION

Dieser Operand kann nur im transaktionslosen Zustand eingegeben werden, andernfalls reagiert DRIVE/WINDOWS mit der Fehlermeldung DRI0084.

Legt den aktuellen Berechtigungsschlüssel *berechtigung* (max. 18 Zeichen) für den New-Style-Zugriff auf SESAM-Datenbanken fest.

Unter MS-Windows und SINIX müssen Sie auch den dynamischen Parameter DBSYSTEM=SESAMSQL angeben. Im BS2000 ist automatisch die geladene Datenbank-Fassung vorbelegt.

Dieser Operand wird auch beim Übersetzen ausgewertet, falls keine OPTION AUTHORIZATION angegeben ist.

Vorbelegung: Leerstring in DRIVE/WINDOWS, D0USER in SESAM

CATALOG

Legt eine SESAM-Datenbank fest.

Unter MS-Windows und SINIX müssen Sie auch den Parameter DYNAMIC DBSYSTEM=SESAMSQL angeben. Im BS2000 ist automatisch die geladene Datenbank-Fassung vorbelegt.

CATALOG=*sesdbname* gilt für SQL-Anweisungen, die im Dialog-Modus eingegeben werden. In Programmen gilt dieser Operand nur für dynamische SQL-Anweisungen.

Dieser Operand wird auch beim Übersetzen ausgewertet, falls keine OPTION CATALOG angegeben ist.

Vorbelegung: Leerstring in DRIVE/WINDOWS, Leerzeichen in SESAM

SCHEMA

Name eines SQL-Schemas, auf das zugegriffen wird, wenn in SQL-Anweisungen kein Name angegeben wurde.

SCHEMA=*schemaname* gilt für SQL-Anweisungen, die im Dialog-Modus eingegeben werden. In Programmen gilt dieser Operand nur für dynamische SQL-Anweisungen. Dieser Operand wird auch beim Übersetzen ausgewertet, falls keine OPTION SCHEMA angegeben ist.

Vorbelegung: Leerstring in DRIVE/WINDOWS, Leerzeichen in SESAM

2.5.2 OPTION - Übersetzung eines Programms steuern

OPTION steuert den Übersetzungslauf eines DRIVE-Programms. Zur Übersetzung von DRIVE-Programmen in Zwischencode oder Objektcode werden Compiler-Optionen eingestellt. Eine vollständige Operandenbeschreibung der Anweisung OPTION finden Sie im DRIVE-Lexikon [3].

Im BS2000 und SINIX kann OPTION in Sourcen (vor der Anweisung PROCEDURE und DECLARE TYPE) oder als Operand mit der Anweisung COMPILE angegeben werden. In MS-WINDOWS **muß** OPTION in Sourcen vor PROCEDURE und DECLARE TYPE angegeben werden.

Parameter-Voreinstellungen werden durch OPTION-Angaben in der Source überschrieben. OPTION-Angaben in der Source werden durch OPTION-Angaben bei COMPILE oder Übersetzungsoptionen, die Sie menügeführt eingeben, überschrieben.

Die zur Übersetzung gültigen OPTION-Werte werden im Zwischencode abgespeichert, so daß sie auch zur Ausführung gelten. Für die Übersetzung sind die Operanden AUTHORIZATION, CATALOG und SCHEMA obligatorisch. Sind keine Werte festgelegt, übernimmt DRIVE/WINDOWS die mit der Anweisung PARAMETER DYNAMIC eingestellten SQL-Umgebungswerte.

```
OPTION { AUTHORIZATION=berechtigung |
       CATALOG=sesdbname |
       SCHEMA=schemaname }
```

AUTHORIZATION

Legt den Berechtigungsschlüssel *berechtigung* für eine SESAM-Datenbank fest. Der hier festgelegte Berechtigungsschlüssel kann nicht mit einer SET SESSION AUTHORIZATION-Anweisung geändert werden.

D.h. wird das Programm mit diesem Operanden übersetzt, so wirkt eine Änderung des aktuellen Berechtigungsschlüssels durch SET oder PARAMETER zur Ausführung nur auf dynamische SQL-Anweisungen.

Vorbelegung: Die aktuelle Einstellung aus PARAMETER DYNAMIC AUTHORIZATION. Ist diese nicht vorhanden, d.h. mit Leerstring vorbelegt, weil in der DRIVE-Sitzung noch keine PARAMETER DYNAMIC AUTHORIZATION-Anweisung abgesetzt wurde, bricht DRIVE/WINDOWS die Übersetzung mit der Fehlermeldung DRI0525 ab.

CATALOG

Legt die Voreinstellung für eine SESAM-Datenbank fest.

Wird der Operand nicht angegeben, so wird der Katalogname der zuletzt ausgeführten Parameter-Einstellung verwendet. Ist diese nicht vorhanden, wird die Übersetzung abgebrochen.

Vorbelegung: Die aktuelle Einstellung aus PARAMETER DYNAMIC CATALOG oder, falls diese nicht vorhanden ist, Leerzeichen. Ist CATALOG mit einem Leerzeichen vorbelegt, so müssen in statischen SQL-Anweisungen die Namen von Schemas, Tabellen und Integritätsbedingungen mit einem Katalognamen qualifiziert werden.

SCHEMA

Wenn in einem Programm der Schemaname in SQL-Anweisungen nicht angegeben ist, wird zum Ablaufzeitpunkt der hier angegebene *schemaname* verwendet. Wird der Operand nicht angegeben, wird der Schemaname der zuletzt ausgeführten Parameter-Einstellung verwendet. Ist diese nicht vorhanden, wird die Übersetzung abgebrochen.

Vorbelegung: Die aktuelle Einstellung aus PARAMETER DYNAMIC SCHEMA oder, falls diese nicht vorhanden ist, Leerzeichen. Ist SCHEMA mit einem Leerzeichen vorbelegt, so müssen in statischen SQL-Anweisungen die Namen von Tabellen und Integritätsbedingungen mit einem Schemanamen qualifiziert werden.

2.5.3 SESAM V2-Einstellungen in fremder Umgebung

Die folgende Tabelle zeigt, wie die Parameter- und Optionseinstellungen in SESAM V1-, UDS- und INFORMIX-Umgebung wirken:

	PAR DYN CATALOG	**PAR DYN SCHEMA**	**PAR DYN AUTHORIZATION**	**OPTION CATALOG**	**OPTION SCHEMA**	**OPTION AUTHORIZATION**
SESAM V1	abgewiesen mit DRI0229	nur im Dialog-Modus erlaubt; schemaname muß der Name einer Basisrelation sein; im Dialog-Modus kann nur diese Basisrelation ohne schemaname referenziert werden	abgewiesen mit DRI0229	ignoriert	schemaname muß der Name einer Basisrelation sein; im Programm mit OPTION SCHEMA kann nur diese Basisrelation ohne schemaname referenziert werden	ignoriert
UDS	abgewiesen mit DRI0229	nur im Dialog-Modus erlaubt	abgewiesen mit DRI0229	ignoriert	schemaname muß der Name einer relationalen Sicht auf ein CODASYL-Subschema sein	ignoriert
INFORMIX	abgewiesen mit DRI0229	abgewiesen mit DRI0229	abgewiesen mit DRI0229	ignoriert	ignoriert	ignoriert

2.5.4 SHOW - Informationen über Metadaten ausgeben

Diese Anweisung wird für SESAM/SQL V2 nicht unterstützt. Sie können sich Informationen über den SESAM-Utility-Monitor einholen oder mit SELECT-Abfragen auf die SESAM-Systemviews (siehe SESAM/SQL-Server, SQL-Sprachbeschreibung Teil1 [18]). Diese SELECT-Abfragen können Sie z.B. als DRIVE-COPY-Element zum Ausgeben von Metadaten über permanente Datenbankobjekte ablegen. Auskünfte über temporäre Objekte, also Cursor und temporäre Views, können Sie sich nicht ausgeben lassen.

Beispiel1

Zum Abfragen von Schemainformationen sind folgende Schritte notwendig:
- stellen Sie die Datenbank mit OPTION ein,
- deklarieren Sie einen Cursor auf die Attribute des Systemviews, den Sie abfragen wollen,
- lesen Sie den Cursor in einer Schleife in eine Hilfsvariable ein
- und geben Sie diese mit DISPLAY aus.

```
OPTION CATALOG=TLDBSQL2 SCHEMA=TLDB AUTHORIZATION=TLAB;

PROC "schema-abfrage";

DCL SCHEMATA_C FOR S CATALOG_NAME, SCHEMA_NAME, SCHEMA_OWNER
    FROM INFORMATION_SCHEMA.SCHEMATA
    WHERE SCHEMA_NAME LIKE 'TL%';

DCL VAR &SCHEMATA_VAR LIKE CURSOR SCHEMATA_C;

CYCLE SCHEMATA_C INTO &SCHEMATA_VAR.*;

   DISPLAY FORM LINE NAMES VALUES &SCHEMATA_VAR;

END CYCLE;

COMMIT WORK;

END PROC;
```

Beispiel2

Zum Abfragen des Aufbaus einer Basistabelle deklarieren Sie folgenden Cursor:

```
OPTION CATALOG=TLDBSQL2 SCHEMA=TLDB AUTHORIZATION=TLAB;

PROC "tabellen-abfrage";

DCL TAB_COL_C FOR S TABLE_NAME, COLUMN_NAME, COLUMN_DEFAULT,
DATA_TYPE, CHARACTER_MAXIMUM_LENGTH, NUMERIC_PRECISION,
NUMERIC_SCALE,
   DATETIME_PRECISION
   FROM INFORMATION_SCHEMA.BASE_TABLE_COLUMNS
   WHERE TABLE_NAME LIKE 'TLDB%';
```

Beispiel3

Zum Abfragen von permanenten Views auf eine Basistabelle deklarieren Sie folgenden Cursor:

```
OPTION CATALOG=TLDBSQL2 SCHEMA=TLDB AUTHORIZATION=TLAB;

PROC "view-abfrage";

DCL VIEWS_C CURSOR FOR S * FROM INFORMATION_SCHEMA."TABLES"
   WHERE TABLE_TYPE = 'VIEW' AND TABLE_NAME LIKE 'TLDB%';
```

2.6 Beispiele und Beispiel-Datenbank

Dieser Abschnitt enthält die SESAM V1-Datenbanken aus der Programmiersprache für DRIVE/WINDOWS V1.1 und DRIVE V6 vor und nach der Migration in eine SESAM V2-Datenbank sowie die DRIVE-Programme, die darauf zugreifen.

2.6.1 Beispiel-Tabellen vor und nach der Migration

Dieser Abschnitt beschreibt die Struktur und den Inhalt der Beispieldaten. Diese sind als SESAM/SQL V1 Datenbanken vorhanden (siehe DRIVE-Programmiersprache V1.1). Die Datensicherung (DB-SIB) MITARBEITER wurde in Abschnitt „Datenbankaufbau und Migration von SESAM V1-Tabellen“ auf Seite 10 nach SESAM V2 migriert.

Für die Beispieldaten werden drei Basistabellen verwendet. Diese Basistabellen enthalten die Mitarbeiter, die Abteilungen und die Projekte eines Betriebes.

Für jede Basistabelle werden die in ihr enthaltenen Daten aufgelistet.

Struktur der Basistabellen

Für die Beispiele werden folgende drei Basistabellen verwendet:

– „ABTEILUNG“

– „MITARBEITER“

– „PROJEKT“

ABTEILUNG

Die Basistabelle „ABTEILUNG“ enthält die Informationen zu den Abteilungen des Betriebes. Sie wird mit einem DRIVE-Programm erzeugt und geladen (siehe Seite 50). Beschreibung der einzelnen Satzelemente der Basistabelle „ABTEILUNG“:

Satzelement	Datentyp	Beschreibung
ABTEILUNG_NR	CHARACTER (4)	Primärschlüssel mit Namen KEY_ABT_NR
BEZEICHNUNG	CHARACTER (10)	Bezeichnung der Abteilung
STANDORT	CHARACTER (20)	Standort der Abteilung
LEITER	CHARACTER (8)	Personalnummer des Abteilungsleiters

MITARBEITER

Die Basistabelle „MITARBEITER“ enthält die Informationen über die Mitarbeiter des Betriebes. Sie wird mit dem Utility-Monitor migriert (siehe Abschnitt „Datenbankaufbau und Migration von SESAM V1-Tabellen“ auf Seite 10).
Beschreibung der einzelnen Satzelemente der Basistabelle „MITARBEITER“:

Satzelement	**Datentyp**	**Beschreibung**
SESAM V1: COMP_PERS_NR	 CHARACTER (8)	SESAM V1: Personalnummer des Mitarbeiters, Compound-key bestehend aus ABT_MIT_NR und LFD_NR, einem automatischen Zählfeld SESAM V2: Dieser Compund-Key mit Zählfeld ist nicht direkt übertragbar auf eine SESAM V2-Datenbank, sondern wird dort als Constraint des Typs Primary Key über zwei Spalten definiert
ABT_MIT_NR	CHARACTER (4)	Abteilungsnummer des Mitarbeiters
LFD_NR	INTEGER	SESAM V1: laufende Nummer des Mitarbeiters (automat. Zählfeld) SESAM V2: CONSTRAINT vom Typ PRIMARY KEY
NACHNAME	CHARACTER (20)	Nachname des Mitarbeiters
VORNAME	CHARACTER (20)	Vorname des Mitarbeiters
LAND	CHARACTER (3)	Land
STRASSE	CHARACTER (26)	Strasse
PLZ	CHARACTER (10)	Postleitzahl
ORT	CHARACTER (20)	Wohnort
GEHALT	NUMERIC (7,2)	Gehalt des Mitarbeiters
SPRACHEN SESAM V2: SPRACHEN (4)	CHARACTER (3)	Fremdsprachen des Mitarbeiters (multiples Feld mit 4 Ausprägungen)
ABT_LEITER	CHARACTER (8)	Personalnummer des Abteilungsleiters
PROJ_MIT	CHARACTER (6)	enthält den Primärschlüsseldes Projekts,an dem der Mitarbeit gerade arbeitet

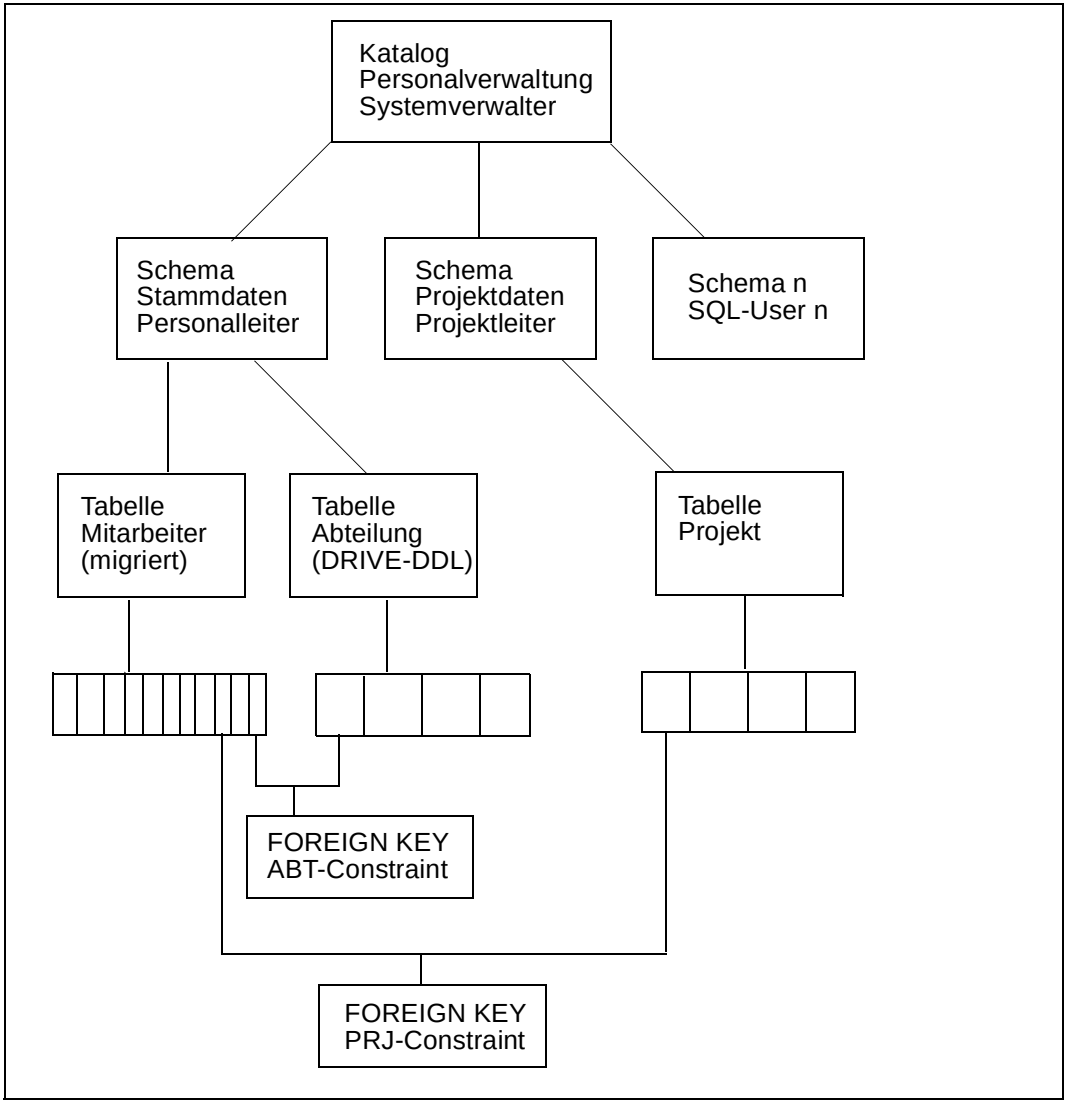
PROJEKT

Die Basistabelle „PROJEKT“ enthält die laufenden Projekte des Betriebes mit der Personalnummer des Projektleiters und dem zur Verfügung stehenden Budget.
Beschreibung der einzelnen Satzelemente der Basistabelle „PROJEKT“:

Satzelement	Datentyp	Beschreibung
PROJEKT_NR	CHARACTER (6)	Primärschlüssel mit Namen KEY_PROJEKT_NR
BEZEICHNUNG	CHARACTER (10)	Bezeichnung des Projekts
BUDGET	NUMERIC (12,2)	zur Verfügung stehendes Budget
PROJ_LEITER	CHARACTER (8)	Personalnummer des Projektleiters

Beziehungen zwischen den Basistabellen

SESAM V2-Beispieldatenbank im Überblick



Daten der Basistabellen

ABTEILUNG

ABTEILUNG_NR	BEZEICHNUNG	STANDORT	LEITER
0106	Zentrale	Muenchen	01060002
0107	Marketing	Kiel	01070004
0108	Transport	Hamburg	01080008
0109	Technik	Hannover	01090004
0110	Personal	Muenchen	01100002
0111	Ausland	Muenchen	01110004
0112	Forschung	Muenchen	01120003

MITARBEITER

ABT_MIT_NR	LFD_NR	NACH-NAME	VORNAME	LAND	STRASSE	PLZ	ORT	GEHALT	SPRACHEN	ABT_LEITER	PROJ_MIT
0106	1	Sennert	Gustl	FRG	Petuelring 26	8000	Muenchen	0580000	ENG	01060002	000106
0106	2	Bergmann	Susanne	FRG	Alter Platz 3	8000	Muenchen	0690000	ENG	01060002	
0106	3	Berghoff	Ruth	FRG	Isartor 7	8000	Muenchen	0300000		01060002	
0106	4	Bergner	Erich	FRG	Sonnemannstr. 512A	8000	Muenchen	0450000		01060002	000106
0107	1	Olsen	Mattes	FRG	Kiekendamm 13b	2300	Kiel	0400000	ENG	01070004	000108
0107	2	Mattsen	Gunter	FRG	Spoekemannswatt	2300	Kiel	0480000	ENG	01070004	000108
0107	3	Nonnen	Paul	FRG	Jollischweg 2b	2300	Kiel	0450000		01070004	
0107	4	Dormagen	Siegfried	FRG	Gundelstr.91	2322	Hemlstorf	0550000	FRASPA	01070004	000108
0108	1	Winterberg	Hannelore	FRG	Mitterstr. 8	2000	Hamburg	0350000	ENG	01080008	
0108	2	Mitscherlich	Hermann	FRG	Hansestadtallee 88	2000	Hamburg	0270000	ENG	01080008	
0108	3	Grenzer	Wilhelm	FRG	Am Kiefernpfad 4	2081	Heist	0230000	FRA	01080008	
0108	4	Paarungen	Morten	FRG	Ollenwatt 6	2050	Hamburg 80	0320000	ENG	01080008	
0108	5	Andermatt	Sylvia	FRG	Hohe Strasse 5	2000	Hamburg	0510000		01080008	000107
0108	6	Schmidt	Bettina	FRG	Hansaring 388	2000	Hamburg	0620000	ENG	01080008	000107
0108	7	Pollinger	Nora	FRG	Goethestr.32	2050	Hamburg 80	0400000	ENG	01080008	
0108	8	Mauer	Rita	FRG	Wiesenstr. 381	2000	Hamburg	0650000	ENG	01080008	
0109	1	Mitscher	Fred	FRG	Wiesenstr. 96	3000	Hannover	0490000	FRA	01090004	000109
0109	2	Jansen	Jutta	FRG	Giesestr. 17	3510	Hannover-Muenden	0470000		01090004	
0109	3	Zimmermann	Peter Johannes	FRG	Berger Markt 119c	3000	Hannover	0580000	ENGSPAFRA	01090004	
0109	4	Sennelaub	Andrea	FRG	Simonenpfad 57	3000	Hannover	0650000		01090004	000109
0109	5	Maier	Bernd	FRG	Kreuzgasse 8	3000	Hannover	0350000	ENG	01090004	
0109	6	Maler	Guenther	FRG	Rohrdamm 2	3000	Hannover	0290000	ENGFRA	01090004	
0110	1	Lorenz	Erna	FRG	Kiesgraben 61	8000	Muenchen	0330000	ENG	01100002	000111
0110	2	Ammerl	Sepp	FRG	Luxemburger Str. 427	8000	Muenchen	0570000	SPA	01100002	000111
0110	3	Dollinger	Johanna	FRG	Pferdesaalstr. 54	8000	Muenchen	0280000	FRA	01100002	000111
0111	1	Manson	Dr. Jack	FRG	Marienburg 1	8000	Muenchen	1200000	GER	01110004	000110
0111	2	van Claeren	Sandra	CH	Berner Weg 59	CH-3000	Bern 33	0590000	ENG	01110004	
0111	3	Miller	James	USA	5009 Beach Boulevard	CA 93440	Los Alamos	0960000	FRA	01110004	000110
0111	4	von Haalen	Walter	USA	30 Third Avenue	NY 11217	New York	1000000	GERSPAFRA	01110004	000110
0112	1	Plenzer	Wolfgang	FRG	Sesemieweg 104	8000	Muenchen	0360000	ENG	01120003	
0112	2	Heimlott	Hansi	FRG	Kanonenstr. 15	8000	Muenchen	0350000		01120003	
0112	3	Sindlinger	Max	FRG	Hollermesse 35	8000	Muenchen	0690000	ENGSPA	01120003	

PROJEKT

PROJEKT_NR	BEZEICHNUNG	BUDGET	PROJ_LEITER
000106	Zeus	000035000000	01060001
000107	Hera	000015000000	01080006
000108	Poseidon	000061000000	01070004
000109	Athene	000050000000	01090004
000110	Aphrodite	000008000000	01110004
000111	Prometheus	000002000000	01100002

2.6.2 Anweisungsdateien für die Migration

1. Katalog anlegen

   Mit der Utility-Anweisung CREATE CATALOG legen Sie einen Katalog an (im Beispiel „PERSONALVERWALTUNG“).

   Das Ergebnis ist eine BS2000-Datei PERSONALVERWALTUNG. CATALOG in der SESAM-Kennung. Der Universal User „SYSTEMVERWALTER“ ist Eigentümer der neuen SQL2-Datenbank.

   Beispiel

   Anweisungsdatei „UTI.PERSONALVERWALTUN.CATALOG“ , die den Katalog „PERSONALVERWALTUNG“ anlegt (vgl. Seite 10):

```
*
SQL CREATE CATALOG "PERSONALVERWALTUNG"  -
    CODE_TABLE '"EBCDIC_DF_03"'  -
    CATALOG_SPACE -
      PRIMARY 200 -
      SECONDARY 36 -
      PCTFREE 40 -
      SHARE -
      NO DESTROY -
      NO LOG -
    USER "SYSTEMVERWALTER"
*
SQL COMMIT WORK
END
```

2. SQL-User definieren

   Mit der Anweisung CREATE USER ... definieren Sie privilegierte SQL-User (im Beispiel „PERSONALLEITER“ und „PROJEKTLEITER“) , da nur ein SQL-User auf eine Datenbank zugreifen kann.

   Beispiel

   Anweisungsdatei „UTI.PERSONALVERWALTUNG.USER“, die die SQL-User definiert (vgl. Seite 10):

```
*
SQL SET CATALOG '"PERSONALVERWALTUNG'"
SQL SET SESSION AUTHORIZATION '"SYSTEMVERWALTER'"
*
SQL CREATE USER "PERSONALLEITER"  AT CATALOG "PERSONALVERWALTUNG"
SQL CREATE USER "PROJEKTLEITER"   AT CATALOG "PERSONALVERWALTUNG"
SQL CREATE USER "OLDSTYLE-ADMIN"  AT CATALOG "PERSONALVERWALTUNG"
SQL COMMIT WORK
END
```

3. Programmzugang für die SQL-User generieren

   Über die SYSTEM_USER kommuniziert ein SQL-User mit SESAM V2. Dazu muß die DRIVE-Anwendung in der gleichen Kennung geladen werden in der die Systemzugänge erzeugt wurden (Beispiel TIAM).

   Beispiel

   Anweisungsdatei „UTI.PERSONALVERWALTUNG.SYSTEMUSER“, die die Systemzugänge definiert (vgl. Seite 10):

```
*
SQL SET CATALOG "'PERSONALVERWALTUNG"'
SQL SET SESSION AUTHORIZATION "'SYSTEMVERWALTER"'
*
SQL CREATE SYSTEM_USER (*,,'STMQM235') -
    FOR "SYSTEMVERWALTER" -
    AT CATALOG "PERSONALVERWALTUNG"
*
SQL CREATE SYSTEM_USER (*,,'STMQM235') -
    FOR "PERSONALLEITER" -
    AT CATALOG "PERSONALVERWALTUNG"
*
SQL CREATE SYSTEM_USER (*,,'STMQM235') -
    FOR "PROJEKTLEITER"  -
    AT CATALOG "PERSONALVERWALTUNG"
```

```
*
SQL CREATE SYSTEM_USER (*,,'STMQM235') -
    FOR "OLDSTYLE-ADMIN"  -
    AT CATALOG "PERSONALVERWALTUNG"
*
SQL COMMIT WORK
END
```

4. Schemas generieren

   Mit der Anweisung CREATE SCHEMA generieren Sie Schemas (im Beispiel „STAMM-DATEN“ und „PROJEKTDATEN“), die den SQL-Usern („PERSONALLEITER“ und „PROJEKTLEITER“) als Eigentümer zugewiesen werden (CREATE SCHEMA ... AUTHORIZATION ...), so daß diese Eigentümer die Schemas verwalten.

   Beispiel

   Anweisungsdatei „UTI.PERSONALVERWALTUNG.SCHEMA“, die die Schemas anlegt (vgl. Seite 10):

```
*
SQL SET CATALOG "'PERSONALVERWALTUNG"'
SQL SET SESSION AUTHORIZATION "'SYSTEMVERWALTER"'
*
SQL CREATE SCHEMA "STAMMDATEN"  -
    AUTHORIZATION "PERSONALLEITER"
*
SQL CREATE SCHEMA "PROJEKTDATEN"  -
    AUTHORIZATION "PROJEKTLEITER"
*
SQL CREATE SCHEMA "OLDSTYLE-DATEN"  -
    AUTHORIZATION "OLDSTYLE-ADMIN"
*
SQL COMMIT WORK
END
```

2.6.3 DRIVE-DDL-Programme für Tabelle ABTEILUNG

Das folgende DRIVE-Programm „DRI.TABLE.ABTEILUNG“ erzeugt die Basistabelle „ABTEILUNG“ als eine SESAM V2-Tabelle im Schema „STAMMDATEN“ (Katalog „PERSONALVERWALTUNG“ (vgl. Seite 11).

```
OPTION AUTHORIZATION="PERSONALLEITER"
      CATALOG     ="PERSONALVERWALTUNG"
      SCHEMA      ="STAMMDATEN";
/* */
 PROC "DRI.TABLE.ABTEILUNG";
/* */
/* ----------- table, column, unique (vom Typ Primary Key) -----------*/
/* */
   CREATE TABLE "ABTEILUNG"
   (
     "ABTEILUNG_NR" CHAR(4) ,
     "BEZEICHNUNG"  CHAR(10),
     "STANDORT"     CHAR(20),
     "LEITER"       CHAR(8) ,
     CONSTRAINT "KEY_ABT_NR" PRIMARY KEY
      ( "ABTEILUNG_NR")
   ) ;
COMMIT WORK ;
END PROC;
```

Das folgende DRIVE-Programm lädt die Tabelle „ABTEILUNG“ mit Datensätzen.

```
OPTION AUTHORIZATION="PERSONALLEITER" CATALOG="PERSONALVERWALTUNG"
      SCHEMA="STAMMDATEN" ;
/*      */
PROC "DRI.LOAD.ABTEILUNG";
DCL VAR &V1 LIKE TABLE ABTEILUNG;
/*          */
SET &V1 = <
         '0106'               ,
         'ZENTRALE  '         ,
         'MUENCHEN          ' ,
         '01060002'
        > ;
/*          */
INSERT INTO ABTEILUNG VALUES (&V1.*) ;
DISPLAY FORM LINE RETURN &V1 ;
/*-------------------------------------------------------------*/
```

```
SET &V1 = <
        '0107'              ,
        'MARKETING '        ,
        'KIEL           ' ,
        '01070004'
       > ;
/*        */
INSERT INTO ABTEILUNG VALUES (&V1.*) ;
DISPLAY FORM LINE RETURN &V1 ;
/*-------------------------------------------------------*/
SET &V1 = <
        '0108'              ,
        'TRANSPORT '        ,
        'HAMBURG        ' ,
        '01080008'
       > ;
/*        */
INSERT INTO ABTEILUNG VALUES (&V1.*) ;
DISPLAY FORM LINE RETURN &V1 ;
/*-------------------------------------------------------*/
SET &V1 = <
        '0109'              ,
        'TECHNIK   '        ,
        'HANNOVER       ' ,
        '01090004'
       > ;
/*        */
INSERT INTO ABTEILUNG VALUES (&V1.*) ;
DISPLAY FORM LINE RETURN &V1 ;
/*-------------------------------------------------------*/
SET &V1 = <
        '0110'              ,
        'PERSONAL  '        ,
        'MUENCHEN       ' ,
        '01100002'
       > ;
/*        */
INSERT INTO ABTEILUNG VALUES (&V1.*) ;
DISPLAY FORM LINE RETURN &V1 ;
/*-------------------------------------------------------*/
SET &V1 = <
        '0111'              ,
        'AUSLAND   '        ,
        'MUENCHEN       ' ,
        '01110004'
       > ;
/*        */
```

```
INSERT INTO ABTEILUNG VALUES (&V1.*) ;
DISPLAY FORM LINE RETURN &V1 ;
/*-----------------------------------------------------------*/
SET &V1 = <
        '0112'              ,
        'FORSCHUNG '        ,
        'MUENCHEN          ' ,
        '01120003'
       > ;
/*         */
INSERT INTO ABTEILUNG VALUES (&V1.*) ;
DISPLAY FORM LINE RETURN &V1 ;
/*-----------------------------------------------------------*/
COMMIT WORK ;
END PROC;
```

2.6.4 DRIVE-DDL-Programm für Zugriffsrechte und Fremdschlüssel

Das folgende DRIVE-Programm „LOAD_FOREIGNKEY“ vergibt alle erforderlichen Rechte und generiert Fremdschlüssel (vgl. Seite 11).

```
PROC "LOAD_FOREIGNKEY" ;
/* */
/* VERGABE DER PRIVILEGIEN SELECT, DELETE, INSERT, UPDATE UND      */
/* REFERENCES AN ALLE SQL-USER DER DATENBANK                       */
/* */
EXEC 'SET SESSION AUTHORIZATION "PERSONALLEITER"';
EXEC 'SET CATALOG "PERSONALVERWALTUNG"';
EXEC 'SET SCHEMA "STAMMDATEN"';
/* */
EXEC 'GRANT ALL PRIVILEGES ON TABLE MITARBEITER TO PUBLIC ' ;
EXEC 'GRANT ALL PRIVILEGES ON TABLE ABTEILUNG TO PUBLIC ' ;
EXEC 'COMMIT WORK' ;
/* */
EXEC 'SET SESSION AUTHORIZATION "PROJEKTLEITER"';
EXEC 'SET CATALOG "PERSONALVERWALTUNG"';
EXEC 'SET SCHEMA "PROJEKTDATEN"';
/* */
EXEC 'GRANT ALL PRIVILEGES ON TABLE PROJEKT TO PUBLIC ' ;
EXEC 'COMMIT WORK' ;
/* */
EXEC 'COMMIT WORK' ;
/* */
END PROC;
```

```
/* FREMDSCHLUESSEL DEFINIEREN ZUM KEY PROJEKT.PROJEKT_NR UND ZUM    */
/* KEY ABTEILUNG.ABTEILUNG_NR                                       */
/* */
EXEC 'SET SESSION AUTHORIZATION "PERSONALLEITER"';
EXEC 'SET CATALOG "PERSONALVERWALTUNG"';
EXEC 'SET SCHEMA "STAMMDATEN"';
/* */
EXEC 'ALTER TABLE MITARBEITER ADD CONSTRAINT "PRJ_CONSTRAINT"   '||'
    FOREIGN KEY ("PROJ_MIT")                                    '||'
    REFERENCES "PROJEKTDATEN"."PROJEKT" ("PROJEKT_NR")' ;
/* */
EXEC 'ALTER TABLE MITARBEITER ADD CONSTRAINT "ABT_CONSTRAINT"   '||'
    FOREIGN KEY ("ABT_MIT_NR")                                  '||'
    REFERENCES "STAMMDATEN"."ABTEILUNG" ("ABTEILUNG_NR")' ;
/* */
EXEC 'COMMIT WORK' ;
END PROC;
```

2.6.5 Beispiel-Programme

Die DRIVE V1.1-Programme (siehe DRIVE-Programmiersprache V1.1) laufen auch unter der Version 2.1 ab, wenn Sie die SQL-Umgebung einstellen mit den Anweisungen:

```
PARAMETER DYNAMIC AUTHORIZATION=PERSONALLEITER;

PARAMETER DYNAMIC CATALOG=PERSONALVERWALTUNG;

PARAMETER DYNAMIC SCHEMA=STAMMDATEN;
```

Diese PARAMETER-Anweisungen gelten für statische und dynamische Anweisungen, wenn in den Programmen keine OPTION-Anweisung mit diesen Operanden vorhanden ist. Ansonsten müssen für die statischen Anweisungen die Anweisung OPTION und für die dynamischen Anweisungen SET-Anweisungen angegeben werden, um AUTHORIZATION, CATALOG und SCHEMA einzustellen.

Bei Remote-Zugriff müssen Sie zusätzlich das Datenbanksystem einstellen, z.B. mit der Anweisung:

```
OPTION DBSYSTEM=SESAMSQL
```

Programmname: MITARBEITER.HAUPT

```
PROCEDURE MITARBEITER;
COPY MITVAR;
DECLARE VARIABLE &ENDE2 CHAR(1);
DCL SCREEN FHSBILD;
DECLARE FORM NEUBILD
  TTITLE NL 1,
       TAB 30,'M I T A R B E I T E R',NL 2,
       '  NEUAUFNAHME',
       TAB 60,'FORMULAR 02',NL 1,
       '  BEENDEN ( E ) : ',RETURN &ENDE2,NL 1,
       ' ','-'(70),NL 3,
       '  ABT_MIT_NR     : ',RETURN &AABT_MIT_NR,NL 1,
       '  NACHNAME       : ',RETURN &ENACHNAME,NL 1,
       '  VORNAME        : ',RETURN &EVORNAME,NL 1,
       '  LAND           : ',RETURN &ELAND,NL 1,
       '  STRASSE        : ',RETURN &ESTRASSE,NL 1,
       '  PLZ            : ',RETURN &EPLZ,NL 1,
       '  ORT            : ',RETURN &EORT,NL 1,
       '  GEHALT         : ',RETURN &EGEHALT,NL 1,
       '  SPRACHEN       : ',RETURN &ESPRACHEN(1),',',
                        RETURN &ESPRACHEN(2),',',
                        RETURN &ESPRACHEN(3),',',
                        RETURN &ESPRACHEN(4),',',NL 1,
       '  ABT_LEITER     : ',RETURN &EABT_LEITER,NL 1,
       '  PROJ_MIT       : ',RETURN &EPROJ_MIT,NL 1
  BTITLE ' ','-'(70),NL 1;
SUBPROCEDURE MITARBEITERAUFNAHME;
 SET &ENDE2 = ' ';
 CYCLE;
  DISPLAY NEUBILD;
  IF &ENDE2 = 'E'
    THEN BREAK SUBPROCEDURE;
  END IF;
  INSERT INTO MITARBEITER VALUES ( &AABT_MIT_NR,*,&AUFNAHMESATZ.*);
  IF &DML_STATE = 'SQL ERROR' THEN
   DISPLAY FORM &DML_STATE,
            ': DER DATENSATZ WURDE NICHT AUFGENOMMEN (',
            &SQL_STATE, ') - DUE-TASTE';
  ELSE IF &DML_STATE <> 'OK' THEN
       SEND MESSAGE 'FEHLER ',&SQL_STATE,
                ' BEI DATENSATZAUFNAHME - DUE-TASTE';
     ELSE COMMIT WORK;
       SEND MESSAGE
       'DER DATENSATZ WURDE AUFGENOMMEN - DUE-TASTE';
     END IF;
  END IF;
```

```
  END CYCLE;
END SUBPROCEDURE;
WHENEVER &DML_STATE CONTINUE;
CYCLE;
  DISPLAY FHSBILD;
  IF &WAHL = '0' THEN BREAK CYCLE;
  END IF;
  IF &WAHL = '1' THEN CALL MITARBEITERAUFNAHME; SET &WAHL = ' ';
  END IF;
  IF &WAHL = '2' THEN CALL MITKORR; SET &WAHL = ' ';
  END IF;
  IF &WAHL = '3' THEN CALL MITLOES; SET &WAHL = ' ';
  END IF;
  IF &WAHL = '4' THEN CALL MITANZ; SET &WAHL = ' ';
  END IF;
  IF &WAHL = '5' THEN CALL MITDRUCK; SET &WAHL = ' ';
  END IF;
END CYCLE;
ROLLBACK WORK;
END PROCEDURE;
```

Name des COPY-Elementes: MITVAR

```
DECLARE VARIABLE &FRAGE PERMANENT CHAR(1) INIT 'N';
DECLARE MITARBEITER CURSOR FOR S ALL * FROM MITARBEITER;
DECLARE DRUCKCURSOR CURSOR FOR S ABT_MIT_NR,LFD_NR,NACHNAME,VORNAME,
          GEHALT,ABT_LEITER,PROJ_MIT FROM MITARBEITER;
DECLARE VARIABLE &MITARBEITERSATZ LIKE CURSOR MITARBEITER,
          &INDEX INT INIT 1,
          &NUMMER INT,
          &VGL CHAR(4),
          &DATUM DATE
           MASK 'Q(10)",DER "ZD"."R(10)"."YYYY',
          &ZEIT TIME
           MASK 'ZH" UHR "ZI',
          1 &SAETZE,
           2 AABT_MIT_NR CHAR(4) INIT '0000',
           2 AUFNAHMESATZ,
            3 ENACHNAME CHAR(20),
            3 EVORNAME CHAR(20) ,
            3 ELAND CHAR(3),
            3 ESTRASSE CHAR(26),
            3 EPLZ CHAR(10),
            3 EORT CHAR(20),
            3 EGEHALT NUM(7,2) CHECK &EGEHALT < 20000 MESSAGE
              '       DAS GEHALT IST ZU HOCH ! ',
            3 ESPRACHEN(4) CHAR(3) INIT '---'
```

```
                CHECK &ESPRACHEN <> '   ' MESSAGE
                '    GEBEN SIE "---" ALS LEERSTELLE EIN',
              3 EABT_LEITER CHAR(8),
              3 EPROJ_MIT CHAR(6) INIT '------'
                CHECK &EPROJ_MIT <> '      ' MESSAGE
                '    GEBEN SIE "------" ALS LEERSTELLE EIN',
            2 DRUCKSATZ,
              3 ABT_MIT_NR_ CHAR(4) REDEFINES &AABT_MIT_NR,
              3 LFD_NR_ INT,
              3 NACHNAME_ CHAR(20) ,
              3 VORNAME_ CHAR(20) ,
              3 GEHALT_ NUM(7,2) MASK 'ZZZZ9P99" DM '",
              3 ABT_LEITER_  CHAR(8) ,
              3 PROJ_MIT_ CHAR(6) ,
            &ABFRAGEBEFEHL CHAR(15) INIT 'DISPLAY FORM ',
            &ABFRAGETEXT1 CHAR(74) INIT
'NL 14,TAB 5,"GEBEN SIE DIE ABT_MIT_NR EIN : ",RETURN &VGL,',
            &ABFRAGETEXT2 CHAR(74) INIT
'NL  2,TAB 5,"GEBEN SIE DIE LAUFENDE NUMMER EIN : ",RETURN &NUMMER;',
            &FEHLERMELDUNG CHAR(74) INIT
'SEND MESSAGE "  DER DATENSATZ IST NICHT VORHANDEN. DUE-TASTE";',
            &DRUCKTEXT CHAR(74) INIT
'NL 15,TAB 15,"SOLL JETZT GEDRUCKT WERDEN ? (J/N) : ",RETURN &FRAGE';
```

Name des externen Unterprogramms: MITKORR

```
PROCEDURE MITARBEITERKORREKTUR;
COPY MITVAR;
DECLARE FORM KORRBILD
  TTITLE NL 1,
       TAB 30,'M I T A R B E I T E R',NL 2,
       '  KORREKTUR VON MITARBEITERDATEN',TAB 60,'FORMULAR 03',NL 2,
       ' ','-'(70),NL 1,
        '  ABT_MIT_NR     : ',&ABT_MIT_NR,NL 1,
        '  LFD_NR         : ',&LFD_NR,NL 1,
        '  NACHNAME       : ',RETURN &NACHNAME,NL 1,
        '  VORNAME        : ',RETURN &VORNAME,NL 1,
        '  LAND           : ',RETURN &LAND,NL 1,
        '  STRASSE        : ',RETURN &STRASSE,NL 1,
        '  PLZ            : ',RETURN &PLZ,NL 1,
        '  ORT            : ',RETURN &ORT,NL 1,
        '  GEHALT         : ',RETURN &GEHALT,NL 1,
        '  SPRACHEN       : ',RETURN &SPRACHEN(1),',',
                          RETURN &SPRACHEN(2),',',
                          RETURN &SPRACHEN(3),',',
                          RETURN &SPRACHEN(4),',',NL 1,
        '  ABT_LEITER     : ',RETURN &ABT_LEITER,NL 1,
        '  PROJ_MIT       : ',RETURN &PROJ_MIT,NL 1
   BTITLE ' ','-'(70),NL 1;
WHENEVER &DML_STATE CONTINUE;
EXEC CONCAT(&ABFRAGEBEFEHL,CONCAT(&ABFRAGETEXT1,&ABFRAGETEXT2));
SELECT ALL * INTO &MITARBEITERSATZ.* FROM MITARBEITER
     WHERE (ABT_MIT_NR = &VGL) AND (LFD_NR = &NUMMER);
IF &DML_STATE <> 'OK' THEN
  SEND MESSAGE 'FEHLER ',&SQL_STATE;
  EXEC &FEHLERMELDUNG;
  SET &VGL = ' ';
  BREAK PROCEDURE;
END IF;
CYCLE WHILE &INDEX < 5;
  IF &SPRACHEN(&INDEX) IS NULL THEN
     SET &SPRACHEN(&INDEX) = '---';
  END IF;
  SET &INDEX = &INDEX + 1;
END CYCLE;
SET &INDEX = 1;
IF &PROJ_MIT IS NULL THEN
  SET &PROJ_MIT = '------';
END IF;
```

```
DISPLAY KORRBILD;
UPDATE MITARBEITER
 SET   NACHNAME   = &NACHNAME,
       VORNAME    = &VORNAME,
       LAND       = &LAND,
       STRASSE    = &STRASSE,
       PLZ        = &PLZ,
       ORT        = &ORT,
       GEHALT     = &GEHALT,
       SPRACHEN(1)= &SPRACHEN(1),
       SPRACHEN(2)= &SPRACHEN(2),
       SPRACHEN(3)= &SPRACHEN(3),
       SPRACHEN(4)= &SPRACHEN(4),
       ABT_LEITER = &ABT_LEITER,
       PROJ_MIT   = &PROJ_MIT
 WHERE (ABT_MIT_NR = &VGL) AND (LFD_NR = &NUMMER);
IF &DML_STATE <> 'OK' THEN
 ROLLBACK WORK;
 SEND MESSAGE 'FEHLER ',&SQL_STATE;
ELSE COMMIT WORK;
END IF;
END PROCEDURE;
```

Name des externen Unterprogramms: MITLOES

```
PROCEDURE MITARBEITERLOESCHEN ;
COPY MITVAR;
DECLARE FORM DELBILD
  TTITLE NL 1,
       TAB 30,'M I T A R B E I T E R',NL 2,
       ' FOLGENDER MITARBEITERSATZ',TAB 60,'FORMULAR 04',NL 1,
       ' SOLL GELOESCHT WERDEN : ',NL 2,
        ' ABT_MIT_NR    : ',&ABT_MIT_NR,NL 1,
        ' LFD_NR        : ',&LFD_NR,NL 1,
        ' NACHNAME      : ',&NACHNAME,NL 1,
        ' VORNAME       : ',&VORNAME,NL 1,
        ' LAND          : ',&LAND,NL 1,
        ' STRASSE       : ',&STRASSE,NL 1,
        ' PLZ           : ',&PLZ,NL 1,
        ' ORT           : ',&ORT,NL 1,
        ' GEHALT        : ',&GEHALT,NL 1,
        ' SPRACHEN      : ',&SPRACHEN(1),',',
                          &SPRACHEN(2),',',
                          &SPRACHEN(3),',',
                          &SPRACHEN(4),',',NL 1,
        ' ABT_LEITER    : ',&ABT_LEITER,NL 1,
        ' PROJ_MIT      : ',&PROJ_MIT,NL 1
```

```
   BTITLE ' ','-'(70),NL 1,
        TAB 50,'LOESCHEN ( J/N ) : ',RETURN &FRAGE;
WHENEVER &DML_STATE CONTINUE;
EXEC CONCAT(&ABFRAGEBEFEHL,CONCAT(&ABFRAGETEXT1,&ABFRAGETEXT2));
SELECT ALL * INTO &MITARBEITERSATZ.* FROM MITARBEITER
     WHERE (ABT_MIT_NR = &VGL) AND (LFD_NR = &NUMMER);
ROLLBACK WORK;
IF &DML_STATE <> 'OK' THEN
 SEND MESSAGE 'FEHLER ',&SQL_STATE;
 EXEC &FEHLERMELDUNG;
 BREAK PROCEDURE;
END IF;
DISPLAY DELBILD;
IF &FRAGE = 'J' THEN
 DELETE FROM MITARBEITER WHERE (ABT_MIT_NR = &VGL)
                  AND (LFD_NR = &NUMMER);
 IF &DML_STATE <> 'OK' THEN
  ROLLBACK WORK;
  SEND MESSAGE 'FEHLER ',&SQL_STATE;
 ELSE COMMIT WORK;
 END IF;
END IF;
SET &FRAGE = 'N';
END PROCEDURE;
```

Name des externen Unterprogramms: MITANZ

```
PROCEDURE MITARBEITERANZEIGEN;
DECLARE VARIABLE &L(4) CHAR(1),
          &G(4) CHAR(1),
          &H1(4) CHAR(3),
          &G_UNTER NUM(7,2) ,
          &G_OBER NUM(7,2) ,
          &INDEX NUM(1) INIT 1,
          &HLAND CHAR(3),
          &SUCHE1 CHAR(60),
          &SUCHE2 CHAR(60),
          &SUCHE CHAR (120),
          &CURSORDEC1 CHAR(70) INIT
   'DECLARE ANZ CURSOR FOR S LAND,NACHNAME,VORNAME,GEHALT ',
          &CURSORDEC2 CHAR(40) INIT
   'FROM MITARBEITER WHERE ',
          &CURSORDEC CHAR(110),
          1 &ANZEIGESATZ,
           2 LAND CHAR(3),
           2 VORNAME CHAR(20),
           2 NACHNAME CHAR(20),
```

```
            2 GEHALT NUM(7,2);
DECLARE ANZ CURSOR;
DECLARE FORM MITARBEITERABFRAGE
    TTITLE NL 1,
        TAB 30,'M I T A R B E I T E R',NL 4,
        TAB 10,'KREUZEN SIE ZUTREFFENDES AN :',NL 3,
        TAB 10,'DEUTSCHLAND ',RETURN &L(1),
        TAB 35,'GEHALT ZWISCHEN 2000 UND 4000 ',RETURN &G(1),NL 2,
        TAB 10,'USA ',RETURN &L(2),
        TAB 35,'GEHALT ZWISCHEN 4000 UND 6000 ',RETURN &G(2),NL 2,
        TAB 10,'SCHWEIZ ',RETURN &L(3),
        TAB 35,'GEHALT ZWISCHEN 6000 UND 8000 ',RETURN &G(3),NL 2,
        TAB 10 ,'ENGLAND ',RETURN &L(4),
        TAB 35,'GEHALT UEBER 8000 ',RETURN &G(4)
    BTITLE ' ','-'(70),NL 1;
DECLARE FORM MITARBEITERAUSGABE
   TTITLE NL 1,
       ' '(30),'MITARBEITERANZEIGE',NL 1,
       ' '(29),'-'(20)
   BTITLE '='(80);
DISPLAY MITARBEITERABFRAGE;
SET &H1(1) = 'FRG';
SET &H1(2) = 'USA';
SET &H1(3) = 'CH';
SET &H1(4) = 'ENG';
CYCLE WHILE &INDEX < 5;
  IF &L(&INDEX) <> ' ' THEN
     SET &HLAND = &H1(&INDEX);
  END IF;
  IF &G(&INDEX) <> ' ' THEN
     SET &G_UNTER = &INDEX * 2000;
     SET &G_OBER = &G_UNTER +2000;
  END IF;
  SET &INDEX = &INDEX + 1;
END CYCLE;
IF &G_OBER = 10000 THEN
  SET &G_OBER =20000;
END IF;
SET &SUCHE1 =' (LAND = &HLAND)';
SET &SUCHE2 =' (GEHALT >= &G_UNTER) AND (GEHALT <= &G_OBER)';
IF (&HLAND = ' ') AND (&G_UNTER <> 0) THEN
  SET &SUCHE = &SUCHE2;
END IF;
IF (&HLAND <> ' ') AND (&G_UNTER = 0) THEN
  SET &SUCHE = &SUCHE1;
END IF;
```

```
IF (&HLAND <> ' ') AND (&G_UNTER <> 0) THEN
   SET &SUCHE = CONCAT(&SUCHE1,CONCAT(' AND ',&SUCHE2));
END IF;
IF (&HLAND = ' ') AND (&G_UNTER = 0) THEN
   SET &SUCHE = ' (GEHALT > 1) ';
END IF;
SET &CURSORDEC = CONCAT(&CURSORDEC1,&CURSORDEC2);
EXEC CONCAT (&CURSORDEC,CONCAT(&SUCHE,';'));
CYCLE ANZ INTO &ANZEIGESATZ.*;
 IF &DML_STATE <> 'OK' THEN
   ROLLBACK WORK;
   SEND MESSAGE 'FEHLER ',&SQL_STATE;
   BREAK PROCEDURE;
 END IF;
 FILL MITARBEITERAUSGABE TABLE VALUES &ANZEIGESATZ;
END CYCLE;
DROP CURSOR ANZ;
COMMIT WORK;
DISPLAY MITARBEITERAUSGABE;
END PROCEDURE;
```

Name des externen Unterprogramms: MITDRUCK

```
PROCEDURE MITARBEITERDRUCK;
COPY MITVAR;
DECLARE LIST MITARBEITERAUSGABE
   TTITLE &DATUM,TAB 37,&ZEIT,TAB 110,'SEITE:',&PAGES,NL 3,
        TAB 60,'MITARBEITERLISTE',NL 1,
        TAB 59,'-'(18),NL 2
   BTITLE NL 2,'='(123);
FILL MITARBEITERAUSGABE TABLE NAMES &DRUCKSATZ;
CYCLE DRUCKCURSOR INTO &DRUCKSATZ.*;
  FILL MITARBEITERAUSGABE TABLE VALUES &DRUCKSATZ;
END CYCLE;
COMMIT WORK;
DISPLAY MITARBEITERAUSGABE;
EXEC CONCAT(&ABFRAGEBEFEHL,&DRUCKTEXT);
IF &FRAGE = 'J' THEN
 SYSTEM 'PRINT *SYSLST';
 SEND MESSAGE 'DIE MITARBEITERLISTE WIRD AUSGEDRUCKT. DUE-TASTE';
 SET &FRAGE = 'N';
ELSE
 SEND MESSAGE
'DIE MITARBEITERLISTE WIRD AM ENDE DER BS2000-SITZUNG GEDRUCKT.';
END IF;
END PROCEDURE;
```

2.7 Pragmas

Dieses Kapitel beschreibt

- wozu Pragmas eingesetzt werden können,
- worin sich ihre Einsatzmöglichkeiten in DRIVE/WINDOWS von denen in ESQL/COBOL-Programmen und im Utility-Monitor unterscheiden,
- wie Pragmas in DRIVE/WINDOWS syntaktisch spezifiziert werden und
- wie DRIVE/WINDOWS Fehler im Umgang mit Pragmas behandelt.

2.7.1 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.

  i 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/WINDOWS zu einem Fehler führt (vgl. Abschnitt „Fehlerbehandlung“ auf Seite 66).

- 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.
- 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).

  i Da DRIVE/WINDOWS jedoch z.Zt. keine DDL-Anweisungen für CALL-DML-Tabellen unterstützt, kann es auch diese Funktionalität z.Zt. nicht unterstützen.

- Bearbeitung von Basistabellen im Zustand „check pending“ (vgl. SESAM V2-Handbuch „SQL-Sprachbeschreibung Teil2:Utilities“ [19]).

Da DRIVE/WINDOWS jedoch z.Zt. keine Utility-Anweisungen unterstützt, kann ein DRIVE-Anwender die Voraussetzung zu dieser Bearbeitung nicht erfüllen (nämlich in der aktuellen SQL-Session den Zustand „check pending“ hergestellt zu haben), so daß DRIVE/WINDOWS auch diese Funktionalität z.Zt. nicht unterstützen kann.

2.7.2 Syntaxunterschiede zu ESQL/COBOL und zum Utility-Monitor

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/WINDOWS mit einer eigenen DRIVE-SQL-Anweisung, der PRAGMA-Anweisung, vereinbart. Eine PRAGMA-Anweisung ist eine Programmanweisung, die zum Übersetzungszeitpunkt ausgewertet wird. Dies ist bei einer statischen PRAGMA-Anweisung der Zeitpunkt der expliziten Sourceübersetzung (COMPILE-Anweisung) oder der impliziten Sourceübersetzung (DO- oder CALL-Anweisung: Vorstufe der Prozedurausführung), bei einer dynamischen PRAGMA-Anweisung die implizite Anweisungsübersetzung (EXECUTE-Anweisung: Generierung und Übersetzung als Vorstufe der Anweisungsausführung). Die genaue Wirkungsweise ist im Kapitel „DRIVE-SQL-Anweisungen“ auf Seite 120 bei der PRAGMA-Anweisung beschrieben.

2.7.3 Statische und dynamische Pragmas

Statische PRAGMA-Anweisungen wirken nur auf statische SQL-Anweisungen, dynamische PRAGMA-Anweisungen wirken nur auf dynamische SQL-Anweisungen.

2.7.4 Pragmaklauseln

Da Pragmas in DRIVE/WINDOWS nur über die PRAGMA-Anweisung

PRAGMA ' { pragma-klausel }, ... '

vereinbart werden können, werden im folgenden im Unterschied zur SESAM V2-Handbuchreihe Pragmaklauseln statt Pragmas beschrieben.

Pragmaklausel PREFETCH

Die Pragmaklausel PREFETCH hat folgende Form:

PREFETCH *n*

Mit dieser Klausel kann der sog. „Schubmodus“ aktiviert werden, indem die Klausel einer (statischen oder dynamischen) DECLARE CURSOR- Anweisung zugeordnet wird. Beim ersten FETCH nach OPEN dieses Cursors versucht dann SESAM bis zu n viele Sätze der Cursortabelle „im voraus“ zu lesen, um nachfolgende FETCH-Anweisungen zu beschleunigen. Diese Funktionalität ist identisch mit der der PREFETCH-Klausel innerhalb der DECLARE CURSOR-Anweisung (siehe Beschreibung in Kapitel „DRIVE-SQL-Anweisungen“ auf Seite 88).

Die „Schubmodus“-Funktionalität wird nur wirksam, wenn mit einer SESAM-Version ab 2.1 gearbeitet wird. Bei SESAM/SQL V2.0 führen die PREFETCH-Klausel innerhalb von DECLARE CURSOR und die Pragmaklausel PREFETCH zu DRIVE-Fehlern: die Klauseln werden zwar von DRIVE/WINDOWS zugelassen (keine DRIVE-Übersetzungsfehler), führen aber zu SESAM-Warnungen und damit zu DRIVE-Fehlern

– bei der SESAM-Vorübersetzung zum Übersetzungszeitpunkt (statische Cursor und Pragmas) und
– bei der SESAM-Präparierung zum Ausführungszeitpunkt (dynamische Cursor und Pragmas und variable Cursor).

Pragmaklausel EXPLAIN

SESAM V2 erstellt für die meisten New-Style-SQL-Anweisungen intern eine Auswertungsvorschrift, den SQL-Zugriffsplan. Der SQL-Optimizer sorgt bei DML-Anweisungen (SELECT bzw. Cursorverarbeitung, INSERT, UPDATE, DELETE) dafür, daß ein besonders effizienter Zugriffsplan erstellt wird, bei dem möglichst wenig Systemressourcen beansprucht werden.

Mit der Erklärungskomponente des Optimizers kann man untersuchen, in welchen Einzelschritten eine DML-Anweisung abgearbeitet wird. Wenn man eine Pragmaklausel EXPLAIN vor der Anweisung vereinbart, erhält man eine lesbare Darstellung des internen Zugriffsplans auf Datei. Die Pragmaklausel EXPLAIN hat folgende Form:

EXPLAIN INTO *dateiname*

Die Ausgabe der Erklärungskomponente und die darin enthaltene Information ist im Kapitel 7 des SESAM V2-Handbuchs „Performance“ [45] beschrieben.

Pragmaklauseln zur Beeinflussung des Zugriffsplans

Mit den folgenden Klauseln kann der Ausführungsplan (SQL-Zugriffsplan) zur Abarbeitung einer New-Style-DML-Anweisung (SELECT bzw. Cursorverarbeitung, INSERT, UPDATE, DELETE) beeinflußt werden:

– Pragmaklausel IGNORE INDEX

  OPTIMIZATION LEVEL *n*

– Pragmaklausel OPTIMIZATION LEVEL

  IGNORE INDEX *indexname*

– Pragmaklausel SIMPLIFICATION

  SIMPLIFICATION { ON | OFF }

(vgl. SESAM V2-Handbuch „Performance“ [45]).

Pragmaklausel ISOLATION LEVEL

Die Pragmaklausel ISOLATION LEVEL hat folgende Form:

```
ISOLATION LEVEL { READ UNCOMMITTED |
                  READ COMMITTED   |
                  REPEATABLE READ  |
                  SERIALIZABLE     }
```

Mit dieser Klausel kann der Isolationslevel für Datenbankzugriffe einer New-Style-DML-Anweisung unabhängig vom Isolationslevel der Transaktion festgelegt werden, in der die Anweisung ausgeführt wird.

Pragmaklausel DATA TYPE

Die Pragmaklausel DATA TYPE hat folgende Form:

```
DATA TYPE OLDEST
```

Mit dieser Klausel können zu einer Nur-CALL-DML-Tabelle Spalten vom Oldest-Style-Typ hinzugefügt werden. Die Verwendung dieser Klausel führt jedoch in DRIVE/WINDOWS stets zu einem Fehler mit &DML_STATE = 'SQL ERROR'.

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. SESAM V2- Handbuch „SQL-Sprachbeschreibung Teil2: Utilities“ [19]). Die Verwendung dieser Klausel führt jedoch in DRIVE/WINDOWS stets zu einem Fehler mit &DML_STATE = 'SQL ERROR'.

2.7.5 Fehlerbehandlung

Beim Umgang mit Pragmas können folgende Fehler gemacht werden:

– Die PRAGMA-Anweisung kann syntaktisch inkorrekt angegeben werden.

  In diesem Fall meldet DRIVE/WINDOWS 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 (vgl. Kapitel „DRIVE-SQL-Anweisungen“ auf Seite 120 für die genaue Wirksamkeit der PRAGMA-Anweisung). 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/WINDOWS einen SQLSTATE der Klasse 01 (Warnung) und vorübersetzt (statisches Pragma) bzw. präpariert (dynamisches Pragma) die SQL-Anweisung ohne Pragmawirkung. Tritt hierbei ein SESAM-Fehler auf, so geht die Warnung und damit der Pragmafehler verloren.

– 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/WINDOWS 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/WINDOWS ebenfalls einen SQLSTATE der Klasse 01 (Warnung) und vorübersetzt bzw. präpariert die SQL-Anweisung ohne Pragmawirkung.

DRIVE/WINDOWS setzt nun alle Warnungen von SESAM (SQLSTATE-Klasse 01) in DRIVE-SQL-Fehler mit &DML_STATE = 'SQL ERROR' und &ERROR = 'OK' um.

Zum Übersetzungszeitpunkt (statische Pragmas) hat dies die Konsequenz, daß die Übersetzung der betroffenen SQL-Anweisung als fehlerhaft gilt und die von SESAM gemeldete Warnung als DRIVE-SQL-Fehler (Meldungsnummer DRI0536) in der Übersetzungsliste festgehalten wird. Die von SESAM ohne Pragmawirkung vorübersetzte SQL-Anweisung wird daher von DRIVE/WINDOWS nicht übernommen.

Zum Ausführungszeitpunkt (dynamische Pragmas) hat dies die Konsequenz, daß im Programm-(bzw. Debug-)Modus alle SESAM-Warnungen WHENEVER-fähig sind. Insbesondere sind also auch alle Pragmafehler WHENEVER-fähig (siehe DRIVE-Programmiersprache [2] , Kapitel „Programmierlogik“). Hierbei ist allerdings folgendes zu beachten:

– Ist für das Fehlerereignis &DML_STATE = 'SQL ERROR' keine WHENEVER-Aktion oder ist die Aktion BREAK definiert, so bricht DRIVE/WINDOWS das Programm ab (bzw. verzweigt zum End-Haltepunkt).

  Ist dabei noch eine Transaktion offen, so wird sie von DRIVE/WINDOWS zurückgesetzt. Insbesondere ist damit die eventuelle Ausführung der betroffenen SQL-Anweisung durch SESAM ohne PRAGMA-Wirkung zurückgesetzt.

– Ist für &DML_STATE = 'SQL ERROR' die WHENEVER-Aktion CONTINUE definiert, so setzt DRIVE/WINDOWS das Programm mit der auf die betroffene SQL-Anweisung folgende SQL- oder DRIVE-Anweisung fort.

  Wird anschließend ein COMMIT WORK durchlaufen, so ist also die Ausführung der betroffenen SQL-Anweisung durch SESAM ohne PRAGMA-Wirkung festgeschrieben.

– Ist für &DML_STATE = 'SQL ERROR' als WHENEVER-Aktion der Aufruf eines internen Unterprogramms vereinbart, so hängt es von der Logik dieses Unterprogramms ab, wie die SESAM-Warnung aufgrund des Pragmafehlers behandelt wird:

  – Wird (ggf. nach diversen DRIVE-Anweisungen zur Fehlerbehandlung) die Transaktion zurückgesetzt (ROLLBACK WORK), so ist auch die Ausführung der betroffenen SQL-Anweisung durch SESAM ohne PRAGMA-Wirkung zurückgesetzt.

– Wird die Transaktion schließlich festgeschrieben (COMMIT WORK), so ist auch die Ausführung der betroffenen SQL-Anweisung durch SESAM ohne PRAGMA-Wirkung festgeschrieben.

– Wird das interne Unterprogramm ohne ROLLBACK WORK oder COMMIT WORK beendet oder abgebrochen, so gilt das für die WHENEVER- Aktion CONTINUE Gesagte.

3 DRIVE-SQL-Anweisungen

Dieses Kapitel beschreibt zwei Klassen von SQL-Anweisungen:

– alle SQL-Anweisungen von SESAM/SQL V2, die DRIVE/WINDOWS unterstützt, ggf. mit Hinweisen auf Einschränkungen oder Erweiterungen.

– SQL-Anweisungen von SESAM/SQL V1, die DRIVE/WINDOWS aus Kompatibilitäts- oder Performancegründen weiterhin unterstützt (insbesondere DROP-Anweisungen).

In ESQL/COBOL-Programmen und im Utility-Monitor werden SQL-Anweisungen nicht mit einem Semikolon abgeschlossen. Dies ist in DRIVE/WINDOWS nur im Dialog-Modus und bei dynamischen Programmanweisungen möglich. In DRIVE-Programmen müssen statische SQL-Anweisungen immer mit einem Semikolon abgeschlossen werden.

In ESQL/COBOL werden SQL-Anweisungen zwischen den Schlüsselwörtern „EXEC SQL“ und „END-EXEC“ eingeschlossen. In Anweisungsdateien für den Utility-Monitor werden SQL-Anweisungen mit dem Schlüsselwort SQL eingeleitet, Fortsetzungszeilen müssen mit dem Zeichen „-“ verbunden werden (mit Verdoppelungsregel innerhalb von SQL-Ausdrükken). Dies ist in DRIVE/WINDOWS nicht erforderlich: die Datenbanksprache SQL ist in die DRIVE-Sprache, eine Programmiersprache der vierten Generation, vollständig integriert.

In ESQL/COBOL-Programmen und Utilitiy-Monitor-Anweisungsdateien werden Kommentarzeilen mit dem Zeichen „*“ eingeleitet. Innerhalb von oder vor SQL-Anweisungen können Kommentare mit der Folge „--“ eingeleitet werden. Ende des Kommentars ist das Zeilenende. In DRIVE/WINDOWS werden Kommentare mit der Folge „/*“ eingeleitet und müssen mit der Folge „*/“ abgeschlossen werden.

In der Regel werden Syntaxverstöße bei SQL-Anweisungen von DRIVE/WINDOWS festgestellt und wie Analysefehler von DRIVE-Anweisungen behandelt (vgl. DRIVE-Programmiersprache [2], Abschnitt „DRIVE-Programme entwickeln“). Hier liegt dann ein Verstoß gegen eine DRIVE-Regel vor, d.h. gegen den Sprachaufbau der Anweisung (Verwendung und Schreibweise von Schlüsselwörtern, Reihenfolge von Klauseln usw.), auf den mit einer entsprechenden DRIVE-Fehlermeldung hingewiesen wird. Bei manchen SQL-Anweisungen können aber auch Syntaxverstöße von SESAM mit einem entsprechenden SQLSTATE gemeldet werden (insbesondere SQLSTATE-Klassen 42 und 01). Hier liegt dann ein Verstoß gegen eine SESAM-Regel vor (semantische Abhängigkeiten, Kontextverstöße, falsche SQL-Metadaten, fehlende Zugriffsrechte, Pragma-Klauseln fehlerhaft oder falsch eingesetzt usw.).

ALTER TABLE - Basistabelle ändern

ALTER TABLE ändert eine bestehende Basistabelle. Sie können Spalten hinzufügen oder ändern und Integritätsbedingungen hinzufügen oder löschen.

Bei einer CALL-DML-Tabelle können Sie nur Spalten ändern. Einschränkungen dabei sind weiter unten beschrieben.

Welche Basistabellen definiert sind, erfahren Sie im View BASE_TABLES des INFORMATION_SCHEMA (siehe SESAM/SQL-Server, SQL-Sprachbeschreibung Teil1 [20], Kapitel „Informationsschemata“).

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.

DRIVE/WINDOWS unterstützt die ALTER TABLE-Anweisung von SESAM/SQL eingeschränkt:

– es können keine Spalten zu CALL-DML-Tabellen hinzugefügt werden

– es kann nur eine Spalte hinzugefügt oder geändert werden

Die entsprechenden Möglichkeiten bei SESAM/SQL stellen eine Erweiterung der SQL2-Norm [47] dar.

```
ALTER TABLE tabelle

 { ADD [ COLUMN ] spaltendefinition |

   ALTER [ COLUMN ] spalte
         { DROP DEFAULT |
           SET grunddatentyp |
           SET voreinstellung } |

   ADD [ CONSTRAINT integritätsbedingungsname ] tabellenbedingung |

   DROP CONSTRAINT integritätsbedingungsname RESTRICT }
```

```
voreinstellung::= DEFAULT { literal |
                  CURRENT DATE |
                  CURRENT TIME |
                  CURRENT TIMESTAMP |
                  [ CURRENT ] USER | SYSTEM USER |
                  NULL }
```

---

tabelle
Name einer Basistabelle (siehe Metavariable *tabellenangabe*, insbesondere wegen Voll- und Teilqualifizierung).

ADD [COLUMN] *spaltendefinition*
Fügt der Basistabelle eine neue Spalte hinzu. Die neue Spalte wird am Ende der Basistabelle angefügt. *spaltendefinition* definiert die Spalte.

spaltendefinition darf nur dann eine DEFAULT-Klausel enthalten, wenn die Tabelle leer ist. Sie dürfen in *spaltendefinition* keine Primärschlüsselbedingung definieren.

Enthält die Tabelle bereits Sätze, wird der NULL-Wert in die neue Spalte eingetragen. Dabei werden die definierten Integritätsbedingungen geprüft.

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

Die Spalte darf nicht in Views, Indizes und Integritätsbedingungen vorkommen. Es werden alle temporären Views gelöscht, die auf der Tabelle basieren.

DROP DEFAULT
Löscht die Voreinstellung für die Spalte.

SET *grunddatentyp*
Neuer Datentyp der Spalte.

Die Dimension einer multiplen Spalte darf nicht geändert werden.

grunddatentyp kann nur CHARACTER, VARCHAR oder NUMERIC sein. Bei CHARACTER und VARCHAR darf die neue (Maximal-)Länge nicht kleiner als die alte sein. Bei NUMERIC darf die neue Stellenanzahl nicht kleiner als die alte sein und muß die Anzahl der Nachkommastellen gleich bleiben.

spalte darf nicht in einer Viewdefinition, in einer Integritätsbedingung oder in einer Indexdefinition referenziert werden.

SET *voreinstellung*
Legt einen neuen SQL-Defaultwert für die Spalte fest.

– *spalte* darf keine multiple Spalte sein.

– *spalte* darf keine CALL-DML-Spalte sein.

– *voreinstellung* muß den Zuweisungsregeln für Defaultwerte genügen (siehe SESAM/SQL-Server SQL-Sprachbeschreibung Teil1 [18], Abschnitt „Defaultwerte für Tabellenspalten“).

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

ADD CONSTRAINT-Klausel
Fügt eine Integritätsbedingung zur Basistabelle hinzu.

CONSTRAINT *integritätsbedingungsname*
Vergibt einen Namen für die Integritätsbedingung. Der einfache Name der Integritätsbedingung muß innerhalb des Schemas eindeutig sein. Der Integritätsbedingungsname kann durch einen Datenbank- und Schemanamen qualifiziert werden. Dieser Datenbank- und Schemaname muß mit dem Datenbank- und Schemanamen der Basistabelle übereinstimmen, zu der die Integritätsbedingung hinzugefügt wird.

CONSTRAINT *integritätsbedingungsname* nicht angegeben:
Die Integritätsbedingung erhält einen Namen nach der Regel:

{ UN | FK | CH } *integritätsbedingungsnummer*

wobei das zweistellige Präfix UN für UNIQUE, FK für FOREIGN KEY und CH für CHECK steht. *integritätsbedingungsnummer* ist eine 16stellige Ganzzahl (Zeitstempel).

tabellenbedingung
Gibt eine Integritätsbedingung für die Tabelle an (siehe Metavariable *tabellenbedingung*). *tabellenbedingung* darf keine Primärschlüsselbedingung definieren.

DROP CONSTRAINT *integritätsbedingungsname* RESTRICT
Löscht die Integritätsbedingung *integritätsbedingungsname*.

Sie können weder die Primärschlüsselbedingung einer Tabelle löschen, noch dürfen Sie eine Eindeutigkeitsbedingung für eine Spalte löschen, solange eine Referenzbedingung einer anderen Tabelle sich auf diese Spalte bezieht.

Besonderheiten für CALL-DML-Tabellen

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

– Es ist nur die Angabe ALTER [COLUMN] *spalte* SET *grunddatentyp* erlaubt. Die Spalte *spalte* kann nicht multipel sein.

– Für *grunddatentyp* sind nur die Datentypen CHARACTER und NUMERIC erlaubt.

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;
```

Das folgende DRIVE-DDL-Programm fügt eine Spalte hinzu und ändert zwei Spalten.

```
OPTION AUTHORIZATION = x;

OPTION CATALOG = y;

/* Spalte in SQL-Tabelle laden */

ALTER TABLE A.T1
  ADD COLUMN C1 CHAR (20) NOT NULL;

/* Defaultwert in SQL-Tabelle ändern */

/* Vorher: Default: 'OLD' */

ALTER TABLE A.T2
  ALTER COLUMN C2 SET DEFAULT 'NEW';

/* Spalte in CALL-DMAL-Tabelle  ändern */

/* Vorher: CHAR (24) */

ALTER TABLE A.T3
  ALTER COLUMN C3 SET CHAR (30);
```

CLOSE - Cursor schließen

CLOSE schließt einen Cursor, der mit der Anweisung DECLARE vereinbart und mit OPEN oder RESTORE geöffnet wurde.

Die Cursorbeschreibung (siehe DECLARE-Anweisung) bleibt erhalten. Die aktuelle Cursorposition kann vor dem Schließen mit STORE gesichert werden.

Ein Cursor kann beliebig oft geschlossen und wieder geöffnet werden. Ein geöffneter Cursor wird auch bei Transaktionsende geschlossen.

Ein CLOSE mit nachfolgendem OPEN auf denselben Cursor ist z.B. sinnvoll, wenn Sie bei der Deklaration des Cursor Variablen in der *cursorbeschreibung* verwendet haben. *cursorbeschreibung* wird mit den zum OPEN-Zeitpunkt gültigen Werten der Variablen aktualisiert und die Cursortabelle mit den aktuellen Daten aus der Datenbank gefüllt.

Im Programm-Modus sind Cursor nur in der Quellprogrammdatei ansprechbar, in der sie mit DECLARE vereinbart werden.

CLOSE cursor

---

cursor
Name des zu schließenden Cursors.

COMMIT WORK - Transaktion beenden

COMMIT WORK beendet eine Transaktion und schreibt die während der Transaktion durchgeführten Änderungen in der Datenbank fest. Die geänderten Daten stehen damit allen anderen Transaktionen zur Verfügung.

COMMIT WORK schreibt außerdem alle Werte fest, die seit dem letzten Transaktionsende mit den Anweisungen SET sowie PARAMETER DYNAMIC und deren Operanden CATALOG/SCHEMA/AUTHORIZATION eingestellt wurden. Dadurch wird in dynamischen Programmen die SQL-Umgebung für nachfolgende Transaktionen festgeschrieben.

Mit der ersten transaktionseinleitenden (siehe unten) SQL-Anweisung nach COMMIT WORK beginnt eine neue Transaktion.

COMMIT WORK schließt alle innerhalb der Transaktion geöffneten Cursor. Ein mit TEMPORARY definierter Cursor wird in einem DRIVE-Programm beim nächsten COMMIT WORK auf höherer Programmebene gelöscht. Cursorpositionen von Nicht-PREFETCH-Cursorn können mit der Anweisung STORE gespeichert werden.

Zu Regeln in Verteilten Anwendungen siehe DRIVE-Programmiersprache [2], Kapitel „Verteilte Anwendungen“ bzw. „Verteilte Transaktionsverarbeitung“.

COMMIT [WORK] [WITH { display | send message | stop }]

WITH

Mit WITH kann eine Anweisung festgelegt werden, die nach Transaktionsende ausgeführt wird.

Wird WITH in einem Dialog-Programm im Betriebsmodus mit TP-Monitor (UTM-Betrieb) nicht angegeben, wird die Transaktion beendet und der Programmlauf im Folge-Teilprogramm ohne Bildschirmausgabe fortgesetzt.

WITH darf nur im Programm-Modus angegeben werden.

display

Ausgeben eines Formats. Folgende Anweisungen dürfen verwendet werden:

– DISPLAY *screenformat* (siehe DRIVE-Lexikon [3], Anweisung DISPLAY *screenformat*).

– DISPLAY *formatname* (siehe DRIVE-Lexikon [3], Anweisung DISPLAY *formatname*).

– DISPLAY FORM (siehe DRIVE-Lexikon [3], Anweisung DISPLAY FORM).

send message

Ausgeben von Meldungen (siehe DRIVE- Lexikon [3], Anweisung SEND MESSAGE).

stop

Beenden der DRIVE-Sitzung (siehe DRIVE-Lexikon [3], Anweisung STOP).

Transaktion

Eine Transaktion wird mit einer transaktionseinleitenden SQL-Anweisung begonnen. Alle folgenden SQL-Anweisungen bis zur nächsten COMMIT WORK- bzw. ROLLBACK WORK-Anweisung gehören zu derselben Transaktion. COMMIT WORK bzw. ROLLBACK WORK beendet die Transaktion.

Transaktion einleiten

Folgende DRIVE-SQL-Anweisungen leiten keine Transaktion ein:

– statische CREATE TEMPORARY VIEW (nicht ausführbar)
– statische DECLARE (nicht ausführbar)
– PERMIT
– PRAGMA (nicht ausführbar)
– SET CATALOG
– SET SCHEMA
– SET SESSION AUTHORIZATION
– SET TRANSACTION
– WHENEVER (nicht ausführbar)

Die Anweisung EXECUTE leitet nur dann eine Transaktion ein, wenn die jeweils auszuführende, dynamisch formulierte Anweisung eine Transaktion einleitet.

Jede andere SQL-Anweisung leitet eine Transaktion ein, wenn sie ausgeführt wird und noch keine Transaktion offen ist.

Anweisungen innerhalb von Transaktionen

Folgende Anweisungen dürfen nicht innerhalb einer Transaktion ausgeführt werden:
– PERMIT
– SET SESSION AUTHORIZATION
– SET TRANSACTION

Eine SQL-Anweisung zur Datenmanipulation (abfragen, ändern) darf nicht innerhalb einer Transaktion ausgeführt oder vorbereitet werden, in der eine SQL-Anweisung zur Definition oder Verwaltung von Schemas ausgeführt wird. D.h. das Mischen von DDL- und DML-Anweisungen in einer Transaktion ist nicht erlaubt.

Auswirkungen von COMMIT WORK

COMMIT WORK hat Auswirkungen auf nachfolgende Transaktionen und auf die in der Transaktion geöffneten Cursor und Voreinstellungen.

Auswirkungen auf nachfolgende Transaktionen

COMMIT WORK setzt Isolations- bzw. Konsistenzlevel und Transaktionsmodus, die mit einer SET TRANSACTION-Anweisung transaktionsspezifisch eingestellt wurden, wieder auf ihre voreingestellten Werte zurück. Eine nachfolgend begonnene Transaktion besitzt daher wieder den voreingestellten Isolations- bzw. Konsistenzlevel und Transaktionsmodus, wenn diese nicht wieder mit SET TRANSACTION geändert wurden.

Auswirkungen auf Cursor

COMMIT WORK schließt alle innerhalb der Transaktion geöffneten Cursor. Soll eine Cursorposition über das Transaktionsende hinaus gerettet werden, können Sie die Position mit der Anweisung STORE sichern und später mit RESTORE wiederherstellen, falls es sich nicht um einen PREFETCH-Cursor handelt.

Innerhalb der aktuellen Transaktion definierte Cursor (siehe DECLARE-Anweisung) werden festgeschrieben, soweit ihre Definitionen nicht wieder aufgehoben wurden (siehe DROP CURSOR-Anweisung). Aufhebungen von Cursordefinitionen werden festgeschrieben.

Auswirkungen auf temporäre Views

Innerhalb der aktuellen Transaktion definierte temporäre Sichten (siehe CREATE TEMPORARY VIEW-Anweisung) werden festgeschrieben, soweit ihre Definitionen nicht wieder aufgehoben wurden (siehe DROP TEMPORARY VIEW-Anweisung). Aufhebungen von Sichtendefinitionen werden festgeschrieben.

Auswirkungen auf Voreinstellungen

Mit SET CATALOG, SET SCHEMA und SET SESSION AUTHORIZATION definierte Voreinstellungen für dynamische Programme sind nach COMMIT WORK festgeschrieben.

Verhalten von SESAM/SQL im Fehlerfall

Kann eine Transaktion aufgrund eines Fehlers nicht ordnungsgemäß beendet werden, setzt SESAM/SQL die gesamte Transaktion zurück. Bei ROLLBACK WORK ist beschrieben, welche Datenbankobjekte davon betroffen sind.

Beispiel

Das folgende Beispiel zeigt, wie Sie einen Cursor in einer Schleife satzweise transaktionsgesichert bearbeiten. Dabei ist Cursorverarbeitung innerhalb einer Schleife mit COMMIT WORK nur erlaubt, wenn der Cursor mit STORE gespeichert und mit RESTORE wiederhergestellt wurde.

```
DECLARE c1 ...;
...
CYCLE c1 INTO &var.*;
  ...
  STORE c1;
  COMMIT WORK;
  RESTORE c1;
  ...
END CYCLE;
```

CREATE SCHEMA - Schema erzeugen

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

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

DRIVE/WINDOWS unterstützt die CREATE SCHEMA-Anweisung von SESAM eingeschränkt: Es kann keine *create_index_definition* verwendet werden, weil DRIVE/WINDOWS die SSL-Anweisung CREATE INDEX nicht unterstützt.

```
CREATE SCHEMA

  { [ catalog . ] schema [ AUTHORIZATION berechtigungsschlüssel ] |
    AUTHORIZATION berechtigungsschlüssel }

  [ { create_table_definition |
      create_view_definition |
      grant_definition } ... ]
```

---

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. Der Berechtigungsschlüssel darf max. 18 Zeichen lang sein.

AUTHORIZATION *berechtigungsschlüssel* nicht angegeben:
Bei einer statischen CREATE SCHEMA-Anweisung wird der über OPTION AUTHORIZATION wirksame Berechtigungsschlüssel Eigentümer des Schemas, bei einer dynamischen CREATE SCHEMA-Anweisung der über die letzte SET SESSION AUTHORIZATION-Anweisung wirksame Berechtigungsschlüssel (siehe auch Voreinstellungsmöglichkeit über PARAMETER DYNAMIC AUTHORIZATION).

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

create_table_definition
CREATE TABLE-Anweisung ohne abschließendes Semikolon, die eine Basistabelle für das Schema erzeugt. Der Tabellenname muß einfach sein oder darf nur mit den Datenbank- und Schemanamen aus der CREATE SCHEMA-Anweisung qualifiziert sein.

create_view_definition
CREATE VIEW-Anweisung ohne abschließendes Semikolon, die einen View für das Schema erzeugt. Der Viewname muß einfach sein oder darf nur mit den Datenbank- und Schemanamen aus der CREATE SCHEMA-Anweisung qualifiziert sein.

grant_definition
GRANT-Anweisung, die Privilegien für eine Basistabelle oder einen View des Schemas vergibt (DML-Rechte der Referenzrechte). Die GRANT-Anweisung darf keine Sonder-Privilegien vergeben.

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

Arbeitsweise von CREATE SCHEMA

Die Definitionen CREATE TABLE, CREATE VIEW und GRANT, die innerhalb der CREATE SCHEMA-Anweisung angegeben werden, werden genau in der angegebenen Reihenfolge ausgeführt. Definitionen, die sich auf bereits existierende Tabellen oder Views beziehen, müssen deshalb nach der Definition 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;
```

CREATE TABLE - Basistabelle erzeugen

CREATE TABLE erzeugt eine SQL-Basistabelle, in der die Daten permanent gespeichert sind. SQL-Tabellen können nur mit SQL und nicht mit CALL-DML bearbeitet werden.

Der aktuelle Berechtigungsschlüssel muß Eigentümer des Schemas sein, dem die Tabelle zugeordnet ist. Wird der Space für die Basistabelle angegeben, muß der aktuelle Berechtigungsschlüssel Eigentümer des Space sein.

DRIVE/WINDOWS unterstützt die CREATE TABLE-Anweisung von SESAM/SQL eingeschränkt: es können keine CALL-DML-Tabellen erzeugt werden

---

```
CREATE TABLE tabelle

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

    [ USING SPACE space ]
```

---

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. Der einfache Tabellenname kann durch einen Datenbank- und Schemanamen qualifiziert werden (siehe auch Metavariable *tabellenangabe*).

spaltendefinition

Definiert Spalten für die Basistabelle (siehe auch Metavariable *spaltendefinition*).

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 der Regel:

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

wobei das zweistellige Präfix UN für UNIQUE, PK für PRIMARY KEY, FK für FOREIGN KEY und CH für CHECK steht. *integritätsbedingungsnummer* ist eine 16stellige Ganzzahl (Zeitstempel).

tabellenbedingung
Definiert eine Integritätsbedingung, die für die Basistabelle gilt (siehe Metavariable *tabellenbedingung*).

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. *space* darf max. 18 Zeichen lang sein. Der einfache Spacename kann mit dem Datenbanknamen qualifiziert werden. Dieser Datenbankname muß mit dem 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 (siehe SESAM/SQL-Server SQL-Sprachbeschreibung Teil1 [18] und Teil 2 [19]).

Beispiel

Das Beispiel zeigt die CREATE TABLE-Anweisung für die 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)
    );
```

CREATE TEMPORARY VIEW - Temporären View deklarieren

CREATE TEMPORARY VIEW deklariert einen temporären View, der innerhalb der Quellprogrammdatei, in der er deklariert wurde, bzw. im Dialog-Modus verwendet werden kann. Ein temporärer View ist eine benannte Datenbankabfrage, die für die Dauer der DRIVE-Sitzung oder bis zu einer entsprechenden DROP TEMPORARY VIEW-Anweisung gespeichert wird. Zu Gültigkeitsbereich und Lebensdauer von temporären Views siehe DRIVE-Programmiersprache [2], Kapitel „Transaktionskonzept“.

Die Anweisung ist in Programmen statisch nur am Anfang des Programms erlaubt, d.h. vor den Verarbeitungsanweisungen. Zum Ablaufzeitpunkt können mit EXECUTE dynamische temporäre Views definiert werden.

Eigentümer eines statischen temporären Views ist der aktuelle Berechtigungsschlüssel zum Zeitpunkt der Übersetzung des Quellprogramms (siehe OPTION AUTHORIZATION). Insbesondere haben alle statischen temporären Views einer Übersetzungseinheit denselben Eigentümer. Eigentümer eines dynamischen temporären Views ist der aktuelle Berechtigungsschlüssel zum Zeitpunkt der Ausführung der dynamischen CREATE TEMPORARY VIEW-Anweisung.

Ein temporärer View wird dem Standardkatalog zugeordnet, der zum Zeitpunkt der Übersetzung (statischer View) bzw. Ausführung (dynamischer View) von CREATE TEMPORARY VIEW eingestellt ist (siehe OPTION CATALOG, SET CATALOG und PARAMETER DYNAMIC CATALOG) oder, höherprior, dem in *abfrageausdruck* durch direkte Qualifizierung referenzierten Katalog.

In einer DRIVE-Sitzung (TIAM-Session oder UTM-Vorgang) können pro SESAM-Datenbank nur dynamische temporäre Views für einen SQL-User deklariert werden. Wird daher nach einer erfolgreichen dynamischen CREATE TEMPORARY VIEW-Anweisung der SQL-User gewechselt (dies erfordert ein vorheriges Transaktionsende), so kann der neue SQL-User nur in einem anderen Katalog dynamische temporäre Views deklarieren. Andernfalls gibt DRIVE/WINDOWS die folgende Fehlermeldung aus:

DRI0536 42SQ6 -127 SCHEMA catalog.MODULE FUER BENUTZER berechtigung NICHT ZUGREIFBAR.

Hierbei ist *catalog* der Name der SESAM-Datenbank, für die schon dynamische temporäre Views anderer SQL-User derselben DRIVE-Sitzung existieren, MODULE ist der reservierte Name des Schemas, dem alle dynamischen temporären Views von *catalog* zugeordnet sind, und *berechtigung* ist der Name des neuen SQL-Users, der in *catalog* keine dynamischen temporären Views deklarieren kann, solange das Schema MODULE nicht leer ist (siehe DROP TEMPORARY VIEW-Anweisung).

Bei der Ausführung der Anweisung, die sich auf den temporären View bezieht, muß der aktuelle Berechtigungsschlüssel das SELECT-Privileg für die in *abfrageausdruck* verwendeten Tabellen besitzen, sofern diese Tabellen einem anderen Schemaeigentümer gehören.

Der Gültigkeitsbereich eines temporären Views wird aufgehoben bei

– Programmende (Ende der Lebensdauer erst bei Anwendungsende)

– Programmabbruch

– DRIVE-Ende (STOP)

– DROP TEMPORARY VIEW *view* bzw. DROP TEMPORARY VIEWS (nur im Dialog-Modus oder innerhalb von EXECUTE, falls der View auch mit EXECUTE deklariert wurde).

```
CREATE TEMPORARY VIEW temp_viewname [ ( { spalte },... ) ]
  AS abfrageausdruck
```

---

temp_viewname

Name des neuen temporären Views. Der Name eines statischen temporären Views darf nicht länger als 24 Zeichen sein. Der Name eines dynamischen temporären Programm-Views oder eines temporären Dialog-Views darf nicht länger als 31 Zeichen sein.

Der Name des temporären Views muß innerhalb der aktuellen Übersetzungseinheit eindeutig sein.

temp_viewname darf nicht „PLAM_DIRECTORY“ heißen.

(*spalte*,...)

Name der Spalte im View. Die Viewspalten müssen nur dann benannt werden, wenn die Spalten der zugrundeliegenden Tabellen nicht eindeutig benannt sind oder wenn Ergebnisspalten ohne Namen vorkommen.

(*spalte*,...) nicht angegeben:
Es gelten die Spaltennamen des Abfrageausdrucks.

AS *abfrageausdruck*

Abfrageausdruck, der aus Basistabellen die Spalten und Sätze auswählt, die den neuen View bilden sollen. Die Spalten des View besitzen denselben Datentyp wie die zugrundeliegenden Spalten aus dem Abfrageausdruck. *abfrageausdruck* darf keine temporären oder persistenten Views referenzieren.

Der Abfrageausdruck darf nicht *variable* enthalten. Werden im View Spalten benannt, so muß die Anzahl der Spalten in der Ergebnistabelle des Abfrageausdrucks mit der Anzahl der benannten Spalten übereinstimmen.

Änderbarer temporärer View

Ein temporärer View ist änderbar (mit INSERT, UPDATE und DELETE), wenn der zugrundeliegende Abfrageausdruck änderbar ist (siehe Metavariable *abfrageausdruck*).

Privilegien für den temporären View

Der aktuelle Berechtigungsschlüssel erhält das SELECT-Privileg für den temporären View. Wenn der View änderbar ist, erhält der aktuelle Berechtigungsschlüssel die Privilegien INSERT, UPDATE und DELETE, falls er diese Privilegien für die zugrundeliegende Basistabelle besitzt.

Alle Zugriffe auf temporäre Views, die sich auf Tabellen einer Datenbank beziehen, müssen mit dem gleichen aktuellen Berechtigungsschlüssel erfolgen.

Einschränkungen für temporäre Views

Neu geschriebene SQL-Anwendungen sollten Views anstelle von temporären Views verwenden, da temporäre Views folgende Nachteile gegenüber Views besitzen:

– Temporäre Views können sich nur auf Basistabellen, nicht auf Views beziehen.

– Temporäre Views gelten nur innerhalb einer Übersetzungseinheit.

– Temporäre Views können nur mit dem aktuellen Berechtigungsschlüssel verwendet werden (keine GRANT-Anweisung für temporäre Views).

– In Multi-SQL-User-Programmen ist die Verwendung von dynamischen temporären Views stark eingeschränkt (siehe oben).

– Das INFORMATION_SCHEMA enthält keine Beschreibung der temporären Views.

Beispiel

Das Beispiel definiert einen temporären View, der die fertigen Aufträge der Basistabelle auftrag enthält.

```
CREATE TEMPORARY VIEW fertig
    AS SELECT * FROM auftrag
        WHERE fertigist IS NOT NULL;
```

CREATE VIEW - View erzeugen

CREATE VIEW erzeugt einen (persistenten) View. Ein View ist eine Tabelle, die durch einen Abfrageausdruck definiert ist, der erst bei der Verwendung des View ausgewertet wird. Persistente Views sind in der Datenbank gespeichert und gehören zu den Metadaten der Datenbank. CREATE VIEW ist eine ausführbare Anweisung, die also im Ausführungsteil von DRIVE-Programmen stehen muß.

Der aktuelle Berechtigungsschlüssel muß Eigentümer des Schemas sein, für das der View erzeugt wird, und muß das SELECT-Privileg für die verwendeten Tabellen besitzen, sofern er nicht Eigentümer der selektierten Tabellen ist.

CREATE VIEW tabelle [(spalte,...)]

AS abfrageausdruck

[WITH CHECK OPTION]

---

tabelle
Name des neuen (persistenten) Views. Der einfache Viewname muß innerhalb der Basistabellen- und Viewnamen des Schemas eindeutig sein. Der einfache Viewname kann durch einen Datenbank- und Schemanamen qualifiziert werden (siehe Metavariable *tabellenangabe*).

(*spalte*,...)
Name der Spalte im View. Die Viewspalten müssen nur dann benannt werden, wenn die Spalten der zugrundeliegenden Tabellen nicht eindeutig benannt sind oder wenn Ergebnisspalten ohne Namen vorkommen.

(*spalte*,...) nicht angegeben:
Es gelten die Spaltennamen des Abfrageausdrucks.

AS *abfrageausdruck*
Abfrageausdruck, der aus bereits bestehenden Basistabellen und persistenten Views die Spalten und Sätze auswählt, die den neuen View bilden sollen (siehe Metavariablen *abfrageausdruck* und *sqlausdruck*). Die Spalten des View besitzen denselben Datentyp wie die zugrundeliegenden Spalten aus dem Abfrageausdruck.

Die Tabellen, die im Abfrageausdruck genannt werden, müssen zur gleichen Datenbank gehören wie der View. Der Abfrageausdruck darf nicht *variable* enthalten. Werden im View Spalten benannt, so muß die Anzahl der Spalten in der Ergebnistabelle des Abfrageausdrucks mit der Anzahl der benannten Spalten übereinstimmen.

WITH CHECK OPTION

Sätze, die Sie über den View eingeben oder ändern, werden auf die Einhaltung der im Abfrageausdruck definierten Bedingung überprüft. Sätze, die die Bedingung nicht erfüllen, werden abgewiesen. Der View muß änderbar sein.

Der Abfrageausdruck darf multiple Spalten nur in der SELECT-Klausel, nicht in der WHERE-Klausel enthalten.

WITH CHECK OPTION nicht angegeben:
Falls der View änderbar ist, können Sätze in den View eingefügt oder geändert werden, die die Bedingung im Abfrageausdruck nicht erfüllen. Solche Sätze sind anschließend nicht über den View zugreifbar.

Änderbarer View

Ein View ist änderbar (mit INSERT, UPDATE und DELETE), wenn der zugrundeliegende Abfrageausdruck änderbar ist (siehe Metavariable *abfrageausdruck*).

Privilegien für den View

Der aktuelle Berechtigungsschlüssel erhält das SELECT-Privileg für den View. Die GRANT-Berechtigung zur Weitergabe dieses Privilegs erhält er nur, wenn er die GRANT-Berechtigung für das SELECT-Privileg aller verwendeten Tabellen besitzt.

Wenn der View änderbar ist, erhält der aktuelle Berechtigungsschlüssel die Privilegien INSERT, UPDATE und DELETE, falls er diese Privilegien für die zugrundeliegenden Basistabellen besitzt. Die GRANT-Berechtigung zur Weitergabe dieser Privilegien erhält er nur, wenn er die GRANT-Berechtigung für die entsprechenden Privilegien der zugrundeliegenden Basistabellen besitzt.

Beispiel

Das Beispiel definiert einen View der die fertigen Aufträge der Basistabelle auftrag enthält.

```
CREATE VIEW fertig
    AS SELECT * FROM auftrag
        WHERE fertigist IS NOT NULL;
```

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 *cursorbeschreibung* angegeben werden.

Bei DRIVE/WINDOWS 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 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 ] [ PREFETCH n ] CURSOR [ FOR cursorbeschreibung ]


    cursorbeschreibung::=

        abfrageausdruck

        [ ORDER BY { { spalte | spalte(posnr) | spaltennummer }
        [ { ASCENDING | DESCENDING } ] },... ]

        [ FOR UPDATE [ OF { spalte },... ] ]


    n::= vorzeichenlose_ganzzahl

    posnr::= vorzeichenlose_ganzzahl

    spaltennummer::= vorzeichenlose_ganzzahl
```

---

cursor

Name für den Cursor. Der Name eines statischen Cursors darf nicht länger als 11 Zeichen sein. Der Name eines dynamischen oder variablen Programm-Cursors oder eines Dialog-Cursors darf nicht länger als 18 Zeichen sein. Innerhalb einer Übersetzungseinheit dürfen nicht mehrere Cursor mit demselben Namen vereinbart werden. Auf einer Programmebene bzw. im Dialog-Modus können zu einem Zeitpunkt keine zwei Cursor mit identischen Namen deklariert sein. 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.

SCROLL nicht angegeben:
Der Cursor kann nur auf den jeweils nächsten Satz positioniert werden. Bei FETCH ist nur die Positionsangabe NEXT erlaubt.

PREFETCH

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 „Pragmas“ auf Seite 62). 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-Fehlern (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 sogenannter **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 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

Abfrageausdruck zur Auswahl von Sätzen und Spalten aus Basistabellen oder Views.

Der Wert für Variablen 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-Klausel

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 (siehe Metavariable *praedikat*), 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 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 und darf keine Bereichsangabe enthalten.

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

ASCENDING ist Voreinstellung.
Die Werte der zugehörigen Spalte werden aufsteigend sortiert.

DESCENDING

Die Werte der zugehörigen Spalte werden absteigend sortiert.

FOR UPDATE

Für statischen oder dynamischen Cursor.

Die FOR UPDATE-Klausel darf nur bei einem änderbaren Cursor angegeben werden (siehe unten). Insbesondere schließen sich die FOR UPDATE- und die ORDER BY-Klausel gegenseitig aus. Mit der FOR UPDATE-Klausel können Sie festlegen, welche Spalten der zugrundeliegenden Tabelle über den Cursor mit UPDATE...WHERE CURRENT OF geändert werden dürfen.

FOR UPDATE dürfen Sie nur angeben, wenn für die Cursorbeschreibung von *cursor* keine SCROLL-Klausel vereinbart wird.

Bei PREFETCH-Cursorn ist FOR UPDATE nicht erlaubt.

FOR UPDATE nicht angegeben:
Ist der Cursor änderbar, können alle Spalten der zugrundeliegenden 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 zugrundeliegenden Tabelle, auch wenn im Abfrageausdruck der Cursorbeschreibung neue Spaltennamen definiert werden.

OF *spalte*,... nicht angegeben:

Jede Spalte der zugrundeliegenden 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 zugrundeliegende 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

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/WINDOWS einen Teil der Blanks für die Einrückung 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

Es wird ein änderbarer Cursor cur vereinbart. Die zugrundeliegende 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

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.

DELETE - Sätze löschen

DELETE löscht Sätze aus einer Tabelle.

Um einen Satz in der angegebenen Tabelle zu löschen, müssen Sie Eigentümer dieser Tabelle sein oder das DELETE-Privileg für diese Tabelle besitzen. Zusätzlich muß der Transaktionsmodus der aktuellen Transaktion READ WRITE sein.

Sind für die Tabelle bzw. die betroffenen Spalten Integritätsbedingungen definiert, werden diese nach dem Löschen geprüft. Ist eine Integritätsbedingung verletzt, werden die Löschungen rückgängig gemacht und ein entsprechender SQLSTATE gesetzt.

DELETE FROM tabelle [WHERE { bedingung | CURRENT OF cursor }]

---

tabelle
: Name der Tabelle, aus der Sätze gelöscht werden sollen. Die Tabelle kann eine Basistabelle, ein änderbarer View oder ein änderbarer temporärer View sein (siehe Metavariable *tabellenangabe*). Bei einem dynamischen Cursor darf kein statischer temporärer View angegeben sein.

WHERE-Klausel
: Die WHERE-Klausel gibt an, welche Sätze gelöscht werden.

  WHERE *bedingung* löscht eine Satzmenge (multiple DELETE-Anweisung), WHERE CURRENT OF *cursor* löscht einen einzelnen Satz.

  WHERE-Klausel nicht angegeben:
  Alle Sätze der Tabelle werden gelöscht.

bedingung
: Bedingung, die die zu löschenden Sätze erfüllen müssen. Ein Satz wird nur gelöscht, wenn er die angegebene *bedingung* erfüllt.

  Spaltenangaben in *bedingung* außerhalb von Unterabfragen dürfen sich nur auf die angegebene Tabelle *tabelle* beziehen.

  Unterabfragen in *bedingung* dürfen sich nicht direkt oder indirekt auf die Basistabelle beziehen, aus der Sätze gelöscht werden sollen.

CURRENT OF *cursor*
: Name des Cursors, über den die zu löschenden Sätze ausgewählt werden. Der Cursor muß änderbar sein (siehe Anweisung DECLARE), und *tabelle* muß die zugrundeliegende Tabelle sein.

Der Cursor muß in derselben Übersetzungseinheit vereinbart sein und vor der DELETE-Anweisung mit OPEN oder RESTORE geöffnet sowie mit FETCH auf einen Satz der Ergebnistabelle positioniert worden sein.

DELETE löscht den Satz in *tabelle*, der sich aus der aktuellen Cursorposition ergibt.

Nach DELETE zeigt der Cursor vor den nachfolgenden Satz der Ergebnistabelle bzw. hinter den letzten, wenn es keinen weiteren Satz gibt. Für eine weitere DELETE...WHERE CURRENT OF-Anweisung müssen Sie den Cursor zuerst wieder mit FETCH auf einen Satz der Ergebnistabelle positionieren.

DELETE ist nicht erlaubt, wenn *cursor* ein PREFETCH-Cursor ist.

DELETE und Transaktionssicherung

SQL erlaubt das Löschen von Sätzen nur innerhalb von Transaktionen. Durch die Definition eines Isolationslevels mit SET TRANSACTION bei konkurrierenden Transaktionen können Sie steuern, welche Auswirkungen der Löschvorgang auf diese Transaktionen hat. Tritt während der Ausführung der DELETE-Anweisung ein Fehler auf, werden sämtliche bereits durchgeführten Löschungen rückgängig gemacht.

Beispiele

1. Alle in Hannover ansässigen Kunden werden aus der Tabelle kunde gelöscht.

```
DELETE FROM kunde WHERE ort = 'Hannover';
```

2. In diesem Beispiel wird ein Cursor verwendet, um die Kunden aus Hannover aus der Tabelle kunde zu löschen.

```
DECLARE cur_kunden CURSOR FOR
  SELECT knr, firma, ort FROM kunde
    WHERE ort = 'Hannover'
  FOR UPDATE;

OPEN cur_kunden;
```

Mit einer Folge von FETCH- und DELETE-Anweisungen können nun alle gefundenen Sätze gelöscht werden.

```
FETCH cur_kunden INTO &Knr, &Firma, &Ort;

DELETE FROM kunde WHERE CURRENT OF cur_kunden;
```

DROP CURSOR - Cursor Deklaration freigeben

DROP CURSOR bzw. DROP CURSORS gibt im Dialog-Modus von DRIVE oder innerhalb von EXECUTE (nur, wenn der Cursor auch innerhalb von EXECUTE deklariert wurde) Cursor frei. Außerdem ist die Anweisung DROP CURSOR im Programm-Modus für variable Cursor erlaubt (siehe DECLARE-Anweisung).

DROP CURSOR(S) unterliegt der Transaktionssicherung, d.h. wenn Sie ROLLBACK WORK angeben, werden die bei DROP CURSOR(S) angegebenen Cursor nicht freigegeben.

Innerhalb einer Transaktion kann DECLARE... CURSOR mehrfach für denselben Cursor verwendet werden, wenn vor einer wiederholten Verwendung ein DROP CURSOR durchlaufen worden ist. Kommen in einer Transaktion DROP CURSOR- und DECLARE... CURSOR-Anweisungen für denselben Cursor vor, so darf die erste dieser Anweisungen keine DROP CURSOR-Anweisung sein.

Ein variabler Cursor kann nur explizit (statisch oder dynamisch) mit DROP CURSOR *cursor* gelöscht werden. Beim DROP auf einen variablen Cursor wird die variable Cursorbeschreibung (FOR-Klausel) gelöscht. D.h. Sie können die Cursorbeschreibung des Cursors nach einer DROP-Anweisung erneut nachdeklarieren. Zwischen DROP und Nachdeklaration sollte ein COMMIT WORK stehen.

DROP { CURSOR cursor | CURSORS }

---

cursor
: Der angegebene Cursor *cursor* wird freigegeben. Ist der Cursor geöffnet, wird implizit ein CLOSE ausgeführt.

CURSORS
: Alle im Dialog-Modus definierten Cursor bzw. alle auf derselben Programmebene mit EXECUTE definierten nicht-variablen Cursor werden freigegeben.

  Die Anweisung DROP CURSORS ist äquivalent zu einer Reihe von DROP CURSOR-Anweisungen, in der alle Cursornamen explizit angegeben werden.

Beispiel

```
DECLARE c1 CURSOR;
EXECUTE 'DECLARE c1 CURSOR FOR '|| &SUCH1;
...
DROP CURSOR c1;
COMMIT WORK;
EXECUTE 'DECLARE c1 CURSOR FOR ' || &SUCH2;
```

DROP SCHEMA - Schema löschen

DROP SCHEMA löscht ein leeres Datenbankschema. Alle Basistabellen, Integritätsbedingungen und Views des Schemas müssen zuvor gelöscht werden.

Welche Schemas definiert sind, erfahren Sie im View SCHEMATA des INFORMATION_SCHEMA (siehe SESAM/SQL-Server SQL-Sprachbeschreibung Teil1 [18], Kapitel „Informationsschemata“).

Der aktuelle Berechtigungsschlüssel muß Eigentümer des Schemas sein.

DROP SCHEMA [catalog .] schema RESTRICT

---

schema

Name des Schemas. Das Schema muß leer sein.

Der einfache Name des Schemas kann durch einen Datenbanknamen qualifiziert werden.

DROP TABLE - Basistabelle löschen

DROP TABLE löscht eine Basistabelle und die zugehörigen Indizes, sofern in der Tabelle keine Datensätze existieren. Eine Basistabelle kann nicht gelöscht werden, wenn sie in einem View oder in einer Integritätsbedingung verwendet wird. Es werden alle temporären Views gelöscht, die auf der Basistabelle definiert sind.

Welche Basistabellen definiert sind, erfahren Sie im View BASE_TABLES des INFORMATION_SCHEMA (siehe SESAM/SQL-Server SQL-Sprachbeschreibung Teil1 [18], Kapitel „Informationsschemata“).

Der aktuelle Berechtigungsschlüssel muß Eigentümer des Schemas sein, zu dem die Tabelle gehört.

Diese Anweisung kann deklarative Anweisungen eines DRIVE-Programms zerstören.

DROP TABLE tabelle RESTRICT

---

tabelle

Name der Basistabelle, die gelöscht werden soll. Die Tabelle muß leer sein. Der entfernte Tabellenname kann durch einen Datenbank- und Schemanamen qualifiziert werden.

DROP TEMPORARY VIEW - Temporären View löschen

DROP TEMPORARY VIEW löscht die Definition eines temporären View. Mit dem View werden implizit auch alle diesen View referenzierenden dynamischen Cursor gelöscht.

Diese Anweisung ist statisch nur im Dialog-Modus erlaubt. Als dynamische Anweisung im Programm-Modus muß sie sich auf einen dynamisch (mit EXECUTE) definierten View des jeweiligen DRIVE-Programms beziehen.

Der aktuelle Berechtigungsschlüssel muß Eigentümer des temporären Views sein.

```
DROP TEMPORARY { VIEW tabelle | VIEWS }
```

---

tabelle
Name des temporären Views, der gelöscht werden soll.

VIEWS
Alle im Dialog-Modus definierten Views bzw. alle auf derselben Programmebene mit EXECUTE definierten Views und die sie referenzierenden Cursor werden freigegeben.

Die Anweisung DROP TEMPORARY VIEWS ist äquivalent zu einer Reihe von DROP TEMPORARY VIEW-Anweisungen, in der alle Viewnamen explizit angegeben werden. DROP TEMPORARY VIEWS wird daher nur ausgeführt, wenn der aktuelle Berechtigungsschlüssel Eigentümer aller Views ist.

DROP VIEW - View löschen

DROP VIEW löscht die Definition eines (persistenten) View. Ein View kann nicht gelöscht werden, wenn er in einer anderen Viewdefinition verwendet wird.

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 SESAM/SQL-Server SQL-Sprachbeschreibung Teil1 [18], Kapitel „Informationsschemata“).

Der aktuelle Berechtigungsschlüssel muß Eigentümer des Schemas sein, zu dem der View gehört.

```
DROP VIEW tabelle RESTRICT
```

---

tabelle
Name des View, der gelöscht werden soll.

FETCH - Cursor positionieren und Satz lesen

FETCH positioniert einen Cursor. Die neue Cursorposition verweist entweder auf einen Satz, vor den ersten Satz oder hinter den letzten Satz der Cursortabelle. Ist die neue Cursorposition auf einem Satz der Cursortabelle, wird dieser Satz zum aktuellen Satz, und die Spaltenwerte dieses Satzes werden gelesen.

Im Programm-Modus werden diese Spaltenwerte in die angegebenen Variablen übertragen (vgl. INTO-Klausel), im Dialog-Modus werden sie am Bildschirm angezeigt (vgl. DRIVE-Programmiersystem [1], Kapitel „Datenzugriffe im Dialog“).

Wird bei FETCH kein Satz gelesen, weil die angegebene Position nicht existiert, wird ein entsprechender SQLSTATE gesetzt, der mit WHENEVER &DML_STATE IN ('TABLE END') behandelt werden kann.

Ein mit SCROLL vereinbarter Cursor kann mit FETCH in beliebiger Reihenfolge auf jeden Satz der Cursortabelle positioniert werden. Ein ohne SCROLL vereinbarter Cursor kann nur auf den jeweils nächsten Satz positioniert werden (FETCH NEXT...).

Die Cursorvereinbarung mit DECLARE muß in derselben Übersetzungseinheit stehen.

Der Cursor muß geöffnet oder wiederhergestellt sein (siehe Anweisungen OPEN und RESTORE).

Zum Zeitpunkt des FETCH darf kein mit einer STORE-Anweisung erstellter Sicherungsstand des Cursors existieren.

Ist für den Cursor der Schubmodus vereinbart (siehe DECLARE ... CURSOR FOR ... und PRAGMA), können Sie nur noch mit FETCH NEXT positionieren. Wurde für einen statischen PREFETCH-Cursor bereits eine (statische) FETCH [NEXT]-Anweisung ausgeführt, so ist nachfolgend bis zum nächsten CLOSE oder COMMIT WORK nur noch genau diese eine FETCH-Anweisung erlaubt, d.h. dieselbe Anweisung in einer Schleife (siehe Anweisung CYCLE im DRIVE-Lexikon [3]) oder in einem internen Unterprogramm (siehe Anweisung SUBPROCEDURE im DRIVE-Lexikon [3]).

```
FETCH [ { NEXT | PRIOR | FIRST | LAST | RELATIVE n | ABSOLUTE n  } ]
      [ FROM ] cursor

      [ INTO  { variable },... ]

      n::=  { [ { + | - } ] vorzeichenlose_ganzzahl | variable }
```

NEXT

NEXT ist Voreinstellung.
Positioniert den Cursor auf den nächsten Satz der Cursortabelle. Bei einem ohne SCROLL oder mit PREFETCH vereinbarten Cursor können Sie nur die NEXT-Klausel verwenden.

Steht der Cursor auf dem letzten Satz der Cursortabelle, wird er hinter den letzten Satz positioniert; steht er bereits hinter dem letzten Satz, bleibt die Position unverändert.

PRIOR

Positioniert den Cursor auf den vorhergehenden Satz der Cursortabelle.

Steht der Cursor auf dem ersten Satz der Cursortabelle, wird er vor den ersten Satz positioniert; steht er bereits vor dem ersten Satz, bleibt die Position unverändert.

PRIOR ist nur zulässig, wenn Sie den Cursor mit SCROLL vereinbart haben.

FIRST

Positioniert den Cursor auf den ersten Satz der Cursortabelle bzw. vor den ersten Satz, wenn die Cursortabelle leer ist.

FIRST ist nur zulässig, wenn Sie den Cursor mit SCROLL vereinbart haben.

LAST

Positioniert den Cursor auf den letzten Satz der Cursortabelle bzw. hinter den letzten Satz, wenn die Cursortabelle leer ist.

LAST ist nur zulässig, wenn Sie den Cursor mit SCROLL vereinbart haben.

ABSOLUTE *n*

Position für den Cursor angeben. ABSOLUTE ist nur zulässig, wenn Sie den Cursor mit SCROLL vereinbart haben.

Für *n* können Sie eine Ganzzahl oder eine Variable vom DRIVE-Datentyp INTEGER oder SMALLINT angeben. Die Cursorposition wird abhängig vom Wert für *n* wie folgt bestimmt:

>0 Der Cursor wird auf den *n*-ten Satz der Cursortabelle positioniert bzw. hinter den letzten Satz, wenn *n* > Anzahl der Sätze der Cursortabelle.

0 Der Cursor wird vor den ersten Satz der Cursortabelle positioniert.

<0 Der Cursor wird auf den (N+1-|n|)-ten Satz der Cursortabelle positioniert, wobei N die Anzahl der Sätze der Cursortabelle ist. Ist |n| > N, wird der Cursor vor den ersten Satz positioniert.

Beispiel:

FETCH ABSOLUTE -1 und FETCH LAST sind äquivalent.

RELATIVE *n*

Position für den Cursor relativ zur aktuellen Position angeben. RELATIVE ist nur zulässig, wenn Sie den Cursor mit SCROLL vereinbart haben.

Für *n* können Sie ein ganzzahliges Literal oder eine Variable vom Typ INT oder SMALLINT angeben. Die Cursorposition wird abhängig vom Wert für *n* wie folgt bestimmt:

>0 Der Cursor wird auf den Satz positioniert, der *n* Sätze hinter seiner aktuellen Position liegt. Ist die neue Position größer als die Anzahl der Sätze der Cursortabelle, wird der Cursor hinter den letzten Satz positioniert.

0 Die Cursorposition bleibt unverändert.

<0 Der Cursor wird auf den Satz positioniert, der *n* Sätze vor seiner aktuellen Position liegt. Wird die neue Position ≤ 1, wird der Cursor vor den ersten Satz positioniert.

[FROM] *cursor*

Name des Cursors.

INTO-Klausel

Angabe, wohin die gelesenen Werte gespeichert werden. Im Programm-Modus muß die Klausel angegeben werden. Im Dialog-Modus ist die Klausel nicht erlaubt.

variable

Name einer Variable, der ein Spaltenwert des Ergebnissatzes zugewiesen wird.

Der Datentyp einer Variable muß mit dem Datentyp des zugehörigen Ausgabewerts verträglich sein. Ist ein Ausgabewert ein Aggregat mit mehreren Elementen, muß die zugehörige Variable ein Vektor mit derselben Anzahl von Elementen sein.

Die Anzahl der angegebenen Variablen muß mit der Anzahl der Spalten in der SELECT-Liste der Cursorbeschreibung übereinstimmen. Andernfalls gibt DRIVE/WINDOWS die Fehlermeldung DRI0504 aus. Der Wert der i-ten Spalte in der SELECT-Liste, der ggf. der NULL-Wert ist, wird der i-ten Variable in der INTO-Klausel zugewiesen.

Verhalten von SESAM/SQL im Fehlerfall

Bei Fehlern beim Lesen der Werte (z.B. numerischer Wert für Zieldatentyp zu groß) hat der Cursor die neue Position, aber die zugewiesenen Werte sind undefiniert.

Bei anderen Fehlern (z.B. Datentypen nicht verträglich) bleibt die Position unverändert und es werden keine Werte gelesen.

Beispiele

Beispiel für FETCH NEXT im Programm-Modus:

```
FETCH cur_auftrag
  INTO &ANR,
       &ADATUM,
       &ATEXT,
       &ASTAT;
```

Beispiel für FETCH mit einem statischen PREFETCH-Cursor:

```
DCL cur_auftrag CURSOR PREFETCH...
            FOR SELECT anr, adatum, atext, astat
   FROM auftrag
   WHERE knr >= &KUNDENNR;

...

CYCLE cur_auftrag INTO &ANR,
                       &ADATUM,
                       &ATEXT,
                       &ASTAT;

...

END CYCLE;
```

GRANT - Privilegien vergeben

GRANT vergibt Tabellen- und Spalten-Privilegien für Basistabellen und Views (DML-Rechte und Referenzrechte) und Sonder-Privilegien für Datenbanken. Ist die persistente GRANT-Anweisung in einer CREATE SCHEMA-Anweisung enthalten, darf die GRANT-Anweisung keine Sonder-Privilegien vergeben. Die Privilegien können durch Berechtigungsschlüssel spezifizierten SQL-Usern oder der Allgemeinheit gewährt werden.

Der aktuelle Berechtigungsschlüssel muß die angegebenen Privilegien vergeben dürfen:

– Er ist der Berechtigungsschlüssel des universellen Benutzers, d.h. Eigentümer der Datenbank.

– Er ist Eigentümer des Schemas, zu dem die Tabelle gehört.

– Er besitzt die GRANT-Berechtigung zur Weitergabe der Privilegien.

Welcher Berechtigungsschlüssel Schema-Eigentümer ist, ist im View SCHEMATA abgelegt. Ob der Berechtigungsschlüssel die GRANT-Berechtigung für ein Privileg besitzt, erfahren Sie aus den Views TABLE_PRIVILEGES, COLUMN_PRIVILEGES und CATALOG_PRIVILEGES (siehe SESAM/SQL-Server SQL-Sprachbeschreibung Teil1 [18, Kapitel „Informationsschemata“).

Nur der Berechtigungsschlüssel, der die Privilegien vergeben hat, kann die Privilegien wieder entziehen.

DRIVE/WINDOWS unterstützt die GRANT-Anweisung von SESAM/SQL eingeschränkt:

– alle Tabellen- und Spaltenprivilegien, gemeinsam oder einzeln

– alle Sonder-Privilegien für Datenbanken gemeinsam

– das Sonder-Privileg CREATE SCHEMA

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 | { tabellen_und_spalten_privileg },... }

ON [ TABLE ] tabelle

TO { PUBLIC | { berechtigungsschlüssel },... }

[ WITH GRANT OPTION ]
tabellen_und_spalten_privileg::= { SELECT | DELETE | INSERT |
                        UPDATE [ ( { spalte },... ) ] |
                        REFERENCES [ ( { spalte },... ) ]
```

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.

tabellen_und_spalten_privileg
: Die Tabellen- und Spalten-Privilegien werden einzeln vergeben. Sie können mehrere 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.

    Das Ändern kann auf die angegebenen Spalten eingeschränkt werden. *spalte* muß ein einfacher 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 (siehe Anweisungen CREATE TABLE und ALTER TABLE).
Die Referenzierung kann auf die angegebenen Spalten eingeschränkt werden. *spalte* muß ein einfacher 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. Der Berechtigungsschlüssel darf max. 18 Zeichen lang sein.

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. Nach Weitervergabe können Privilegien nicht mehr entzogen werden. 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 SCHEMA }

ON CATALOG catalog

TO { PUBLIC | { berechtigungsschlüssel },... }

[ WITH GRANT OPTION ]
```

ALL SPECIAL PRIVILEGES
: Alle Sonder-Privilegien werden vergeben, die der aktuelle Berechtigungsschlüssel für die in der ON-Klausel angegebene Datenbank vergeben darf. ALL SPECIAL PRIVILEGES umfaßt die Sonder-Privilegien CREATE SCHEMA (siehe unten) sowie CREATE USER, CREATE STOGROUP und UTILITY (siehe SESAM/SQL-Server SQL-Sprachbeschreibung Teil1 [18] und Teil 2 [19]).

CREATE SCHEMA
: Sonder-Privileg, das das Definieren von Datenbank-Schemas erlaubt.

ON CATALOG *catalog*
: Name der Datenbank, für die Sie Sonder-Privilegien vergeben wollen.

TO PUBLIC

: Die Sonder-Privilegien werden der Allgemeinheit verliehen. Jeder Berechtigungsschlüssel besitzt zusätzlich zu seinem eigenen die Sonder-Privilegien, die an PUBLIC verliehen wurden. Auch später hinzugefügte Berechtigungsschlüssel besitzen diese Sonder-Privilegien.

TO {*berechtigungsschlüssel*},...

: Die Sonder-Privilegien gelten für den Berechtigungsschlüssel *berechtigungsschlüssel*. Sie können andere Berechtigungschlüssel angeben. Der Berechtigungsschlüssel darf max. 18 Zeichen lang sein.

WITH GRANT OPTION

: Die angegebenen Berechtigungsschlüssel erhalten zusätzlich zu den Sonder-Privilegien die GRANT-Berechtigung, d.h. die Berechtigungsschlüssel sind berechtigt, die erhaltenen Sonder-Privilegien an andere Berechtigungsschlüsssel weiterzugeben. Nach Weitergabe können Sonder-Privilegien nicht mehr entzogen werden. Die Klausel WITH GRANT OPTION dürfen Sie nicht in Verbindung mit PUBLIC verwenden.

WITH GRANT OPTION nicht angegeben:

Die angegebenen Berechtigungsschlüssel können die verliehenen Sonder-Privilegien nicht weitergeben.

Beispiel

Die erste GRANT-Anweisung vergibt mehrere Tabellen-Privilegien, die zweite vergibt das Sonder-Privileg CREATE SCHEMA an einen bereits bestehenden Berechtigungsschlüssel.

```
GRANT SELECT,INSERT,UPDATE ON TABLE telefonliste TO berta;

GRANT CREATE SCHEMA ON CATALOG meinedb TO hugo;
```

INSERT - Sätze in Tabelle einfügen

INSERT fügt in eine bestehende Tabelle einen oder mehrere Sätze ein.

Um einen Satz in die angegebenen Tabelle einzufügen, müssen Sie Eigentümer dieser Tabelle sein oder das INSERT-Privileg für diese Tabelle besitzen. Zusätzlich muß der Transaktionsmodus der aktuellen Transaktion READ WRITE sein.

Die in der INSERT-Anweisung (und in Voreinstellungen) vorkommenden Literale CURRENT USER bzw. USER und SYSTEM USER sowie die Zeitfunktionen CURRENT DATE, CURRENT TIME und CURRENT TIMESTAMP werden einmal ausgewertet. Die berechneten Werte gelten für alle Einfügungen.

Sind für die Tabelle bzw. die betroffenen Spalten Integritätsbedingungen definiert, werden diese nach dem Einfügen geprüft. Ist eine Integritätsbedingung verletzt, werden die Einfügungen rückgängig gemacht und ein entsprechender SQLSTATE gesetzt.

```
INSERT INTO tabelle

  { [ ( { spalte | spalte(posnr) | spalte(min-max) },... ) ]

    { VALUES ( { sqlausdruck | DEFAULT | NULL | * },... ) |
      VALUES { sqlausdruck | DEFAULT | NULL | * } |
      abfrageausdruck } |

    DEFAULT VALUES }


      [ RETURN INTO variable ]
```

---

tabelle
: Name der Tabelle, in die Sätze eingefügt werden sollen. Die Tabelle kann eine Basistabelle, ein änderbarer View oder ein änderbarer temporärer View sein (siehe Metavariable *tabellenangabe*).

spalte
: Name einer einfachen Spalte, in die ein Wert eingefügt werden soll.

  Für *spalte* geben Sie eine Spalte der angegebenen Tabelle an. Der Spaltenname darf nicht mit einer Tabellenangabe qualifiziert werden. Die Reihenfolge der Spalten muß nicht mit der Reihenfolge in der Tabelle übereinstimmen.
  Innerhalb der Spaltenliste darf eine einfache Spalte nicht doppelt vorkommen.

spalte(*posnr*)
Element einer multiplen Spalte, in das ein Wert eingefügt werden soll.

Die multiple Spalte muß in der Tabelle enthalten sein. Werden andere Elemente einer multiplen Spalte angegeben, muß der Teilbereich aus den angegebenen Spaltenelementen lückenlos sein. Jedes Element darf genau einmal vorkommen.
Für *posnr* geben Sie eine vorzeichenlose Ganzzahl ≥ 1 an.

spalte(*min-max*)
Teilbereich von Spaltenelementen einer multiplen Spalte, die mit Werten belegt werden sollen. Die multiple Spalte muß in der Tabelle enthalten sein. Die angegebenen Teilbereiche müssen so gewählt werden, daß jedes Spaltenelement mit Positionsnummer zwischen 1 und der größten angegebenen Positionsnummer genau einmal vorkommt.

Für *min* und *max* geben Sie vorzeichenlose Ganzzahlen ≥ 1 an; *max* muß ≥ *min* sein.

Keine Spaltenangabe:
Die folgenden Angaben geben Werte für alle Spalten der Tabelle *tabelle* an, wobei die Spaltenreihenfolge gilt, die bei CREATE TABLE, ALTER TABLE bzw. CREATE VIEW oder CREATE TEMPORARY VIEW angegeben wurde.

VALUES-Klausel
Werte für die zuvor angegebenen oder für alle Spalten einzeln angeben. Wenn nur ein Wert eingetragen wird, können die Klammern weggelassen werden.

Mit dieser Form der INSERT-Anweisung kann genau ein Satz in die Tabelle *tabelle* eingefügt werden.

Ist eine Spaltenauswahl angegeben, muß für jede Angabe in der Spaltenliste ein entsprechender Wert in der VALUES-Klausel existieren. Nicht angegebene Spalten werden auf den voreingestellten Wert gesetzt, wenn eine Voreinstellung definiert ist, ansonsten auf den NULL-Wert.

Ist keine Spaltenauswahl angegeben, muß für jede Spalte der Tabelle ein entsprechender Wert angegeben werden.

Für die Zuweisung der Werte gelten die Zuweisungsregeln aus der SESAM/SQL-Server SQL-Sprachbeschreibung Teil 1 [18], Abschnitt „Werte in Tabellenspalten eintragen“.

Der *i*-te Wert der VALUES-Liste wird der *i*-ten Spalte in der Spaltenliste zugewiesen, wenn eine Spaltenliste angegeben wurde, ansonsten der *i*-ten Spalte der Tabelle.

sqlausdruck
Ausdruck, dessen Wert einer Spalte zugeordnet wird. Der Wert des Ausdrucks muß mit dem Datentyp der Spalte verträglich sein.

Ist *sqlausdruck* eine Variable, kann auch ein Vektor angegeben werden. Die Spalte muß dann eine multiple Spalte sein und die Anzahl der Elemente des Vektors muß mit der Anzahl der Spaltenelemente übereinstimmen.

Unterabfragen in *sqlausdruck* dürfen sich nicht direkt oder indirekt auf die Basistabelle beziehen, in die Sätze eingefügt werden.

Ist *sqlausdruck* ein Aggregat (siehe Metavariable *wert*), das einer multiplen Spalte zugewiesen werden soll, so muß die Anzahl der Werte mit der Anzahl der Spaltenelemente übereinstimmen, und der Datentyp jeder Komponente des Aggregats muß mit dem Datentyp der Zielspalte verträglich sein.

DEFAULT
Nur für einfache Spalte.

Zugehörige Spalte mit voreingestelltem Wert belegen. Die Voreinstellung wird bei der Definition der Spalte festgelegt. Ist keine Voreinstellung definiert, wird die Spalte mit dem NULL-Wert belegt.

NULL
Nur für einfache Spalte.

Zugehörige Spalte mit dem NULL-Wert belegen.

*
Nur für einfache Spalte.

Wertangabe für eine Spalte, für die SESAM/SQL den Wert bestimmt (Zählspalte).

Die Spalte muß einen Ganz- oder Festpunktzahltyp (SMALLINT, INT, DECIMAL, NUMERIC) haben und zu einem Primärschlüssel gehören. Die Spalte darf weder in einer Referenzbedingung noch in einer Check-Bedingung der Tabelle *tabelle* enthalten sein. SESAM/SQL belegt die Spalte so, daß der Primärschlüssel innerhalb der Tabelle eindeutig ist.

* darf nur einmal in der VALUES-Klausel vorkommen.

abfrageausdruck
Die einzufügenden Werte werden über einen Abfrageausdruck angegeben.

Mit dieser Form der INSERT-Anweisung kann eine Satzmenge in die Tabelle *tabelle* eingefügt werden (multiple INSERT-Anweisung).

abfrageausdruck ist ein Abfrageausdruck, dessen Ergebnistabelle die einzufügenden Sätze liefert. Liefert *abfrageausdruck* eine leere Tabelle, wird kein Satz eingefügt und ein entsprechender SQLSTATE gesetzt, der mit WHENEVER &DML_STATE IN ('TABLE END') behandelt werden kann.

Ist eine Spaltenauswahl angegeben, muß die Anzahl der Spalten der Ergebnistabelle mit der Anzahl der Angaben in der Spaltenliste übereinstimmen. Nicht angegebene Spalten werden auf den voreingestellten Wert gesetzt, wenn eine Voreinstellung definiert ist, ansonsten auf den NULL-Wert.

Ist keine Spaltenauswahl angegeben, muß für jede Spalte der Tabelle eine entsprechende Spalte in der Ergebnistabelle existieren.

Der Wert der *i*-ten Ergebnisspalte wird der *i*-ten Spalte in der Spaltenliste zugewiesen, wenn eine Spaltenliste angegeben wurde, ansonsten der *i*-ten Spalte der Tabelle.

Jeder Wert muß mit dem Datentyp der zugehörigen Spalte verträglich sein.

In den FROM-Klauseln des Abfrageausdruck und der im Abfrageausdruck enthaltenen Unterabfragen dürfen Sie keine Tabelle angeben, die sich auf die zugrundeliegende Basistabelle, in die die neuen Sätze eingefügt werden, bezieht. Insbesondere dürfen Sie nicht *tabelle* angeben.

DEFAULT VALUES

Einen Satz in der Tabelle *tabelle* eintragen, der nur aus den spaltenspezifischen Voreinstellungen besteht (siehe Metavariable *spaltendefinition*).

Die Spalten, für die eine Voreinstellung definiert ist, werden mit dem voreingestellten Wert belegt. Spalten, für die keine Voreinstellung definiert ist, werden mit dem NULL-Wert belegt.

RETURN INTO

Der Wert der Spalte, die durch Angabe von * in der VALUES-Klausel von SESAM/SQL belegt wird, wird in eine Variable abgespeichert.

Die RETURN INTO-Klausel ist nur erlaubt, wenn die VALUES-Klausel einen Stern * enthält.

variable

Name der Variable, in die der Wert der Zählspalte eingetragen wird. *variable* muß einen Ganz- oder Festpunktzahltyp (SMALLINT, INT, DECIMAL, NUMERIC) haben.

Werte für multiple Spalte einfügen

Bei einer multiplen Spalte können Werte für einzelne Spaltenelemente sowie für Teilbereiche eingefügt werden.

Ein Element einer multiplen Spalte wird durch seine Positionsnummer in der multiplen Spalte angesprochen.

Ein Teilbereich einer multiplen Spalte wird durch die Positionsnummern des ersten und letzten Elements des Teilbereichs angesprochen.

Die Position eines Elements innerhalb der multiplen Spalte kann sich ändern (siehe Anweisung UPDATE).

INSERT und Integritätsbedingungen

Durch Angabe von Integritätsbedingungen bei der Definition der Basistabelle können Sie den Wertebereich für die entsprechenden Spalten einschränken. Die in der INSERT-Anweisung angegebenen Werte müssen den definierten Integritätsbedingungen genügen.

INSERT und Transaktionssicherung

INSERT leitet eine Transaktion ein, wenn keine Transaktion offen ist. Durch die Definition eines Isolationslevels bei konkurrierenden Transaktionen können Sie steuern, welche Auswirkungen die INSERT-Anweisung auf diese Transaktionen hat.
Tritt während des Einfügens ein Fehler auf, werden alle bereits eingefügten Sätze wieder gelöscht.

Beispiel

In die Tabelle auftrag soll ein neuer Satz aufgenommen werden.

```
INSERT INTO auftrag (anr, knr, konr, atext)
  VALUES (500, 105, 35, 'Beratung');
```

OPEN - Cursor öffnen

OPEN öffnet einen Cursor, der mit DECLARE vereinbart wurde.

– Die in der Cursorbeschreibung vorkommenden Variablen werden ausgewertet.

– Die in der Cursorbeschreibung vorkommenden Literale CURRENT USER bzw. USER und SYSTEM USER sowie die Zeitfunktionen CURRENT DATE, CURRENT TIME und CURRENT TIMESTAMP werden ausgewertet.

  Alle zurückgelieferten Werte enthalten gleiches Datum und/oder gleiche Uhrzeit. Diese Werte gelten für die Cursortabelle, solange der Cursor offen ist, und auch, wenn der Cursor mit RESTORE wiederhergestellt wird.

Nach der OPEN-Anweisung ist der Cursor vor den ersten Satz der Ergebnistabelle positioniert, auch wenn die Cursorposition zuvor mit STORE gesichert wurde. Eine zuvor gespeicherte Cursorposition kann nach OPEN nicht mit RESTORE wiederhergestellt werden.

Ein Cursor ist nur in der Übersetzungseinheit ansprechbar, in der er mit DECLARE vereinbart wurde. Eine statische Cursorvereinbarung mit DECLARE muß im Programmtext vor der OPEN-Anweisung stehen.

Bei einem dynamischen Cursor muß die Cursorbeschreibung zum Ausführungszeitpunkt der OPEN-Anweisung deklariert sein.

Der Cursor muß geschlossen sein.

Ein geöffneter Cursor wird durch eine der folgenden Anweisungen geschlossen:
– CLOSE
– COMMIT WORK
– DROP CURSOR
– DROP TEMPORARY VIEW, falls der Cursor den angesprochenen temporären View referenziert

---

OPEN cursor

---

cursor
Name des Cursors, der geöffnet werden soll.

Beispiel

Alle Aufträge lesen, die vor dem 1.1.1994 fertiggestellt wurden.

```
DECLARE cur_fertig CURSOR FOR
  SELECT knr,atext FROM auftrag WHERE fertigist < '1994-01-01';

OPEN cur_fertig;
```

PERMIT - Benutzeridentifikation für Old-Style angeben

Um mit SESAM/SQL V1.x erstellte Programme auch weiterhin in unveränderter Form ablaufen lassen zu können, ist die PERMIT-Anweisung weiterhin erlaubt, die Ausführung einer PERMIT-Anweisung hat jedoch im New-Style keinerlei Auswirkungen. Ein SESAM/SQL V1.x-New-Style-Programm kann in der aktuellen Version von SESAM/SQL nur erfolgreich ausgeführt werden, wenn ein aktueller Berechtigungsschlüssel mit einem gültigen Systemzugang eingestellt ist (siehe OPTION AUTHORIZATION, SET AUTHORIZATION und PARAMETER DYNAMIC AUTHORIZATION), für den mit GRANT die benötigten Privilegien für die angesprochenen SQL- und CALL-DML/SQL-Tabellen definiert wurden.

Old-Style-Prozeduren benötigen für den Zugriff auf kennwortgeschützte CALL-DML-Tabellen auch weiterhin ein Kennwort (siehe SESAM/SQL V1, Aufbau und Wartung [46], Kapitel „Der SESAM-Kennwortschutz“). Diese Benutzeridentifikation kann wie in DRIVE/WINDOWS V1.1 mit der PERMIT-Anweisung für den Old-Style- bzw. Mischbetrieb bereitgestellt werden. Beim ersten DO oder CALL während der DRIVE-Sitzung (TIAM-Session oder UTM-Vorgang) auf eine SESAM-Old-Style-Prozedur wird das zuletzt eingegebene PERMIT-Kennwort an das Old-Style-Laufzeitsystem übergeben. Dieses Kennwort gilt dann für alle Old-Style-Zugriffe der DRIVE-Sitzung. Insbesondere kann also im Mischbetrieb auf verschiedene kennwortgeschützte CALL-DML-Tabellen nur immer mit ein- und demselben sessionweiten Kennwort zugegriffen werden.

Ein SESAM/SQL V1.x-Programm muß vor dem Ablauf unter der aktuellen Version von SESAM/SQL neu übersetzt werden (siehe auch SESAM/SQL-Server „Umstellen von SESAM/SQL-Datenbanken und -Anwendungen“ [22] und DRIVE-Programmiersystem [13]).

Die Anweisung PERMIT leitet keine Transaktion ein.

```
PERMIT SCHEMA = { tabelle | variable } [ PASSWORD = wert ]
```

---

tabelle

Einfacher Name einer CALL-DML-Tabelle (siehe Metavariable *tabellenangabe*).

Die Namen der zugreifbaren CALL-DML-Tabellen können der DBH-Startanweisung ADD-OLD-TABLE-CATALOG-LIST entnommen werden.

variable

Siehe Metavariable *variable*. Enthält den einfachen Namen einer CALL-DML-Tabelle.

PASSWORD = *wert*

Wertzuweisung für ein Kennwort (siehe Metavariable *wert*). *wert* muß eine alphanumerische Variable oder ein Literal sein, welche(s) das Kennwort enthält. Das Kennwort darf max. 3 Zeichen lang sein und muß den SESAM-Konventionen genügen. Im Programm-Modus muß *wert* als Variable angegeben werden.

PASSWORD nicht angegeben:

Es gilt die Standardvereinbarung PASSWORD = '␣␣␣'.

Beispiel

```
PERMIT SCHEMA = "GEN-CALLRBT-2"
       PASSWORD = 'XX3';
```

PRAGMA - Pragmaklauseln vereinbaren

PRAGMA ermöglicht die Vereinbarung von Pragmaklauseln. Die vereinbarten Klauseln werden von DRIVE/WINDOWS 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 sind im Abschnitt „Pragmas“ auf Seite 62 kurz dargestellt. Für Einzelheiten wird auf die Handbücher zu SESAM/SQL-Server [18] und [45] verwiesen.

Die Anweisung PRAGMA ist nicht ausführbar und leitet daher keine Transaktion ein.

Die PRAGMA-Anweisung ist nur im Programm-Modus erlaubt. Ein statisches PRAGMA kann überall zwischen PROCEDURE und END PROCEDURE stehen, ein dynamisches PRAGMA überall, wo EXECUTE erlaubt ist. Ein statisches PRAGMA wird zum Übersetzungszeitpunkt ausgewertet (ist also nicht ausführbar) und wirkt in DRIVE/WINDOWS auf die textuell nächste statische SQL-Anweisung und nur auf diese. 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 alle Pragmaklauseln auf die SQL-Anweisung wirken (siehe unten). Ein dynamisches PRAGMA wird zum Ablaufzeitpunkt ausgewertet und wirkt in DRIVE/WINDOWS auf die chronologisch nächste SQL-Anweisung und nur auf diese. Die DRIVE-Wirkung entspricht an der Benutzeroberfläche der von statischen PRAGMA-Anweisungen. Statische PRAGMA-Anweisungen wirken daher nur auf statische SQL-Anweisungen, und dynamische PRAGMA-Anweisungen wirken nur auf dynamische SQL-Anweisungen.

```
PRAGMA literal

literal::= '{ pragma_klausel },...'

pragma_klausel::= { PREFETCH n |
                    EXPLAIN INTO datei |
                    IGNORE INDEX indexname |
                    OPTIMIZATION LEVEL n |
                    SIMPLIFICATION { ON | OFF } |
                    ISOLATION LEVEL
                      { READ UNCOMMITTED |
                        READ COMMITTED |
                        REPEATABLE READ |
                        SERIALIZABLE } |
                    DATA TYPE OLDEST |
                    CHECK { ON | OFF } }
```

literal

Ein alphanumerisches Literal, das eine Liste von Pragmaklauseln enthält. Der Inhalt des Literals wird von DRIVE/WINDOWS nicht geprüft.

pragma_klausel

Bei SESAM/SQL syntaktisch mögliche Pragmaklausel.

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.

DATA TYPE und CHECK haben in DRIVE/WINDOWS keine SESAM-Wirkung.

Hat eine in *literal* enthaltene Pragmaklausel keine SESAM-Wirkung, so tritt in DRIVE/WINDOWS beim Übersetzen (statisches PRAGMA) bzw. bei der Ausführung (dynamisches PRAGMA) der zugeordneten SQL-Anweisung ein Fehler mit &DML_STATE = 'SQL ERROR' und &SQL_CLASS = '01' auf.

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 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 SESAM/SQL-Server SQL-Sprachbeschreibung Teil 1 [18]).

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/WINDOWS 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 Ausführungzeitpunkt 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 [45] 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 = '$YDRI20.EXPLAINFILE';

/* Die Datei $YDRI20.EXPLAINFILE muß mit USER-ACC = ALL-USERS */
/* katalogisiert sein */

EXECUTE 'SET SESSION AUTHORIZATION ""DRI-USER1" "';

/* Der Berechtigunsschlüssel 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 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 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 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 SIMPLIFICATION

Die kompletten Optimierungstechniken zur Simplification (Teil der algebraischen Optimierung) werden ein- bzw. abgeschaltet, d.h. die dort skizzierten Optimierungsschritte werden entweder alle durchlaufen (ON) oder keiner von ihnen (OFF).

SIMPLIFICATION { ON | OFF }

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

Da in DRIVE/WINDOWS die ADD COLUMN-Klausel bei ALTER TABLE für CALL-DML-Tabellen nicht erlaubt ist, hat die Klausel in DRIVE/WINDOWS keine SESAM-Wirkung.

Pragmaklausel CHECK

Die Pragmaklausel CHECK hat in DRIVE/WINDOWS keine SESAM-Wirkung.

RESTORE - Cursor wiederherstellen

RESTORE öffnet einen mit STORE gespeicherten Cursor.

Der Cursor wird mit derselben Cursorbeschreibung geöffnet wie beim letzten OPEN. Wurden Variablen zwischenzeitlich verändert, hat dies keine Auswirkungen auf die ermittelte Ergebnistabelle.

Auch die in der Cursorbeschreibung vorkommenden Literale CURRENT USER bzw. USER und SYSTEM USER sowie die Zeitfunktionen CURRENT DATE, CURRENT TIME und CURRENT TIMESTAMP werden nicht neu ausgewertet.

Die mit STORE abgespeicherte Cursorposition kann verlorengehen, wenn in der Zwischenzeit in derselben oder einer anderen Transaktion Sätze ab der gespeicherten Position gelöscht wurden oder der Satz, auf den der Cursor positioniert war, so geändert wurde, daß er nicht mehr zur Cursortabelle gehört.

Ist für den Cursor keine Cursorposition mehr gespeichert, wird der Cursor nicht geöffnet und ein entsprechender SQLSTATE gesetzt.

Ansonsten wird der Cursor geöffnet und die Cursorposition wiederhergestellt. Um einen Satz zu löschen (DELETE ... WHERE CURRENT OF) oder zu ändern (UPDATE ... WHERE CURRENT OF), muß der Cursor mit FETCH auf diesen Satz positioniert werden.

Nach Ausführung der RESTORE-Anweisung werden alle mit STORE für diesen Cursor gespeicherten Informationen gelöscht. Vor der nächsten RESTORE-Anweisung muß zuerst eine Cursorposition mit STORE gespeichert werden.

Der wiederherzustellende Cursor muß mit STORE in derselben DRIVE-Sitzung (Dialog-Cursor) bzw. in demselben Programm-Lauf (Programm-Cursor) gespeichert und zum RESTORE-Zeitpunkt geschlossen sein. Die Transaktionen, die die STORE- und RESTORE-Anweisung enthalten, müssen denselben Isolationslevel bzw. Konsistenzlevel haben (siehe Anweisung SET TRANSACTION).

Ist ein dynamischer Cursor zum RESTORE-Zeitpunkt bei SESAM nicht mehr deklariert, so wird er von DRIVE/WINDOWS nachdeklariert. Da er dann aber nicht geöffnet ist, setzt SESAM bei RESTORE einen entsprechenden SQLSTATE.

RESTORE cursor

---

cursor
Name des Cursors, der wiederhergestellt werden soll.

Cursor nach der RESTORE-Anweisung bearbeiten

Nach einer RESTORE-Anweisung muß mit einer FETCH-Anweisung auf einen Satz positioniert werden.

Beispiel

FETCH NEXT positioniert auf den nächsten Satz in der Cursortabelle.

Erst dann kann mit einer UPDATE- oder DELETE-Anweisung auf den Cursor zugegriffen werden.

REVOKE - Privilegien entziehen

REVOKE entzieht Berechtigungsschlüsseln oder der Allgemeinheit Tabellen- und Spalten-Privilegien oder Sonder-Privilegien. Die temporären Views des Berechtigungsschlüssels, die auf der Tabelle beruhen, werden gelöscht .

Privilegien können einem Berechtigungsschlüssel nur von dem Berechtigungsschlüssel entzogen werden, der das Privileg vergeben hat (siehe Anweisung GRANT). Wenn die Privilegien weitergegeben wurden, können sie nicht entzogen werden. Ein Privileg, aufgrund dessen ein View oder eine Referenzbedingung definiert wurde, kann nicht entzogen werden.

Welche Privilegien welchen Berechtigungsschlüsseln zugeordnet sind, erfahren Sie in den Views TABLE_PRIVILEGES, COLUMN_PRIVILEGES und CATALOG_PRIVILEGES des INFORMATION_SCHEMA (siehe SESAM/SQL-Server SQL-Sprachbeschreibung Teil1 [18], Kapitel „Informationsschemata“).

Die REVOKE-Anweisung hat zwei Formate, ein Format für den Entzug Tabellen- und Spalten-Privilegien, das andere für den Entzug Sonder-Privilegien.

REVOKE-Format für Tabellen- und Spalten-Privilegien:

```
REVOKE { ALL PRIVILEGES | { tabellen_und_spalten_privileg },... }

ON [ TABLE ] tabelle

FROM { PUBLIC | { berechtigungsschlüssel },... } RESTRICT


tabellen_und_spalten_privileg::= { SELECT |
                                   DELETE |
                                   INSERT |
                                   UPDATE [ ( { spalte },...) ] |
                                   REFERENCES [ ( { spalte },... ) ] }
```

ALL PRIVILEGES
: Alle Tabellen-Privilegien werden entzogen, die der aktuelle Berechtigungsschlüssel entziehen darf. ALL PRIVILEGES umfaßt die Privilegien SELECT, DELETE, INSERT, UPDATE und REFERENCES.

tabellen_und_spalten_privileg
: Die Tabellen- und Spalten-Privilegien werden einzeln entzogen. Sie können mehrere 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 einfacher 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 einfacher 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.

REVOKE-Format für Sonder-Privilegien:

```
REVOKE { ALL SPECIAL PRIVILEGES | CREATE SCHEMA }

ON CATALOG catalog

FROM { PUBLIC | { berechtigungsschlüssel },... }  RESTRICT
```

ALL SPECIAL PRIVILEGES
: Alle Sonder-Privilegien werden entzogen, die der aktuelle Berechtigungsschlüssel entziehen darf (siehe Anweisung GRANT).

CREATE SCHEMA
: Sonder-Privileg, das das Definieren von Datenbank-Schemas erlaubt, wird entzogen.

FROM PUBLIC
: Die Sonder-Privilegien werden der Allgemeinheit entzogen. Individuelle Sonder-Privilegien von Berechtigungsschlüsseln werden dadurch nicht berührt.

FROM {*berechtigungsschlüssel*},...
: Die Sonder-Privilegien werden dem Benutzer mit dem Berechtigungsschlüsssel *berechtigungsschlüssel* entzogen. Sie können mehrere Berechtigungsschlüssel angeben.

ON CATALOG *catalog*
: Name der Datenbank, für die Sie Sonder-Privilegien entziehen wollen.

Beispiel

Die folgende REVOKE-Anweisung entzieht das UPDATE-Privileg für alle Spalten der Tabelle telefonliste.

```
REVOKE UPDATE ON TABLE telefonliste FROM berta RESTRICT;
```

ROLLBACK WORK - Transaktion zurücksetzen

ROLLBACK WORK beendet eine Transaktion und setzt Änderungen zurück, die seit dem Ende der letzten SQL-Transaktion durchgeführt wurden.

ROLLBACK WORK setzt folgende Änderungen zurück:
– Alle innerhalb der Transaktion geöffneten Cursor werden geschlossen.
– geänderte Daten in SQL-Schemas
– dynamisch formulierte Anweisungen und Cursorbeschreibungen
– mit STORE abgespeicherte Cursorpositionen
– SET-Anweisungen eines (dynamischen) Programms (AUTHORIZATION, CATALOG, SCHEMA), die nicht mit COMMIT WORK festgeschrieben wurden
– mit PERMIT angegebene Benutzeridentifikationen für Old-Style-Prozeduren

Im Dialog-Modus setzt ROLLBACK WORK die Parameter-Werte weder bei SESAM noch bei DRIVE/WINDOWS zurück (siehe Anweisung PARAMETER DYNAMIC).

Die Anweisung SET TRANSACTION kann nicht zurückgesetzt werden.

Zu Regeln in Verteilten Anwendungen siehe DRIVE-Programmiersprache, Kapitel „Verteilte Anwendungen“ bzw. „Verteilte Transaktionsverarbeitung“.

Mit der ersten fehlerfreien transaktionseinleitenden SQL-Anweisung nach ROLLBACK WORK beginnt eine neue Transaktion.

```
ROLLBACK [ WORK ] [ WITH RESET ]
```

WITH RESET

WITH RESET ist nur im Programm-Modus erlaubt und bezieht sich nur auf die DRIVE-Steuerung.
Das Programm wird auf den Stand des letzten COMMIT WORK zurückgesetzt und mit der Anweisung fortgesetzt, die diesem COMMIT WORK folgt (Verhalten wie bei einem internen ROLLBACK WORK). Der Inhalt der DRIVE-Variablen wird auf die Werte zurückgesetzt, die zum Zeitpunkt des COMMIT WORK galten. In SINIX und MS-Windows werden außerdem bei grafischer Oberfläche alle nach diesem COMMIT WORK geöffneten Fenster geschlossen (siehe DRIVE- Programmiersprache [2], Kapitel „Transaktionskonzept“). Wurde seit Programmstart noch kein COMMIT WORK durchlaufen, wird das Programm mit einer Fehlermeldung abgebrochen.
Bezüglich der Datenbank sind die Anweisungen ROLLBACK WORK und ROLLBACK WORK WITH RESET identisch.

Wird die Anweisung ROLLBACK WORK WITH RESET ohne eine Bedingung angegeben, besteht die Gefahr einer Endlosschleife, da das DRIVE-Programm mit der Anweisung fortgesetzt wird, die auf die letzte COMMIT-WORK-Anweisung folgt.

Implizite Ausführung von ROLLBACK WORK

SESAM/SQL setzt eine Transaktion durch implizite Ausführung eines ROLLBACK WORK zurück, wenn eine der folgenden Situationen eintritt:

– Innerhalb der aktuellen Transaktion tritt ein nicht behebbarer Fehler auf.

– Für zwei oder mehrere Transaktionen, die gleichzeitig auf bestimmte SQL-Daten zugreifen, kann der eingestellte Isolationslevel nicht gewährleistet werden (siehe auch SESAM/SQL-Server „Basishandbuch“ [20]).

– Eine Transaktion ist für lange Zeit unterbrochen und belegt Betriebsmittel, die von anderen Transaktionen benötigt werden (siehe auch SESAM/SQL-Server „Basishandbuch“ [20]).

Die Wirkung ist dieselbe wie beim expliziten Aufruf von ROLLBACK WORK WITH RESET.

SELECT - Einzelnen Satz lesen

Mit der SELECT-Anweisung kann genau ein Satz einer Tabelle gelesen werden. Die gelesenen Spaltenwerte werden in Variablen abgespeichert.

Wenn die Ergebnistabelle mehr als einen Satz enthalten würde, liest die SELECT-Anweisung keinen Satz, es wird lediglich ein entsprechender SQLSTATE gesetzt. Zum Lesen von mehrsätzigen Ergebnistabellen muß ein Cursor verwendet werden.

Für die SELECT-Anweisung müssen Sie Eigentümer der Tabellen sein, aus denen die Werte abgefragt werden, oder das SELECT-Privileg für die angesprochenen Tabellen besitzen.

Im Programm-Modus überträgt SELECT die Spaltenwerte in die mit INTO angegebenen Variablen.
Im Dialog-Modus gibt SELECT die Spaltenwerte typgerecht auf den Bildschirm aus (siehe DRIVE-Sprachbeschreibung [2], Kapitel „Variablen und Konstanten einsetzen“). Zur NULL-Wert-Darstellung auf dem Bildschirm siehe DRIVE-Sprachbeschreibung [2], Kapitel „Datenbanken bearbeiten“.

Über die FROM-Klausel und gegebenenfalls WHERE-Klausel können Sie einen Join bilden, d.h. zwei oder mehrere Tabellen miteinander verknüpfen und das kartesische Produkt aus allen beteiligten Tabellen bilden (siehe auch Metavariable *join_ausdruck*).

```
SELECT [ { ALL | DISTINCT } ] select_liste
    [ INTO { variable },... ]
    FROM tabellenangabe,...
    [ WHERE bedingung ]
    [ GROUP BY spalte,... ]
    [ HAVING bedingung ]
```

Mit Ausnahme der INTO-Klausel sind die Klauseln der SELECT-Anweisung wie für den SELECT-Ausdruck definiert (siehe Metavariable *select_ausdruck*).

INTO

Bei einer SELECT-Anweisung im Programm-Modus müssen Sie mit der INTO-Klausel die Variablen angeben, in die die Spaltenwerte des Ergebnissatzes ausgegeben werden sollen.

variable

Name einer Variablen, der ein Spaltenwert des Ergebnissatzes zugewiesen wird oder Spezifikation einer Komponente oder einer Komponentenliste einer strukturierten Variablen (siehe Metavariable *variable*).

Der Datentyp einer Variable muß mit dem Datentyp der zugehörigen Ergebnisspalte verträglich sein. Ist eine Ergebnisspalte ein Aggregat mit mehreren Elementen, muß die zugehörige Variable ein Vektor mit derselben Anzahl von Elementen sein.

Die Anzahl der angegebenen Variablen muß mit der Anzahl der Spalten von *select_liste* der SELECT-Anweisung übereinstimmen. Der Wert der i-ten Spalte in *select_liste* wird der i-ten Variable in der INTO-Klausel zugewiesen. Ist der zuzuweisende Wert der NULL-Wert, wird die Variable auf NULL gesetzt.

Gibt es keinen oder mehr als einen Ergebnissatz, werden alle Variablen nicht gesetzt.

Gibt es keinen Ergebnissatz, wird ein SQLSTATE gesetzt, der mit WHENEVER &DML_STATE IN ('TABLE END') behandelt werden kann; gibt es mehr als einen Ergebnissatz, wird ein SQLSTATE gesetzt, der mit WHENEVER &DML_STATE IN ('SQL ERROR') behandelt werden kann.

Beispiel

Name und Adresse der Firma mit der Kundennummer 100 lesen und in den Variablen Firma, Plz, Ort und Strasse ablegen.

Da die Kundennummer innerhalb der Tabelle kunde eindeutig ist, ist sichergestellt, daß die Abfrage maximal einen Satz liefert:

```
SELECT firma, plz, ort, strasse
INTO &Firma,&Plz,&Ort,&Strasse
FROM kunde
WHERE knr=100;
```

SET CATALOG - Datenbanknamen voreinstellen

SET CATALOG legt den voreingestellten Datenbanknamen für einfache Schemanamen fest, die in nachfolgenden EXECUTE- Anweisungen vorkommen. Für alle statischen SQL-Anweisungen wird weiterhin der mit OPTION CATALOG voreingestellte Datenbankname für einfache Schemanamen ergänzt. Bis zur ersten Ausführung von SET CATALOG (oder SET SCHEMA) wird als voreingestellter Datenbankname für alle dynamischen Anweisungen der mit PARAMETER DYNAMIC CATALOG festgelegte Datenbankname verwendet.

Der mit SET CATALOG voreingestellte Datenbankname gilt bis ein neuer Datenbankname mit SET CATALOG bzw. SET SCHEMA eingestellt wird oder bis zum Anwendungsende. Im Dialog-Modus gilt immer die DRIVE-Voreinstellung durch PARAMETER DYNAMIC CATALOG. Zum Zusammenspiel der SET CATALOG-Anweisung mit den Anweisungen OPTION CATALOG und PARAMETER DYNAMIC CATALOG siehe auch Kapitel „Arbeiten mit SESAM/SQL V2“ auf Seite 25.

Die Anweisung SET CATALOG ist nur im Programm-Modus erlaubt und leitet keine Transaktion ein. Sie ist auch innerhalb von Transaktionen erlaubt und sollte, zur Verdeutlichung ihrer ausschließlichen Wirkung auf dynamische SQL-Anweisungen, nur als dynamische Anweisung verwendet werden.

```
SET CATALOG voreingest_catalog

    voreingest_catalog::= { alphanumerisches_literal | variable }
```

---

voreingest_catalog
Name der Datenbank, die für den aktuellen Programm-Lauf voreingestellt wird (max. 18 Zeichen lang).

alphanumerisches_literal
Der Datenbankname wird als alphanumerisches Literal angegeben.

variable
Der Datenbankname wird als alphanumerische Variable vom Typ CHARACTER oder VARCHAR angegeben. Die Variable darf kein Vektor sein. Der Variablen darf nicht der NULL-Wert zugeordnet sein.

Die SET CATALOG-Anweisung wird in dynamischen Programmen eingesetzt. Mit ihr kann der DRIVE-sitzungsspezifische Datenbankname verändert werden. Alle nachfolgenden SQL-Anweisungen des dynamischen Programms können sich nur auf diesen Datenbanknamen beziehen.

Gültigkeit des voreingestellten Datenbanknamens:

Maximal bis Anwendungsende (Übergang in den Dialog-Modus) oder bis zur nächsten SET CATALOG-Anweisung in demselben oder in einem anderen Programm, welche auch innerhalb der Transaktion abgesetzt werden kann.

Die Anweisung ist nur im Programm-Modus erlaubt.

Sie sollte in dynamischer Form (EXECUTE) angegeben werden, da sie nur für dynamische Anweisungen Wirkung zeigt. Statische Anweisungen verwenden den Datenbanknamen der Compileroption als Voreinstellung.

Beispiel

```
EXECUTE 'SET CATALOG "meinedb"';
```

Nachfolgende dynamische SQL-Anweisungen verwenden zur Qualifizierung von einfachen Schemanamen den Datenbanknamen 'meinedb'.

SET SCHEMA - Schemanamen voreinstellen

SET SCHEMA legt den voreingestellten Schemanamen für einfache Namen von Integritätsbedingungen, Indizes und Tabellen fest, die in nachfolgenden EXECUTE-Anweisungen vorkommen. Für alle statischen SQL-Anweisungen wird weiterhin der mit OPTION SCHEMA voreingestellte Schemaname für einfache Namen von Integritätsbedingungen, Indizes und Tabellen ergänzt. Bis zur ersten Ausführung von SET SCHEMA wird als voreingestellter Schemaname für alle dynamischen Anweisungen der mit PARAMETER DYNAMIC SCHEMA festgelegte Schemaname verwendet.

Der mit SET SCHEMA voreingestellte Schemaname gilt bis ein neuer Schemaname mit SET SCHEMA eingestellt wird oder bis zum Anwendungsende. Im Dialog-Modus gilt immer die DRIVE-Voreinstellung durch PARAMETER DYNAMIC SCHEMA. Zum Zusammenspiel der SET SCHEMA-Anweisung mit den Anweisungen OPTION SCHEMA und PARAMETER DYNAMIC SCHEMA siehe auch Kapitel „Arbeiten mit SESAM/SQL V2“ auf Seite 25.

Wird in der Anweisung SET SCHEMA ein einfacher Schemaname mit einem Datenbanknamen qualifiziert, wird auch der mit PARAMETER DYNAMIC CATALOG voreingestellte Datenbankname für die nachfolgenden dynamischen SQL-Anweisungen geändert.

Die Anweisung SET SCHEMA ist nur im Programm-Modus erlaubt und leitet keine Transaktion ein. Sie ist auch innerhalb von Transaktionen erlaubt und sollte, zur Verdeutlichung ihrer ausschließlichen Wirkung auf dynamische SQL-Anweisungen, nur als dynamische Anweisung verwendet werden.

```
SET SCHEMA voreingest_schema

    voreingest_schema::= { alphanumerisches_literal | variable }
```

---

voreingest_schema
: Name des Schemas, das für die aktuelle DRIVE-Sitzung voreingestellt wird (max. 31 Zeichen lang). Der einfache Schemaname kann mit einem Datenbanknamen qualifiziert werden.
  Wird der einfache Schemaname mit einem Datenbanknamen qualifiziert, wird dieser Datenbankname als voreingestellter Datenbankname verwendet, wie wenn er mit SET CATALOG eingestellt worden wäre.

alphanumerisches_literal
: Der Schemaname wird als alphanumerisches Literal angegeben.

variable

Der Schemaname wird als alphanumerische Variable vom Typ CHARACTER oder VARCHAR angegeben. Die Variable darf kein Vektor sein. Der Variablen darf nicht der NULL-Wert zugeordnet sein.

Die SET SCHEMA-Anweisung wird in dynamischen Programmen eingesetzt. Mit ihr kann der DRIVE-sitzungsspezifische Schemaname verändert werden. Alle nachfolgenden SQL-Anweisungen des dynamischen Programms können sich nur auf diesen Schemanamen beziehen.

Gültigkeit des voreingestellten Schemnamens:

Maximal bis Anwendungsende (Übergang in den Dialog-Modus) oder bis zur nächsten SET SCHEMA-Anweisung in demselben oder in einem anderen Programm, welche auch innerhalb der Transaktion abgesetzt werden kann.

Die Anweisung ist nur im Programm-Modus erlaubt.

Sie sollte in dynamischer Form (EXECUTE) angegeben werden, da sie nur für dynamische Anweisungen Wirkung zeigt. Statische Anweisungen verwenden den Schemanamen der Compileroption als Voreinstellung.

Beispiel

```
EXECUTE 'SET SCHEMA "meinedb.andromeda"';
```

Nachfolgende dynamische SQL-Anweisungen verwenden zur Namensqualifizierung von SQL-Objekten den Schemanamen ‚meinedb.andromeda' und den Datenbanknamen ‚meinedb'.

SET SESSION AUTHORIZATION - Berechtigungsschlüssel festlegen

SET SESSION AUTHORIZATION legt den aktuellen Berechtigungsschlüssel für die dynamischen SQL-Anweisungen einer DRIVE-Anwendung fest.

Eine SET SESSION-Anweisung überschreibt die folgenden Werte:
– die Standard-Vorbelegung bei SESAM (D0USER)
– die Voreinstellung, die mit PARAMETER DYNAMIC SESSION AUTHORIZATION vorgenommen wurde, allerdings nur bis zum Anwendungsende (Übergang in den Dialog-Modus)
– den mit der letzten SET SESSION AUTHORIZATION-Anweisung eingestellten Wert.

Der aktuelle Berechtigungsschlüssel für die dynamischen SQL-Anweisungen einer DRIVE-Anwendung wird entweder mit der Anweisung PARAMETER DYNAMIC AUTHORIZATION voreingestellt oder mit der Anweisung SET SESSION AUTHORIZATION festgelegt. Wird an beiden Stellen kein dynamischer Berechtigungsschlüssel festgelegt, so wird der von SESAM voreingestellte Berechtigungsschlüssel D0USER als aktueller Berechtigungsschlüssel der DRIVE-Anwendung verwendet.

Die Anweisung SET SESSION AUTHORIZATION ist nur im Programm-Modus und nur innerhalb einer EXECUTE-Anweisung erlaubt. Sie leitet keine Transaktion ein und darf nur außerhalb einer SQL-Transaktion verwendet werden.

SET SESSION -Anweisungen können keinen Berechtigungsschlüssel ändern, der mit OPTION AUTHORIZATION angegeben wurde, d.h. sie wirken nicht auf statische SQL-Anweisungen.

```
SET SESSION AUTHORIZATION neuer_berechtigungsschlüssel

    neuer_berechtigungsschlüssel::= { alphanumerisches_literal | variable }
```

---

neuer_berechtigungsschlüssel
Name des neuen Berechtigungsschlüssels, der für den aktuellen Programm-Lauf gelten soll. Der neue Berechtigungsschlüssel gilt bis zur nächsten Anweisung SET SESSION AUTHORIZATION oder bis Anwendungsende (Übergang in den Dialog-Modus). Der Berechtigungsschlüssel darf max. 18 Zeichen lang sein.

literal
Der neue Berechtigungsschlüssel wird als alphanumerisches Literal angegeben.

variable

Der neue Berechtigungsschlüssel wird als alphanumerische Variable angegeben. Die Variable darf kein Vektor sein. Der Variablen darf nicht der NULL-Wert zugeordnet sein.

Die SET SESSION AUTHORIZATION-Anweisung wird in dynamischen Programmen eingesetzt. Mit ihr kann der DRIVE-sitzungsspezifische Berechtigungsschlüssel verändert werden. Alle nachfolgenden SQL-Anweisungen des dynamischen Programms werden nur für diesen Berechtigungsschlüssel ausgeführt.

Gültigkeit des aktuellen Berechtigungsschlüssels:

Maximal bis Anwendungsende (Übergang in den Dialog-Modus) oder bis zur nächsten SET SESSION AUTHORIZATION-Anweisung in demselben oder in einem anderen Programm, welche nur außerhalb von Transaktionen abgesetzt werden kann.

Die Anweisung ist nur im Programm-Modus erlaubt und nur innerhalb einer EXECUTE-Anweisung, sie darf also nicht statisch im Programm stehen. Sie muß in dynamischer Form angegeben werden, da sie bei DRIVE/WINDOWS nur für dynamische Anweisungen Wirkung zeigt. Statische Anweisungen verwenden den Berechtigungsschlüssel der Compileroption als aktuellen Berechtigungsschlüssel.

Beispiel

Für die DRIVE-Anwendung wird ein neuer Berechtigungsschlüssel für dynamische SQL-Anweisungen festgelegt. Der aktuelle UTM- bzw. BS2000-Benutzer muß einen Systemzugang mit diesem Berechtigungsschlüssel besitzen.

```
EXECUTE 'SET SESSION AUTHORIZATION "berta"';
```

Alle nachfolgenden dynamischen SQL-Anweisungen werden mit dem Berechtigungsschlüssel 'berta' ausgeführt.

SET TRANSACTION - Transaktionseigenschaften festlegen

Mit SET TRANSACTION können Sie Isolations- bzw. Konsistenzlevel und Transaktionsmodus der nachfolgenden SQL-Transaktion festlegen.

Der Isolations- bzw. Konsistenzlevel einer Transaktion gibt an, wie stark das Lesen von Sätzen in der Transaktion durch gleichzeitige Schreibzugriffe einer konkurrierenden Transaktion beeinflußt wird.

Mit dem Transaktionsmodus können Sie bestimmen, ob innerhalb der nachfolgenden Transaktion Tabellensätze nur gelesen oder auch geändert werden dürfen.

i Wenn Sie einen Isolations- bzw. Konsistenzlevel festlegen, beeinflussen Sie auch den Grad an Parallelität und damit die Performanz: je mehr Phänomene ausgeschlossen werden, desto geringer ist der Grad an Parallelität.

Die mit SET TRANSACTION getroffenen Einstellungen sind nur für die unmittelbar folgende Transaktion gültig. Nach Ende der Transaktion gelten wieder die Voreinstellungen.

Die Anweisung SET TRANSACTION leitet keine Transaktion ein und darf nur außerhalb einer SQL-Transaktion verwendet werden.

```
SET TRANSACTION { level [ [ , ] transaktionsmodus ] |
                transaktionsmodus [ [ , ] level ] }

level::= { ISOLATION LEVEL { READ UNCOMMITTED | READ COMMITTED |
                             REPEATABLE READ | SERIALIZABLE } |
           CONSISTENCY LEVEL konsistenzlevel }

transaktionsmodus::= { READ ONLY | READ WRITE }
```

Wie in früheren SESAM/SQL Versionen können Sie das Komma zwischen den beiden Angaben weglassen. Soll Ihre Anwendung portierbar sein, müssen Sie allerdings das Komma setzen.

ISOLATION LEVEL

Isolationslevel einstellen.

Arbeiten mehrere Transaktionen gleichzeitig mit denselben Tabellen, können Phänomene eintreten, in denen Lesezugriffe in einer Transaktion durch gleichzeitige schreibende Zugriffe einer anderen Transaktion beeinflußt werden. Mit dem Isolationslevel legen Sie fest, welche dieser Phänomene in den nachfolgenden SQL-Transaktionen zugelassen sein sollen.

Folgende Phänomene sind von Bedeutung:

– dirty read:
  Eine Transaktion ändert einen Satz oder nimmt einen Satz neu auf. Eine zweite Transaktion liest diesen Satz, bevor die erste Transaktion die Änderung festgeschrieben hat. Wird die erste Transaktion zurückgesetzt, hat die zweite Transaktion einen Satz gelesen, der nie festgeschrieben worden ist.

– non-repeatable read:
  Eine Transaktion liest einen Satz. Bevor diese Transaktion beendet wird, ändert oder löscht eine zweite Transaktion denselben Satz und schreibt die Änderung fest. Versucht danach die erste Transaktion diesen Satz nochmals zu lesen, werden entweder geänderte Werte zurückgegeben oder ein Fehler, weil der Satz inzwischen gelöscht wurde. Die Leseoperation liefert also ein anderes Ergebnis als beim ersten Mal.

– phantom:
  Eine Transaktion liest Sätze, die eine bestimmte Suchbedingung erfüllen. Anschließend nimmt eine zweite Transaktion Sätze auf, die ebenfalls diese Suchbedingung erfüllen. Wiederholt die erste Transaktion danach die Abfrage, enthält die Ergebnistabelle auch die neu aufgenommenen Sätze.

READ UNCOMMITTED

Isolationslevel, der den geringsten Schutz vor konkurrierenden Transaktionen bietet. Alle oben beschriebenen Phänomene sind möglich. In der nachfolgenden SQL-Transaktion können gelesene Sätze noch nicht festgeschrieben sein und von anderen Transaktionen nach dem Lesen geändert werden.

READ UNCOMMITTED ist nicht zulässig, wenn gleichzeitig der Transaktionsmodus READ WRITE festgelegt wurde.

READ COMMITTED

Die Phänomene non-repeatable read und phantom können auftreten. In der nachfolgenden SQL-Transaktion können gelesene Sätze von anderen Transaktionen nach dem Lesen geändert werden. Es werden keine Sätze gelesen, die noch nicht festgeschrieben sind.

REPEATABLE READ
Das Phänomen phantom kann auftreten. Die Phänomene non-repeatable read und dirty read sind nicht möglich.

SERIALIZABLE
Vollständiger Schutz vor konkurrierenden Transaktionen ist gewährleistet. Die Phänomene dirty read, non-repeateable read und phantom können nicht auftreten. Die Existenz konkurrierender Transaktionen ist für die nachfolgende Transaktion nicht sichtbar.

CONSISTENCY LEVEL
Alternativ zum Isolationslevel bietet SESAM/SQL aus Gründen der Aufwärtskompatibilität zu früheren Versionen auch den Parameter CONSISTENCY LEVEL an. Damit definieren Sie einen Konsistenzlevel, der analog zum Isolationslevel festlegt, ob die Phänomene dirty read, non-repeatable read und phantom auftreten können.

konsistenzlevel
Vorzeichenlose Ganzzahl, mit: $0 \leq$ *konsistenzlevel* ≤ 4.

Level	**gesetzte Sperren**	**gelesene Sätze**
0	Gelesene Sätze werden nicht gegen Ändern durch andere Transaktionen gesperrt.	alle Sätze, auch die von anderen Transaktionen gegen Änderungen gesperrte
1	Gelesene Sätze werden gegen Änderungen durch andere Transaktionen (bis Transaktionsende) gesperrt, außer wenn diese schon gesperrt sind.	wie bei 0
2	wie bei 0	nur die Sätze, die nicht von anderen Transaktionen gegen Ändern gesperrt sind
3	Gelesene Sätze werden gegen Änderungen durch andere Transaktionen (bis Transaktionsende) gesperrt.	wie bei 2
4	Gelesene Sätze werden wie bei 3 gesperrt. Bei nicht vorhandenen Sätzen wird durch Sperren gegen Änderungen von anderen Transaktionen sichergestellt, daß sie nicht von anderen Transaktionen eingefügt werden können.	wie bei 2

Die folgende Tabelle zeigt die Zuordnung von Konsistenz- zu Isolationslevel und welche Phänomene jeweils auftreten können.

Isolationslevel	**Konsistenz-level**	**dirty read**	**non-repeatable read**	**phantom**
READ UNCOMMITTED	0	x	x	x
-	1	x	x 1)	x
READ COMMITTED	2	-	x	x
REPEATABLE READ	3	-	-	x
SERIALIZABLE	4	-	-	-

zu 1)
das Phänomen non-repeatable read kann nur für Sätze auftreten, die vorher mit dirty read gelesen wurden.

READ ONLY
Transaktionsmodus READ ONLY einstellen.

Innerhalb der Transaktion sind nur lesende Datenbankzugriffe möglich. READ ONLY ist Voreinstellung beim Isolationslevel READ UNCOMMITTED bzw. den Konsistenzleveln 0 und 1.

READ WRITE
Transaktionsmodus READ WRITE einstellen.

Innerhalb der Transaktion sind lesende und schreibende Datenbankzugriffe möglich. READ WRITE ist Voreinstellung bei den Isolationsleveln READ COMMITTED, REPEATABLE READ und SERIALIZABLE bzw. bei den Konsistenzleveln 2, 3 und 4.

READ WRITE ist nicht zulässig, wenn gleichzeitig der Isolationslevel READ UNCOMMITED festgelegt wurde.

Voreinstellung

Existiert für Isolations- bzw. Konsistenzlevel ein Eintrag in der benutzerspezifischen Konfigurationsdatei (siehe SESAM/SQL-Server „Basishandbuch“ [20]), wird dieser Wert als Voreinstellung genommen. Andernfalls sind der Isolationslevel SERIALIZABLE, der Konsistenzlevel 4 und der Transaktionsmodus READ WRITE voreingestellt.

STORE - Cursorposition speichern

STORE speichert die aktuelle Cursorposition.

Am Ende einer Transaktion werden alle geöffneten Cursor geschlossen. Um in der folgenden Transaktion trotzdem wieder auf den Inhalt der Ergebnistabelle zugreifen zu können, muß vor Transaktionsende mit STORE die aktuelle Cursorposition gespeichert werden. Ein gespeicherter Cursor kann mit RESTORE wiederhergestellt werden.

Nach STORE ist kein FETCH mehr möglich.

Eine mit STORE gespeicherte Cursorposition wird durch die Anweisungen RESTORE und OPEN entwertet.

Der Cursor muß geöffnet und auf einen Satz positioniert sein. STORE schließt den Cursor nicht. Die gespeicherte Cursorposition bleibt maximal bis zum Ende der DRIVE-Sitzung (Dialog-Cursor) bzw. des Programm-Laufs (Programm-Cursor) erhalten.

STORE ist für einen PREFETCH-Cursor nicht erlaubt.

STORE cursor

---

cursor

Name des Cursors, dessen Position gespeichert wird.

Der Aufruf überschreibt eine zuvor mit STORE gespeicherte Cursorposition für denselben Cursor.

UPDATE - Spaltenwerte ändern

UPDATE ändert Spaltenwerte von Sätzen in einer Tabelle.

Die in der UPDATE-Anweisung (und in Voreinstellungen) vorkommenden Literale CURRENT USER bzw. USER und SYSTEM USER sowie die Zeitfunktionen CURRENT DATE, CURRENT TIME und CURRENT TIMESTAMP werden einmal ausgewertet, und die berechneten Werte gelten für alle Änderungen.

Um einen Satz in der angegebenen Tabelle zu ändern, müssen Sie Eigentümer dieser Tabelle sein oder das UPDATE-Privileg für jede zu ändernde Spalte besitzen. Zusätzlich muß der Transaktionsmodus der aktuellen Transaktion READ WRITE sein.

Sind für die Tabelle bzw. die betroffenen Spalten Integritätsbedingungen definiert, werden diese nach dem Ändern geprüft. Ist eine Integritätsbedingung verletzt, werden die Änderungen rückgängig gemacht und ein entsprechender SQLSTATE gesetzt.

```
UPDATE tabelle SET {
        { spalte | spalte(posnr) | spalte(min-max) }
        = { sqlausdruck | DEFAULT | NULL }
            },...
    [ WHERE { bedingung | CURRENT OF cursor } ]
```

---

tabelle
: Name der Tabelle, in der Sätze geändert werden sollen. Die Tabelle kann eine Basistabelle, ein änderbarer View oder ein änderbarer temporärer View sein (siehe Metavariable *tabellenangabe*). Bei einem dynamischen Cursor darf kein statischer temporärer View angegeben sein.

spalte
: Name einer einfachen Spalte, deren Inhalt geändert werden soll. Die Spalte muß in der Tabelle enthalten sein. Der Spaltenname darf nicht mit einer Tabellenangabe qualifiziert werden. Eine Spalte darf nur einmal innerhalb einer UPDATE-Anweisung vorkommen.

spalte(*posnr*)
: Element einer multiplen Spalte, dessen Wert geändert werden soll.

  Die multiple Spalte muß in der Tabelle enthalten sein. Werden mehrere Elemente einer multiplen Spalte angegeben, muß der Teilbereich aus den angegebenen Spaltenelementen lückenlos sein. Jedes Element darf genau einmal vorkommen.

  Für *posnr* geben Sie eine vorzeichenlose Ganzzahl ≥ 1 an.

spalte(*min-max*)
Teilbereich von Spaltenelementen einer multiplen Spalte, die mit Werten belegt werden sollen. Die multiple Spalte muß in der Tabelle enthalten sein. Werden mehrere Elemente einer multiplen Spalte angegeben, muß der Teilbereich aus den angegebenen Spaltenelementen lückenlos sein. Jedes Element darf genau einmal vorkommen.

Für *min* und *max* geben Sie vorzeichenlose Ganzzahlen ≥ 1 an; *max* muß ≥ *min* sein.

sqlausdruck
Ausdruck, dessen Wert der vorangehenden Spalte zugeordnet wird. Der Wert des Ausdrucks muß mit dem Datentyp der Spalte verträglich sein.

Ist *sqlausdruck* eine Variable, kann auch ein Vektor angegeben werden. Die Spalte muß dann eine multiple Spalte sein und die Anzahl der Elemente des Vektors muß mit der Anzahl der Spaltenelemente übereinstimmen.

Ist *sqlausdruck* ein Aggregat (siehe Metavariable *wert*), das einer multiplen Spalte zugewiesen werden soll, so muß die Anzahl der Ausprägungen mit der Anzahl der Spaltenelemente übereinstimmen, und der Datentyp jeder Komponente des Aggregats muß mit dem Datentyp der Zielspalte verträglich sein.

Für *sqlausdruck* gelten folgende Einschränkungen:

– Weder die *tabelle* zugrundeliegende Basistabelle noch ein View auf diese Basistabelle dürfen in der FROM-Klausel einer Unterabfrage stehen, die in *sqlausdruck* vorkommt.

– Mengenfunktionen (AVG, MAX, MIN, SUM, COUNT) sind nicht erlaubt.

DEFAULT
Nur für einfache Spalte.

Die zugehörige Spalte wird mit dem voreingestellten Wert belegt, wenn eine Voreinstellung für die Spalte definiert ist (siehe Metavariable *spaltendefinition*), ansonsten mit dem NULL-Wert.

NULL
Der vorangehenden Spalte wird der NULL-Wert zugewiesen.

WHERE-Klausel
Die WHERE-Klausel gibt an, welche Sätze geändert werden.

WHERE *bedingung* ändert eine Satzmenge (multiple UPDATE-Anweisung), WHERE CURRENT OF *cursor* ändert einen einzelnen Satz.

WHERE-Klausel nicht angegeben:
Alle Sätze der Tabelle werden geändert.

bedingung
Bedingung, die die zu ändernden Sätze erfüllen müssen. Ein Satz wird nur geändert, wenn er die angegebene *bedingung* erfüllt.

Für *bedingung* gelten folgende Einschränkungen:

– Spaltenangaben in *bedingung* außerhalb von Unterabfragen dürfen sich nur auf die angegebene Tabelle beziehen.

– Weder die *tabelle* zugrundeliegende Basistabelle noch ein View auf diese Basistabelle dürfen in der FROM-Klausel einer Unterabfrage stehen, die in *bedingung* vorkommt.

Wenn kein Satz *bedingung* erfüllt, wird kein Satz verändert und ein SQLSTATE gesetzt, der mit WHENEVER &DML_STATE IN ('TABLE END') behandelt werden kann.

CURRENT OF *cursor*

Name des Cursors, über den der zu ändernde Satz bestimmt wird. *tabelle* muß die in der ersten FROM-Klausel der Cursorbeschreibung angegebene Tabelle sein.

Für den Cursor gelten folgende Bedingungen:

– Der Cursor muß sich auf die Tabelle *tabelle* beziehen.

– Der Cursor muß änderbar sein.

– Der Cursor muß in derselben Übersetzungseinheit vereinbart, zum Ausführungszeitpunkt der UPDATE-Anweisung geöffnet und mit FETCH auf einen Satz der Tabelle positioniert sein.

UPDATE ändert den Satz, auf den der Cursor *cursor* zeigt.

UPDATE ist nicht erlaubt, wenn *cursor* ein PREFETCH-Cursor ist.

Wurde der Cursor *cursor* mit der FOR UPDATE-Klausel und Spaltenangaben vereinbart, können nur die dort angegebenen Spalten geändert werden.

Die UPDATE-Anweisung beeinflußt die Position des Cursors nicht. Soll der nächste Satz der Ergebnistabelle geändert werden, müssen Sie den Cursor mit FETCH auf diesen Satz positionieren.

Werte einer multiplen Spalte ändern

Bei einer multiplen Spalte können Werte für einzelne Spaltenelemente sowie für Teilbereiche geändert werden.

Ein Element einer multiplen Spalte wird durch seine Positionsnummer in der multiplen Spalte angesprochen.

Ein Teilbereich einer multiplen Spalte wird durch die Positionsnummern des ersten und letzten Elements des Teilbereichs angesprochen.

Die Position eines Elements innerhalb der multiplen Spalte kann sich ändern. Wenn ein Element mit einer niedrigeren Position auf den NULL-Wert gesetzt wird, werden alle nachfolgenden Elemente nach links verschoben und der NULL-Wert hinten angehängt. Insofern kann ein Unterschied bestehen, ob eine multiple Spalte durch eine UPDATE-Anweisung mit einer Liste von SET-Klauseln oder durch eine Folge von UPDATE-Anweisungen mit einfachen SET-Klauseln geändert wird.

UPDATE und Integritätsbedingungen

Durch Angabe von Integritätsbedingungen bei der Definition der Basistabelle können Sie den möglichen Inhalt von *tabelle* einschränken. Nach allen Änderungen durch die UPDATE-Anweisung muß der Inhalt von *tabelle* den definierten Integritätsbedingungen genügen.

UPDATE und änderbarer persistenter View

Ist bei der Definition eines änderbaren View CHECK OPTION angegeben, können nur Sätze in den View eingefügt werden, die den Abfrageausdruck der Viewdefinition erfüllen.

UPDATE und Transaktionssicherung

UPDATE leitet eine Transaktion ein, wenn keine Transaktion offen ist. Durch die Definition eines Isolationslevels bei konkurrierenden Transaktionen können Sie steuern, welche Auswirkungen die UPDATE-Anweisung auf konkurrierende Transaktionen hat (siehe Anweisung SET TRANSACTION).
Tritt während des Änderungsvorgangs ein Fehler auf, so werden sämtliche bereits durchgeführten Änderungen rückgängig gemacht.

Beispiele

1. Mindestbestand aller Artikel auf 20 erhöhen.

```
UPDATE artikel SET  minbestand = 20
        WHERE minbestand < 20;
```

2. Änderung des Mindestbestands unter Verwendung eines Cursors:

```
DECLARE cur_artikel CURSOR FOR
      SELECT minbestand FROM artikel
      WHERE  minbestand < 20
      FOR UPDATE;
OPEN cur_artikel;
```

Mit einer Folge von FETCH- und UPDATE-Anweisungen können die betroffenen Sätze geändert werden.

```
FETCH cur_artikel INTO &Minbestand;
UPDATE artikel SET minbestand = 20
        WHERE CURRENT OF cur_artikel;
```

3. In der Tabelle farbtab wird für die Farbe orange die Intensität der einzelnen Farbkomponenten geändert. Die Spalte rgb für die Farbintensität ist eine multiple Spalte:

```
UPDATE farbtab SET rgb(1-3) = <0.8, 0.4, 0>
        WHERE farbname  = 'orange';
```

WHENEVER - Fehlerbehandlung definieren

Mit der Anweisung WHENEVER, die nur im Programm-Modus erlaubt und nicht ausführbar ist, werden die Einträge der DRIVE-Systemvariablen &DML_STATE (= &ERROR_STATE.DML_STATE) abgefragt und Fehlerausgänge definiert (siehe auch DRIVE-Lexikon [3], Anweisung WHENEVER, und DRIVE-Programmiersprache [2], Abschnitte „Systemvariablen“ und „Fehler abfangen, Endekriterien“). &DML_STATE ist nach Ausführung einer (statischen oder dynamischen) SQL-Anweisung und nach Ausführung einer CYCLE *cursor* INTO ...-Anweisung sowie der zugehörigen END CYCLE-Anweisung entweder mit dem Eintrag 'OK' oder mit einem spezifischen Fehlerstatus (siehe unten) belegt.

Beim Datenbanksystem SESAM/SQL V2 legt WHENEVER die Reaktion auf die Ausführung dieser Anweisungen fest, falls sie mit einem SQLSTATE <> '00000' (SQL-Anweisung erfolgreich ausgeführt), <> '01SA1' (Satz von fremder, noch offener Transaktion gesperrt) und <> '02000' (Kein Satz gelesen oder geändert) beendet wurden. Der SQLSTATE 00000 ist ein Gutfall, der nicht zu einem Fehlerfall umdefiniert werden kann. Die beiden anderen SQLSTATEs sind voreingestellte Gutfälle, die zu Fehlerfällen umdefiniert werden können (siehe unten).

Die WHENEVER-Anweisung kann mehrmals in einem DRIVE-Programm vorkommen. Sie muß im Deklarationsteil hinter den Definitionen von internen Unterprogrammen stehen (siehe DRIVE-Lexikon [3], Anweisung SUBPROCEDURE). Mit WHENEVER wird spezifiziert, bei welchen (evtl. allen) Fehlersituationen (Ausnahmebedingungen) welche Fehlerausgänge (Ausnahmeaktionen) ausgeführt werden sollen. Werden für eine Fehlersituation, d.h. &DML_STATE wurde mit einem bestimmten Wert <> 'OK' belegt, mehrere WHENEVER angegeben, gilt die letzte Anweisung.

WHENEVER &DML_STATE [IN (*status,...*)]

{ CONTINUE | CALL *subprogname* | BREAK }

&DML_STATE
: DRIVE-Systemvariable vom Typ CHAR(16), die nach Ausführung jeder SQL-Anweisung von DRIVE/WINDOWS mit einem Wert belegt wird.

IN-Klausel

Spezifikation einer Ausnahmebedingung oder einer Liste von Ausnahmebedingungen. Die Ausnahmebedingung &DML_STATE IN (*status*) ist erfüllt, wenn &DML_STATE der Wert *status* zugewiesen wurde. Eine Liste von Ausnahmebedingungen ist erfüllt, wenn &DML_STATE einer der in der Liste angegebenen Werte zugewiesen wurde.

status::= {'TABLE END' | 'DIRTY READ' | 'SQL ERROR' |
'CURSOR SQL ERROR' | 'TOO MANY CURSORS' |
'TEMP SYS ERROR' | 'ACC SYS ERROR' |
'ADMIN SYS ERROR' | 'LIMIT REACHED' }

Wird die IN-Klausel nicht angegeben, ist dies gleichbedeutend mit der Angabe aller möglichen Ausnahmebedingungen.

{ CONTINUE | CALL *subprogname* | BREAK }

Definition des Fehlerausgangs (Ausnahmeaktion) für die spezifizierte(n) Ausnahmebedingung(en).

CONTINUE

Bei Eintritt (einer) der spezifizierten Ausnahmebedingung(en) wird das Programm fortgesetzt.

CALL *subprogname*

Bei Eintritt (einer) der spezifizierten Ausnahmebedingung(en) wird das interne Unterprogramm *subprogname* aufgerufen. Tritt bei Abarbeitung von *subprogname* erneut ein Fehler auf, für den ein Fehlerausgang mit WHENEVER definiert wurde, wird das Programm abgebrochen (keine Fehlerpropagation).

BREAK

Bei Eintritt (einer) der spezifizierten Ausnahmebedingung(en) wird das Programm abgebrochen.

i Für die Ausnahmebedingungen &DML_STATE IN ('TABLE END') und &DML_STATE IN ('DIRTY READ') ist der Fehlerausgang CONTINUE voreingestellt, für alle anderen Ausnahmebedingungen ist der Fehlerausgang BREAK voreingestellt.

Abbildung von SQLCODEs auf &DML_STATE-Einträge

SQLCODEs werden kompatibel zu SESAM/SQL V1 bzw. DRIVE/WINDOWS V1.1 auf &DML_STATEs abgebildet (siehe Kapitel „Fehlermeldungen“ und DRIVE-Programmiersprache [2], Abschnitt „Systemvariablen“).

Abbildung von SQLSTATEs auf &DML_STATE-Einträge

Der SQLSTATE 00000 wird auf &DML_STATE = 'OK' abgebildet.

Der SQLSTATE 02000 wird auf &DML_STATE = 'TABLE END' abgebildet.

Der SQLSTATE 01SA1 wird auf &DML_STATE = 'DIRTY READ' abgebildet.

Der SQLSTATE 24SA5 wird auf &DML_STATE = 'CURSOR SQL ERROR' abgebildet.

Die SQLSTATEs 25SA3 und 25SA5 werden auf &DML_STATE = 'SQL ERROR' abgebildet.

Die SQLSTATEs 91SA3 und 91SA5 werden auf &DML_STATE = 'LIMIT REACHED' abgebildet.

Alle anderen SQLSTATEs werden auf &DML_STATE = 'SQL ERROR' abgebildet, falls nicht &DML_STATE aufgrund eines gleichzeitig gemeldeten SQLCODEs und dessen Abbildung (siehe oben) anders belegt wird.

Der SQLSTATE von SESAM/SQL V2 wird in der DRIVE-Systemvariablen &SQL_STATE (= &ERROR_STATE.SQL_STATE) abgelegt. &SQL_STATE ist eine strukturierte Variable, die den SQLSTATE in die Fehlerklasse &SQL_CLASS (= &SQL_STATE.SQL_CLASS) und die Unterklasse &SUB_CLASS (= &SQL_STATE.SUB_CLASS) zerlegt, die separat ansprechbar sind.

In einem WHENEVER-Unterprogramm *subprogname* zur Ausnahmebedingung &DML_STATE IN ('SQL ERROR') kann daher jeder SQLSTATE einer spezifischen Fehlerbehandlung unterzogen werden. Insbesondere kann bei bestimmten SQL-STATEs die aktuelle SQL-Transaktion zurückgesetzt (ROLLBACK), die DRIVE-Transaktion zurückgesetzt (ROLLBACK WITH RESET), das Programm abgebrochen (BREAK) oder fortgesetzt (END SUBPROC) werden.

Bedeutung der einzelnen &DML_STATE-Einträge

'OK'
SQL-Anweisung erfolgreich ausgeführt.

Die Bedingung &DML_STATE IN ('OK') bzw. = 'OK' ist keine Ausnahmebedingung. Sie kann jedoch ggf. in IF- oder CASE-Anweisungen nach SQL-Anweisungen bzw. in CYCLE-Anweisungen verwendet werden. Für den Produktiveinsatz wird jedoch die Fehlerbehandlung mit WHENEVER-Anweisungen und -Unterprogrammen empfohlen.

'TABLE END'
Kein Satz gelesen oder geändert

Bedeutung
Bei einer FETCH- oder SELECT-Anweisung wurde kein Satz gelesen,
bei einer DELETE-Anweisung wurde kein Satz gelöscht,
bei einer UPDATE-Anweisung wurde kein Satz verändert,
bei einer INSERT-Anweisung wurde kein Satz eingefügt.

Maßnahme
Das DRIVE-Programm muß ggf. geändert werden.

Wurde in einem WHENEVER ein Fehlerausgang <> CONTINUE für 'TABLE END' spezifiziert, so wird dieser nicht ausgeführt, wenn in einer CYCLE *cursor* INTO ...-Anweisung die Fehlersituation 'TABLE END' eintritt.

'DIRTY READ'
Satz von fremder, noch offener Transaktion gesperrt

Bedeutung
Der ausgegebene Satz enthält Daten, die möglicherweise von einer fremden, noch offenen Transaktion geändert wurden. Es besteht die Möglichkeit eines „dirty read"-Phänomens (siehe DRIVE-Programmiersprache [2], Kapitel „Transaktionskonzept"). Diese Situation ist nur möglich bei Konsistenzlevel 0 oder 1, d.h. Isolationslevel READ UNCOMMITTED.

Maßnahme
Die SQL-Anweisung muß ggf. wiederholt werden und/oder es muß ggf. ein anderes Konsistenzlevel definiert werden (siehe Anweisung SET TRANSACTION oder Pragmaklausel ISOLATION LEVEL).

'SQL ERROR'
SQL-Anweisung nicht erfolgreich oder ohne Pragmawirkung ausgeführt

Bedeutung
Die genaue Bedeutung entspricht der von SESAM für den zugehörigen SQLSTATE xxSxx empfohlenen (Anweisung SYSTEM 'HELP SEWxxxx' eingeben).

Maßnahme
Die empfohlene Maßnahme entspricht der von SESAM für den zugehörigen SQLSTATE xxSxx empfohlenen (Anweisung SYSTEM 'HELP SEWxxxx' eingeben).

'CURSOR SQL ERROR'
Cursor ist nicht gespeichert

Bedeutung
Der Cursor wurde nicht mit einer STORE-Anweisung gespeichert oder ist mittlerweile durch ein Rücksetzen der Transaktion entwertet worden.

Maßnahme
DRIVE-Programm ändern.

'TOO MANY CURSORS'
Zulässige DRIVE-Systemgrenze für die Anzahl von dynamischen Cursorvereinbarungen überschritten

Bedeutung
Es sind bereits 20 dynamische oder variable Programm-Cursor oder Dialog-Cursor vereinbart.

Maßnahme
Einige Cursor löschen (siehe Anweisung DROP) oder DRIVE-Programm abbrechen (siehe Anweisung BREAK).

'TEMP SYS ERROR'
kurzfristiges Hindernis beim SESAM-RTS oder SESAM-DBH

Bedeutung
siehe 'SQL ERROR'

Maßnahme
siehe 'SQL ERROR'

'ACC SYS ERROR'
Unverträglichkeiten innerhalb eines SQL-Schemas

Bedeutung
siehe 'SQL ERROR'

Maßnahme
siehe 'SQL ERROR'

'ADMIN SYS ERROR'
SESAM-Administrator-Eingriff erforderlich

Bedeutung
siehe 'SQL ERROR'

Maßnahme
siehe 'SQL ERROR'

'LIMIT REACHED'
SESAM-Systemgrenze erreicht

Bedeutung
siehe 'SQL ERROR'

Maßnahme
siehe 'SQL ERROR'

Bei der Fehlersituation 'LIMIT REACHED' liegt nicht notwendig ein SQLSTATE vor.

4 DRIVE-SQL-Metavariablen

SQL ist vollständig in die DRIVE-Sprache integriert. Dies bedeutet, daß viele DRIVE-Metavariablen in SQL-Anweisungen verwendet werden können (z.B. Bedingungen und Variablen, vgl. *bedingung* und *variable*). Der vom jeweiligen Datenbanksystem unterstützte SQL-Dialekt beinhaltet aber immer Einschränkungen und Erweiterungen gegenüber der SQL-Norm, aus denen sich Unterschiede zu DRIVE-Metavariablen ergeben können. Nur wenn SQL-Objekte angesprochen werden sollen (z.B. Tabellen und Spalten, vgl. *tabellenangabe* und *sqlausdruck*) werden Metavariablen benötigt, die nur in SQL-Anweisungen und nicht in DRIVE-Anweisungen verwendet werden können. In diesem Kapitel werden SQL-Metavariablen und DRIVE-SQL-Metavariablen beschrieben. Die Beschreibung der DRIVE-SQL-Metavariablen beinhaltet den für die Unterstützung von SESAM/SQL V2 relevanten Ausschnitt aus der vollständigen Beschreibung im DRIVE-Lexikon [3].

Die folgenden Metavariablen können nur in SQL-Anweisungen verwendet werden (SQL-Metavariablen):

- *tabelle*
- *spaltendefinition*
- *grunddatentyp* (auch in DRIVE-Anweisungen)
- *voreinstellung*
- *spaltenbedingung*
- *tabellenbedingung*
- *abfrageausdruck*
  - *tabellenangabe*
  - SELECT-Ausdrücke (vgl. *select_ausdruck*)
    - Projektionen (vgl. *select_liste*)
    - FROM-Klauseln
    - WHERE-Klauseln
    - GROUP BY-Klauseln
    - HAVING-Klauseln
  - Join-Ausdrücke (vgl. *join_ausdruck*)
  - Unionen
  - Unterabfragen (vgl. *unterabfrage*).

Die folgenden Metavariablen können in SQL-Anweisungen und ggf. in DRIVE-Anweisungen verwendet werden (DRIVE-SQL-Metavariablen):

- *sqlausdruck*
  - Werte (vgl. *wert*)
    - Literale (vgl. *literal*)
      - alphanumerische Literale
      - numerische Literale
      - Zeitliterale
    - Variabalen (vgl. *variable*)
      - einfache Variablen
      - strukturierte Variablen
      - indizierte Variablen
    - Aggregate
  - drei Zeitfunktionen (vgl. *zeitfunktion*)
  - Spaltenreferenzen „[*tabelle*.] *spalte* [({*posnummer* | *min-max*})]“
  - sechs Mengenfunktionen (vgl. *mengenfunktion*)
  - Unterabfragen (vgl. *unterabfrage*)
  - zwei Userfunktionen (SQL-User und Systemzugang)
  - Operatoren (unär +, -, binär +, -, *, /, ||)
- *bedingung*
- sieben Gruppen von Prädikaten (vgl. *praedikat*)
- Konjunktionen, Disjunktionen, Negationen

Diese Metavariablen stellen eine konsistente Erweiterung der DRIVE-Metavariablen *ausdruck* und *bedingung* gemäß DRIVE-Lexikon [3] dar. Nur wenn in *sqlausdruck* und *bedingung* Spaltenreferenzen, Mengenfunktionen, Unterabfragen oder Userfunktionen vorkommen, handelt es sich um eine DRIVE-SQL-Metavariable, die keine DRIVE-Metavariable ist, also nur in SQL-Anweisungen verwendet werden darf.

DRIVE/WINDOWS V2.1 unterstützt bei *abfrageausdruck* und *sqlausdruck* vollständig die Funktionalität der Version 2.0 von SESAM/SQL, jedoch nicht die Erweiterungen der Version 2.1.

abfrageausdruck

Abfrageausdrücke sind in SESAM/SQL das zentrale Mittel für die Datenabfrage.

Mit einem Abfrageausdruck können Sätze und Spalten aus Basistabellen, Views und temporären Views ausgewählt werden. Die gefundenen Sätze bilden die Ergebnistabelle.

Ein Abfrageausdruck ist ein Teil einer SQL-Anweisung. Ein Abfrageausdruck kann in Unterabfragen und in einer der folgenden SQL-Anweisungen vorkommen:

CREATE TEMPORARY VIEW	Temporären View definieren
CREATE VIEW	View definieren
DECLARE	Cursor vereinbaren
INSERT	Sätze in Tabelle einfügen

Um einen Abfrageausdruck in einer SQL-Anweisung zu verwenden, müssen Sie Eigentümer der mit dem Abfrageausdruck angesprochenen Tabellen sein oder das SELECT-Privileg für diese Tabellen besitzen.

---

abfrageausdruck::= { select_ausdruck | join_ausdruck | (abfrageausdruck) }
[UNION [ALL] abfrageausdruck]

---

select_ausdruck
SELECT-Ausdruck, siehe Metavariable *select_ausdruck*.

join_ausdruck
Join-Ausdruck, siehe Metavariable *join_ausdruck*.

(*abfrageausdruck*)
Unterabfrage, siehe Metavariable *unterabfrage* .

UNION
Die UNION-Klausel verbindet zwei Abfrageausdrücke. Die Ergebnistabelle enthält alle Sätze, die in der ersten oder zweiten Ergebnistabelle vorkommen. Sie können mehr als zwei Ergebnistabellen verbinden, wenn Sie die UNION-Klausel mehrmals verwenden.

Für die Verknüpfung von Abfrageausdrücken mittels UNION müssen folgende Bedingungen erfüllt sein:

– Die Ergebnistabellen der beiden UNION-Operanden müssen dieselbe Anzahl von Spalten haben und die Datentypen entsprechender Spalten müssen verträglich sein. Die Datentypen der Ergebnisspalten werden nach den weiter unten beschriebenen Regeln bestimmt.

Wenn die entsprechenden Spalten der beiden Ausgangstabellen den gleichen Namen haben, erhält die Ergebnisspalte diesen Namen. Ansonsten ist der Name der Ergebnisspalte undefiniert.

– Es dürfen nur einfache Spalten ausgewählt werden.

Mit der UNION-Klausel gebildete Abfrageausdrücke sind nicht änderbar.

ALL
Doppelte Sätze in der Ergebnistabelle bleiben erhalten.

ALL nicht angegeben:
Doppelte Sätze werden entfernt (Duplikatelimination).

Änderbarkeit eines Abfrageausdrucks

Ein Abfrageausdruck ist änderbar, wenn folgende Bedingungen erfüllt sind:

– Der Abfrageausdruck enthält keinen Join-Ausdruck.

– Der Abfrageausdruck enthält keine UNION-Klausel.

– In der *select_liste* dürfen nur Spaltennamen angegeben werden. Andere Elemente von *sqlausdruck*, beispielsweise Unterabfragen, Mengenfunktionen und Zeitfunktionen oder Literale, sind nicht erlaubt. Einfache Spalten dürfen nicht mehrfach angegeben werden. Teilbereiche von multiplen Spalten dürfen sich nicht überlappen.

– In der FROM-Klausel darf nur eine Tabelle angegeben sein oder eine änderbare Unterabfrage. Ist eine Tabelle angegeben, muß sie eine Basistabelle oder ein änderbarer View sein.

– In der WHERE-Klausel darf keine Unterabfrage vorkommen.

– Das Schlüsselwort DISTINCT darf nicht angegeben werden.

– Der SELECT-Ausdruck darf keine GROUP BY- oder HAVING-Klausel enthalten.

Datentyp der Ergebnisspalten bei UNION

Bei der Verknüpfung von zwei Abfrageausdrücken mit UNION ist der Datentyp der Ergebnisspalten durch folgende Regeln festgelegt.

– Beide Ausgangsspalten haben Datentypen CHAR:
Die Ergebnisspalte hat Datentyp CHAR mit der größeren Länge.

– Eine Ausgangsspalte hat Datentyp VARCHAR und die andere Ausgangsspalte CHAR oder VARCHAR:
Die Ergebnisspalte hat Datentyp VARCHAR mit der größeren bzw. größeren maximalen Länge.

- Beide Ausgangsspalten haben ganzzahligen Typ oder Festpunktzahl-Typ (INT, SMALLINT, NUMERIC, DEC):
  Der Datentyp der Ergebnisspalte ist Ganz- oder Festpunktzahl.
  - Die Nachkommastellenzahl ist die größere der beiden Nachkommastellenzahlen der Ausgangsspalten.
  - Die Gesamtstellenzahl ist die größere der beiden Vorkommastellenzahlen plus die größere der beiden Nachkommastellenzahlen der beiden Ausgangsspalten, höchstens jedoch 31.
- Eine Ausgangsspalte hat Gleitpunktzahl-Typ (REAL, DOUBLE, FLOAT), die andere hat einen beliebigen numerischen Datentyp:
  Die Ergebnisspalte hat Datentyp DOUBLE PRECISION.
- Beide Ausgangsspalten haben Zeitdatentyp:
  Beide Spalten müssen denselben Zeitdatentyp haben und die Ergebnisspalte hat auch diesen Datentyp.

bedingung

Bedingungen dienen dazu, die Menge der Sätze einzuschränken, die von einer Tabellenoperation betroffen sind. Es werden nur die Sätze berücksichtigt, die die angegebene *bedingung* erfüllen. Sie können *bedingung* angeben beim Löschen (DELETE), Ändern (UPDATE) und Auswählen von Sätzen (SELECT) und beim Verbinden von Tabellen (Join-Ausdruck). In Tabellen- und Spaltenbedingungen können Sie *bedingung* angeben, um CHECK-Klauseln als Integritätsbedingungen zu formulieren (siehe Metavariablen *spaltenbedingung* und *tabellenbedingung*).

Sie formulieren *bedingung* in einer WHERE-, HAVING-, ON- oder CHECK-Klausel, die in folgenden Anweisungen bzw. Abfrage-Ausdrücken vorkommen können:

- WHERE-Klausel
  - DELETE-Anweisung
  - SELECT-Anweisung
  - SELECT-Ausdruck bei CREATE TEMPORARY VIEW, CREATE VIEW, DECLARE, INSERT
  - UPDATE-Anweisung
- HAVING-Klausel
  - SELECT-Anweisung
  - SELECT-Ausdruck bei CREATE TEMPORARY VIEW, CREATE VIEW, DECLARE, INSERT
- ON-Klausel im Join-Ausdruck
- CHECK-Klausel bei CREATE TABLE und ALTER TABLE

bedingung besteht aus Prädikaten und eventuell logischen Operatoren. Die Prädikate sind die Operanden der logischen Operatoren.

Zur Auswertung von *bedingung* werden die Operatoren auf die Ergebnisse der Operanden angewendet. Das Ergebnis ist einer der Wahrheitswerte **wahr**, **falsch** oder **unbestimmt**.

```
bedingung::= { [ bedingung AND ] { [ NOT ] { praedikat | (bedingung) } } |
        bedingung OR { [ NOT ] { { praedikat | (bedingung) } } }
```

---

praedikat
Spezifikation eines Prädikats (siehe Seite 188).

(*bedingung*)

Geschachtelte Bedingung, um eine Bedingung mit mehr als einem logischen Operator bzw. Prädikat mit Prioritätenangaben zu bilden.

AND

Logisches UND (Konjunktion).

Ergebnis

Op1 AND Op2		**Op2**		
		wahr	**falsch**	**unbestimmt**
Op2	**wahr**	wahr	falsch	unbestimmt
	falsch	falsch	falsch	falsch
	unbestimmt	unbestimmt	falsch	unbestimmt

Tabelle 1: logischer Operator AND

OR

Logisches ODER (Disjunktion).

Ergebnis

Op1 OR Op2		**Op1**		
		wahr	**falsch**	**unbestimmt**
Op2	**wahr**	wahr	wahr	wahr
	falsch	wahr	falsch	unbestimmt
	unbestimmt	wahr	unbestimmt	unbestimmt

Tabelle 2: logischer Operator OR

NOT

Logische Negation.

Ergebnis

		NOT Op
Op	**wahr**	falsch
	falsch	wahr
	unbestimmt	unbestimmt

Tabelle 3: logischer Operator NOT

Prioritäten

– Klammerausdrücke haben die höchste Priorität.

– NOT hat Vorrang vor AND und OR.

– AND hat Vorrang vor OR.

– Operatoren gleicher Priorität werden von links nach rechts angewendet.

Nimmt *bedingung* in einer IF-Anweisung (vgl. DRIVE-Lexikon [3]) den Wahrheitswert **unbestimmt** oder **falsch** an, wird in den ELSE-Pfad verzweigt.

Nimmt *bedingung* in einer CYCLE-Anweisung den Wahrheitswert **unbestimmt** oder **falsch** an, wird die Ausführung der Anweisung beendet.

join_ausdruck

Ein Join ist die Verknüpfung von Daten aus mehreren Tabellen. Eine Tabelle kann auch mit sich selbst verknüpft werden.

Ein Join wird gebildet, indem zwei oder mehrere Tabellen miteinander verknüpft werden. Es gibt zwei Möglichkeiten, einen Join zu formulieren:

– mit einem Join-Ausdruck (insbesondere als Teil einer FROM-Klausel), mit dem

  1. das kartesische Produkt aus allen beteiligten Tabellen gebildet wird;
  2. durch Join-Bedingungen Sätze aus dem kartesischen Produkt der verknüpften Tabellen ausgewählt werden.

     Alle Sätze des kartesischen Produkts, die die Join-Bedingungen erfüllen, werden in die Ergebnistabelle aufgenommen.

– ohne Join-Ausdruck: in einem *select-ausdruck* bzw. in einer SELECT-Anweisung über die FROM-Klausel (enthält beteiligte Tabellen) und gegebenenfalls WHERE-Klausel (enthält Join-Bedingungen).

  Aufgrund der Auswertungsregeln für *select-ausdruck* (siehe Seite 205) ist diese Möglichkeit vom Ergebnis gleichwertig zur Möglichkeit mit Join-Ausdruck.

Ein Join-Ausdruck enthält die zu verknüpfenden Tabellen, die gewünschte Join-Operation und eine Join-Bedingung.

Ein Join-Ausdruck kann angegeben werden:

– als Abfrageausdruck in einer SQL-Anweisung

– in der FROM-Klausel eines *select_ausdruck* oder einer SELECT-Anweisung

– in einer Unterabfrage in *select_liste* und HAVING-Klausel

Die Ergebnistabelle eines Join-Ausdrucks ist nicht änderbar.

Das Ergebnis eines Join-Ausdrucks besteht aus ein, zwei oder drei Teilergebnissen.

Zunächst werden aus dem kartesischen Produkt der beiden Tabellenangaben alle Sätze entfernt, die nicht die in der ON-Klausel formulierte Join-Bedingung erfüllen.

Beim Jointyp INNER bilden die übriggebliebenen Sätze das Ergebnis, andernfalls bilden sie das erste Teilergebnis.

Beim Jointyp LEFT OUTER werden alle nicht berücksichtigten Sätze der ersten Tabellenangabe mit einem fiktiven NULL-Satz der zweiten Tabellenangabe multipliziert, d.h. jeder nicht berücksichtigte Satz der ersten Tabellenangabe wird für jedes Attribut der zweiten Tabellenangabe um einen NULL-Wert ergänzt. Die erste, d.h. linke Tabellenangabe heißt dominante Tabelle.

Beim Jointyp RIGHT OUTER wird ebenso verfahren mit vertauschten Rollen der ersten und zweiten Tabellenangabe. Die zweite, d.h. rechte Tabellenangabe heißt dominante Tabelle. Die so erhaltenen Sätze bilden das zweite Teilergebnis.

Beim Jointyp FULL OUTER werden beide Verfahren durchgeführt und liefern das zweite und dritte Teilergebnis.

Die insgesamt erhaltenen Teilergebnisse werden zum Ergebnis des äußeren Joins vereinigt.

---

join_ausdruck::=

{ tabellenangabe [{ INNER | { LEFT | RIGHT | FULL } [OUTER] }]
JOIN tabellenangabe ON bedingung |
(join_ausdruck) }

---

tabellenangabe
Angabe einer Tabelle, aus der Daten gelesen werden.

INNER

INNER ist Voreinstellung.

INNER-Operator zum Bilden eines Inner-Join. Bei einem Inner-Join enthält die Ergebnistabelle nur die Sätze des kartesischen Produkts, die die Join-Bedingung erfüllen (siehe oben).

{ LEFT | RIGHT | FULL | [OUTER] }
Operatoren zum Bilden eines Outer-Join. Eine Tabelle, die Teil eines Outer-Join ist, darf keine multiplen Spalten enthalten.

Bei einem Outer-Join legen Sie mit der Art des Outer-Join die dominante(n) Tabelle(n) fest (siehe oben).

Erfüllt ein Satz der dominanten Tabelle nicht die Join-Bedingung, wird der Satz trotzdem in die Ergebnistabelle übernommen. Die Ergebnisspalten, die sich auf die andere Tabelle beziehen, werden auf NULL-Werte gesetzt.

LEFT
Die Tabelle links vom LEFT-Operator ist die dominante Tabelle.

RIGHT
Die Tabelle rechts vom RIGHT-Operator ist die dominante Tabelle.

FULL

Die Tabellen links und rechts vom FULL-Operator sind beide dominante Tabellen. FULL erzeugt die Vereinigung der mit LEFT und RIGHT erzeugten Tabellen.

bedingung
Bedingung, die als Join-Bedingung zum Verknüpfen der angegebenen Tabellen verwendet wird (siehe Metavariable *bedingung*).

Für eine in *bedingung* angegebene Spalte gilt:

Die Spalte muß entweder Teil einer der zu verknüpfenden Tabellen sein oder, im Fall von Unterabfragen, Teil einer Tabelle aus einem übergeordneten *select_ausdruck*.

Wenn in *bedingung* eine Mengenfunktion vorkommt, muß eine der beiden Bedingungen gelten:

– Die Mengenfunktion ist in einer Unterabfrage enthalten.

– Der Join-Ausdruck ist in einer *select_liste* oder HAVING-Klausel enthalten und die Spaltenangabe im Argument der Mengenfunktion ist eine Außenreferenz (siehe Metavariable *mengenfunktion*).

(*join_ausdruck*)
Geschachtelter Join-Ausdruck, um einen Join aus mehr als zwei Tabellen mit Prioritätenangaben zu bilden.

literal

Außer für NULL-Werte gibt es für jede Gruppe von Werten entsprechende Literale, die in folgender Übersicht zusammengestellt sind:

---

literal::=

{ charliteral | numliteral | datumzeitliteral }

---

i Die bei SESAM V1 mögliche Angabe eines sedezimalen Literals (X'string'[(n)]) ist bei SESAM V2 nicht möglich. Stattdessen muß ein gewünschter Sedezimalwert einer passenden DRIVE-Variablen zugewiesen und dann diese Variable statt des Literals verwendet werden.

charliteral - Alphanumerische Literale

Die Syntax für ein alphanumerisches Literal ist wie folgt definiert:

---

charliteral::= 'string'

string::= [zeichen] ...

---

string

Beliebige Zeichenkette. *string* muß von Hochkommas (') eingeschlossen werden. Wird ein Hochkomma in *string* verwendet, muß es doppelt angegeben werden. Das verdoppelte Hochkomma wird als ein Zeichen gewertet. *string* kann leer sein und darf höchstens 256 Zeichen enthalten. Zeichenketten der Länge 0 sind also als Literale erlaubt, obwohl es nicht möglich ist, einen Datentyp CHARACTER(0) zu definieren.

zeichen

Beliebiges Zeichen (EBCDI-Code im BS2000, ASCII-Code im SINIX und im MS-Windows).

i Die in DRIVE-Anweisungen mögliche Angabe eines Wiederholungsfaktors ist in SQL-Anweisungen nicht möglich.

numliteral - Numerische Literale

Die Syntax für numerische Literale ist wie folgt definiert:

---

numliteral::= { ganzzahl | festpunktzahl | gleitpunktzahl | $PI }

ganzzahl::= [{ + | - }] vorzeichenlose_ganzzahl

festpunktzahl::= [{ + | - }] vorzeichenlose_ganzzahl [.vorzeichenlose_ganzzahl]

gleitpunktzahl::= festpunktzahl E [{ + | - }] vorzeichenlose_ganzzahl

vorzeichenlose_ganzzahl::= ziffer...

Ganzzahl- und Festpunktzahl-Literale dürfen maximal 31 Ziffern enthalten.

Der Datentyp des Literals ist Ganzzahl, Festpunktzahl oder Gleitpunktzahl mit der angegebenen Anzahl von Vor- und Nachkommastellen.

$PI
: Abkürzung für die Zahl 3,141592653 ... (Umfang eines Kreises geteilt durch seinen Durchmesser).

ziffer
: Dezimalziffer 0 bis 9.

Bei *ganzzahl* ist die Angabe eines Punkts nach *vorzeichenlose_ganzzahl* in DRIVE/WINDOWS nicht erlaubt.

Bei *festpunktzahl* ist die Angabe eines Punkts in DRIVE/WINDOWS nur erlaubt, wenn *vorzeichenlose_ganzzahl* zweimal angegeben wird. In *gleitpunktzahl* dürfen keine Leerstellen vorkommen.

datumzeitliteral - Zeitliterale

Die Syntax für Zeitliterale ist wie folgt definiert:

---

datumzeitliteral::=

{ DATE (jahr-monat-tag) |
TIME (stunde:minute:sekunde [.bruchteil]) |
TIMESTAMP (jahr-monat-tag stunde:minute:sekunde [.bruchteil]) }

DATE
: Datum. Der Datentyp ist DATE.

TIME
: Uhrzeit. Der Datentyp ist TIME oder TIME(3). Der Datentyp TIME, d.h. das Weglassen der Sekundenbruchteile, ist in SQL-Anweisungen nicht erlaubt.

TIMESTAMP
: Zeitstempel. Der Datentyp ist TIMESTAMP(3).

jahr
: Vierstellige vorzeichenlose Ganzzahl zwischen 0001 und 9999, die das Jahr angibt.

monat
: Zweistellige vorzeichenlose Ganzzahl zwischen 01 und 12, die den Monat angibt.

tag
: Zweistellige vorzeichenlose Ganzzahl zwischen 01 und 31 (passend zu Monat und Jahr), die den Tag angibt.

stunde
: Zweistellige vorzeichenlose Ganzzahl zwischen 00 und 23, die die Stunde angibt.

minute
: Zweistellige vorzeichenlose Ganzzahl zwischen 00 und 59, die die Minute angibt.

sekunde
: Zweistellige vorzeichenlose Ganzzahl zwischen 00 und 99, die die Sekunden angibt. Die Angabe von bis zu zwei Schaltsekunden (Werte 60 und 61) ist in DRIVE/WINDOWS nicht erlaubt.

bruchteil
: Dreistellige vorzeichenlose Ganzzahl zwischen 000 und 999, die die Sekundenbruchteile angibt.

Datumsangaben „vor Christus“ sind also nicht möglich.

Eine Datumsangabe muß die Regeln des Gregorianischen Kalenders erfüllen, auch wenn sie ein Datum vor Einführung des Gregorianischen Kalenders bezeichnet. Insbesondere existieren daher die Tage zwischen DATE(1582-10-05) und DATE(1582-10-14) nicht.

i Die Trennzeichen zwischen den Komponentenwerten müssen genau eingehalten werden:

Bindestrich „-“ zwischen Jahr, Monat und Tag
Leerzeichen „ “ zwischen Tag und Stunde
Doppelpunkt „:“ zwischen Stunde, Minute und Sekunde
Punkt „.“ zwischen Sekunde und Sekundenbruchteilen.

mengenfunktion

Mengenfunktionen bestimmen Durchschnitt, Anzahl, Maximum, Minimum und Summe einer Menge von Werten bzw. die Anzahl der Sätze einer Ergebnistabelle.

Mengenfunktionen sind nur in SQL-Anweisungen erlaubt.

```
mengenfunktion::= { { AVG | COUNT | MAX | MIN | SUM }
                    ( [ { ALL | DISTINCT } ] sqlausdruck ) |

                  COUNT(*) }
```

ALL
: ALL ist Voreinstellung.
Alle Werte werden berücksichtigt, auch solche die doppelt vorkommen.

DISTINCT
: Nur verschiedene Werte werden berücksichtigt. Duplikate werden ignoriert.

  Bei den Funktionen MAX() und MIN() hat DISTINCT keine Auswirkung auf das Ergebnis.

sqlausdruck
: Ausdruck, der die Werte in der Menge bestimmt (siehe Metavariable *sqlausdruck*).

  Für jede Mengenfunktion außer COUNT(*) ist festgelegt, welche Datentypen *sqlausdruck* haben darf. In der folgenden Beschreibung jeder einzelnen Funktion sind die erlaubten Datentypen angegeben.

  Für *sqlausdruck* gelten folgende Einschränkungen:

  - *sqlausdruck* darf keine multiple Spalte enthalten.
  - *sqlausdruck* darf keine Mengenfunktion enthalten.
  - *sqlausdruck* darf keine Unterabfrage enthalten.
  - Wenn ein Spaltenname in *sqlausdruck* eine Spalte eines übergeordneten Abfrageausdrucks angibt (sog. **Außenreferenz**), darf *sqlausdruck* nur diesen Spaltennamen enthalten.

Die Mengenfunktion muß dann eine der beiden folgenden Bedingungen erfüllen:

– Die Mengenfunktion ist in einer *select_liste* enthalten.

– Die Mengenfunktion ist in einer Unterabfrage einer HAVING-Klausel enthalten. Der Spaltenname muß eine Spalte des *select_ausdruck* angeben, der die HAVING-Klausel enthält.

AVG() - Arithmetisches Mittel berechnen

AVG() berechnet das arithmetische Mittel aus einer Menge von numerischen Werten. NULL-Werte werden nicht berücksichtigt.

AVG([{ ALL | DISTINCT }] sqlausdruck)

ALL
ALL ist Voreinstellung.
Alle Werte werden berücksichtigt, auch solche die doppelt vorkommen.

DISTINCT
Nur verschiedene Werte werden berücksichtigt. Duplikate werden ignoriert.

sqlausdruck
Numerischer Ausdruck.

Ergebnis

Ohne GROUP BY-Klausel:
Arithmetisches Mittel der Werte in der aus *sqlausdruck* berechneten Menge.

Mit GROUP BY-Klausel:
Pro Gruppe das arithmetische Mittel der Werte für diese Gruppe.

Ist die Menge der aus *sqlausdruck* berechneten Werte leer, ist das Ergebnis bzw. das Ergebnis für diese Gruppe der NULL-Wert.

Datentyp: wie *sqlausdruck* mit folgender Stellenzahl:

– Ganzzahl oder Festpunktzahl:

  Die Gesamtstellenzahl ist 31, die Nachkommastellenzahl ist 31-g+n.
  g und n sind die Gesamtstellenzahl und Nachkommastellenzahl von *sqlausdruck*.

– Gleitpunktzahl:

  Die Gesamtstellenzahl entspricht 21 Binärstellen bei REAL und 53 Binärstellen bei DOUBLE PRECISION.

Beispiele

1. SELECT ohne GROUP BY:
   Durchschnittssatz der Leistungen in der Tabelle leistung berechnen:

   ```
   SELECT AVG(lsatz) FROM leistung;
   ```

   ```
   783.33
   ```

   Wenn Sie in die Tabelle einen Satz eintragen, der in der Spalte lsatz den NULL-Wert enthält, ändert sich das Ergebnis nicht.

2. SELECT mit GROUP BY:
   Für jede Auftragsnummer wird der Durchschnittssatz berechnet:

   ```
   DECLARE ... CURSOR FOR SELECT anr, AVG(lsatz)
   FROM leistung
   GROUP BY anr;
   ```

   ```
   anr
   200     1026
   211      662.5
   250      662.5
   ```

COUNT(*) - Tabellensätze zählen

COUNT(*) zählt die Sätze einer Tabelle. Sätze, die NULL-Werte enthalten, werden mitgezählt. COUNT(*) ist nur in der *select_liste* eines *select_ausdruck* erlaubt (siehe Metavariable *select_ausdruck*).

COUNT(*)

Ergebnis

Ohne GROUP BY-Klausel:
Anzahl der Sätze der Ergebnistabelle des zugehörigen *select_ausdruck* (bzw. der zugehörigen SELECT-Anweisung). Doppelte Sätze und Sätze, die nur NULL-Werte enthalten, werden mitgezählt.

Mit GROUP BY-Klausel:
Pro Gruppe in der Ergebnistabelle die Anzahl der Sätze in dieser Gruppe.

Datentyp: numerisch (Ganzzahl) mit Stellenzahl 31.

Beispiele

1. SELECT ohne GROUP BY:
   Aus der Tabelle kunde abfragen, wie viele Kunden in München wohnen:

```
SELECT COUNT(*) FROM kunde WHERE ort='Muenchen';
```

```
3
```

2. SELECT mit GROUP BY:
   Die Kunden getrennt nach Orten zählen:

```
DECLARE ... CURSOR FOR SELECT ort, COUNT(*)
FROM kunde
GROUP BY ort;
```

```
ort
Berlin              1
Bern 33             1
Hannover            1
Moenchengladbach    1
Muenchen            3
New York, NY        1
```

COUNT() - Elemente zählen

COUNT() zählt die Elemente einer Menge von Werten. NULL-Werte werden nicht mitgezählt.

COUNT([{ ALL | DISTINCT }] sqlausdruck)

ALL
ALL ist Voreinstellung.
Alle Werte werden berücksichtigt, auch solche, die doppelt vorkommen.

DISTINCT
Nur verschiedene Werte werden berücksichtigt. Duplikate werden ignoriert.

sqlausdruck
Numerischer, alphanumerischer oder Zeitwert-Ausdruck.

Ergebnis

Ohne GROUP BY-Klausel:
Anzahl der Werte in der aus *sqlausdruck* berechneten Menge.

Mit GROUP BY-Klausel:
Pro Gruppe die Anzahl der Werte für diese Gruppe.

Datentyp: numerisch (Ganzzahl) mit Stellenzahl 31.

Beispiele

1. SELECT ohne GROUP BY:
   Aus der Tabelle leistung die Anzahl verschiedener Leistungsbeschreibungen bestimmen:

```
SELECT COUNT(DISTINCT ltext) FROM leistung;
```

```
7
```

2. SELECT mit GROUP BY:
   Für jede Auftragsnummer die Anzahl verschiedener Leistungen zählen:

```
DECLARE ... CURSOR FOR SELECT anr, COUNT(DISTINCT ltext)
FROM leistung
GROUP BY anr;
```

```
anr
200         2
211         4
260         2
```

MAX() - Maximum bestimmen

MAX() bestimmt den größten Wert einer Menge von Werten. NULL-Werte werden nicht berücksichtigt.

Der Vergleich von Werten (mit vergleichbarem Datentyp) ist bei der Metavariablen *bedingung* beschrieben.

```
MAX( [ { ALL | DISTINCT } ] sqlausdruck )
```

ALL ist Voreinstellung.

DISTINCT
: Die Angabe DISTINCT ist syntaktisch erlaubt, hat aber keine Auswirkung auf das Ergebnis.

sqlausdruck
: Numerischer Ausdruck, alphanumerischer Ausdruck oder Zeitwert-Ausdruck.

Ergebnis

Ohne GROUP BY-Klausel:
: Größter Wert in der aus *ausdruck* berechneten Menge.

Mit GROUP BY-Klausel:
: Pro Gruppe der größte Wert für diese Gruppe.

Ist die Menge der aus *sqlausdruck* berechneten Werte leer, ist das Ergebnis bzw. das Ergebnis für diese Gruppe der NULL-Wert.

Datentyp: wie *sqlausdruck*.

Beispiele

1. SELECT ohne GROUP BY:
   Aus der Tabelle leistung den höchsten Leistungssatz für Auftrag 211 abfragen:

   ```
   SELECT MAX(lsatz) FROM leistung WHERE anr=211;
   ```

   ```
   1200
   ```

2. DECLARE ... CURSOR FOR SELECT mit GROUP BY:
   Für jede Auftragsnummer den höchsten Leistungssatz bestimmen:

```
DECLARE... CURSOR FOR SELECT anr, MAX(lsatz)
FROM leistung
GROUP BY anr;
```

```
anr
200     1600
211     1200
250     1200
```

MIN() - Minimum bestimmen

MIN() bestimmt das kleinste Element einer Menge von Werten. NULL-Werte werden nicht berücksichtigt.

Der Vergleich von Werten (mit vergleichbarem Datentyp) ist bei der Metavariablen *bedingung* beschrieben.

MIN([{ ALL | DISTINCT }] sqlausdruck)

ALL ist Voreinstellung.

DISTINCT
Die Angabe DISTINCT ist syntaktisch erlaubt, hat aber keine Auswirkung auf das Ergebnis.

sqlausdruck
Numerischer Ausdruck, alphanumerischer Ausdruck oder Zeitwert-Ausdruck.

Ergebnis

Ohne GROUP BY-Klausel:
Kleinster Wert in der aus *sqlausdruck* berechneten Menge.

Mit GROUP BY-Klausel:
Pro Gruppe der kleinste Wert für diese Gruppe.

Ist die Menge der aus *sqlausdruck* berechneten Werte leer, ist das Ergebnis bzw. das Ergebnis für diese Gruppe der NULL-Wert.

Datentyp: wie *sqlausdruck*.

Beispiele

1. SELECT ohne GROUP BY:
   Aus der Tabelle leistung den niedrigsten Leistungssatz für Auftrag 211 abfragen:

```
SELECT MIN(lsatz) FROM leistung WHERE anr=211;  Š
```

```
50
```

2. SELECT mit GROUP BY:
   Für jede Auftragsnummer den niedrigsten Leistungssatz bestimmen:

```
DECLARE ... CURSOR FOR SELECT anr, MIN(lsatz)
FROM leistung
GROUP BY anr;
```

```
anr
200      75
211      50
250     125
```

SUM() - Summe berechnen

SUM() berechnet die Summe aller Werte einer Menge. NULL-Werte werden nicht berücksichtigt.

SUM([{ ALL | DISTINCT }] sqlausdruck)

ALL
ALL ist Voreinstellung.
Alle Werte werden berücksichtigt, auch solche, die doppelt vorkommen.

DISTINCT
Nur verschiedene Werte werden berücksichtigt. Duplikate werden ignoriert.

sqlausdruck
Numerischer Ausdruck.

Ergebnis

Ohne GROUP BY-Klausel:
Summe der Werte in der aus *sqlausdruck* berechneten Menge.

Mit GROUP BY-Klausel:
Pro Gruppe die Summe der Werte für diese Gruppe.

Ist die Menge der aus *sqlausdruck* berechneten Werte leer, ist das Ergebnis bzw. das Ergebnis für diese Gruppe der NULL-Wert.

Datentyp: wie *sqlausdruck* mit folgender Stellenzahl:

– Ganzzahl oder Festpunktzahl:
Die Gesamtstellenzahl ist 31, die Nachkommastellenzahl bleibt gleich.

– Gleitpunktzahl:
Die Gesamtstellenzahl entspricht 21 Binärstellen bei REAL und 53 Binärstellen bei DOUBLE PRECISION.

Ist die Summe der Werte für diesen Datentyp zu groß, erfolgt eine SESAM-Fehlermeldung.

Beispiel

Aus der Tabelle verwendung für jede Artikelnummer die Summe der Bestandteile berechnen:

```
DECLARE ... CURSOR FOR SELECT artnr, SUM(anzahl)
FROM verwendung
GROUP BY artnr;

...
```

```
artnr
  1       4
120       27
200       20
```

praedikat - Prädikat spezifzieren

praedikat besteht aus Operanden und Operatoren. Entsprechend den Operatoren ist *praedikat* in folgende Gruppen unterteilt:

– Vergleich von zwei Werten

– Vergleich mit einer Ergebnisspalte (nur in SQL-Anweisungen möglich)

– Bereichsabfrage

– Elementabfrage

– Mustervergleich (nur in SQL-Anweisungen möglich)

– Vergleich auf NULL-Wert

– Existenzabfrage (nur in SQL-Anweisungen möglich)

praedikat liefert den Wahrheitswert **wahr**, **falsch** oder **unbestimmt**. Der Wert von *praedikat* wird berechnet, indem die Werte der Operanden berechnet werden und die jeweiligen Operatoren auf die berechneten Werte angewendet werden.

In der folgenden Übersicht ist die Syntax aller Gruppen von *praedikat* zusammengestellt:

```
praedikat::=

{ sqlausdruck { = | < | > | <= | >= | <> } sqlausdruck |

 sqlausdruck { = | < | > | <= | >= | <> } { ANY | SOME | ALL } unterabfrage |

 sqlausdruck [ NOT ] BETWEEN sqlausdruck AND sqlausdruck |

 sqlausdruck [ NOT ] IN { unterabfrage | (sqlausdruck,sqlausdruck,...) } |

 [ tabelle . ] { spalte | spalte(posnummer) | spalte(min-max) } [ NOT ] LIKE
          muster [ ESCAPE zeichen ] |

 [ tabelle . ] { spalte | spalte(posnummer) | spalte(min-max) }
          IS [ NOT ] NULL |

 EXISTS unterabfrage }
```

Die Prädikate sind in der Reihenfolge beschrieben, in der sie in der Übersicht aufgelistet sind:

– Vergleich von zwei Werten
– Vergleich mit einer Ergebnisspalte
– Bereichsabfrage
– Elementabfrage
– Mustervergleich
– Vergleich auf NULL-Wert
– Existenzabfrage

Vergleich von zwei Werten

Zwei Ausdrücke mit vergleichbarem Datentyp werden gemäß dem angegebenen Vergleichsoperator verglichen.

---

sqlausdruck1 { = | < | > | <= | >= | <> } sqlausdruck2

---

sqlausdruck1, *sqlausdruck2*
Operanden für den Vergleich.

Die Werte von *sqlausdruck1* und *sqlausdruck2* müssen entweder einfach sein oder Namen von multiplen Spalten. Ist ein Operand eine multiple Spalte, so darf diese Spaltenangabe keine Außenreferenz sein (siehe Metavariable *mengenfunktion*).

Ist *sqlausdruck1* einfach, müssen *sqlausdruck1* und *sqlausdruck2* vom gleichen *grunddatentyp* sein.

Ist *sqlausdruck1* strukturiert, gelten nur die Vergleichsoperatoren gleich = und ungleich <>. Ausnahme: multiple Spalten.

Vergleichsoperatoren

=	Vergleich auf gleich
<	Vergleich auf kleiner
>	Vergleich auf größer
<=	Vergleich auf kleiner oder gleich
>=	Vergleich auf größer oder gleich
<>	Vergleich auf ungleich

Ergebnis

sqlausdruck1 einfach:
Unbestimmt, wenn mindestens ein Operand den NULL-Wert ergibt.

Wahr, wenn beide Operanden Nicht-NULL-Werte sind und der Vergleich zutrifft.

Falsch, sonst.

sqlausdruck1 multiple Spalte:
Jede Ausprägung von *sqlausdruck1* wird mit *sqlausdruck2* verglichen. Die Vergleichsergebnisse werden mit OR verknüpft.

Beispiel

Wenn X eine multiple Spalte mit 3 Elementen ist, ist der Vergleich

```
X(1-3) >= 13
```

zu folgenden Vergleichen äquivalent:

```
X(1) >= 13 OR
X(2) >= 13 OR
X(3) >= 13
```

Vergleichsregeln

Wie eine Vergleichsoperation ausgeführt wird, ist abhängig vom Datentyp der Operanden. Die Vergleichsregeln sind im folgenden zusammengestellt.

NULL-Werte

Ist ein Operand der NULL-Wert, liefern alle Vergleiche als Ergebnis den Wahrheitswert **unbestimmt**.

Alphanumerische Werte

Zwei alphanumerische Werte werden von links nach rechts Zeichen für Zeichen verglichen. Sind die beiden Werte unterschiedlich lang, wird die kürzere Zeichenkette rechts mit Leerzeichen aufgefüllt, so daß beide gleich lang sind.

Zwei Zeichenketten sind gleich, wenn sie an jeder Position das gleiche Zeichen haben.

Wenn zwei Zeichenketten nicht gleich sind, bestimmt der EBCDI-Code der ersten beiden unterschiedlichen Zeichen, welche Zeichenkette größer bzw. kleiner ist. Dies gilt uneingeschränkt im BS2000. Auf den Plattformen MS-Windows und SINIX gilt dies innerhalb von SQL-Anweisungen, weil dann der Vergleich im BS2000 von SESAM durchgeführt wird. Innerhalb von DRIVE-Anweisungen dagegen wird der Vergleich von DRIVE/WINDOWS auf der Clientplattform durchgeführt, so daß der ASCII-Code das Vergleichsergebnis bestimmt.

Numerische Werte

Zwei numerische Werte sind gleich, wenn sie beide 0 sind oder dasselbe Vorzeichen und denselben Betrag haben.

Zeitwerte

Datum, Zeit und Zeitstempel können verglichen werden. Der Datentyp der beiden Operanden muß jedoch identisch sein.

– Ein Datum ist größer als ein anderes, wenn es jünger ist.

– Eine Uhrzeit ist größer als eine andere, wenn sie eine spätere Zeit angibt.

– Ein Zeitstempel ist größer als ein anderer, wenn entweder das Datum jünger ist oder, bei gleichem Datum, die Uhrzeit größer ist.

Beispiele

1. Alphanumerische Werte vergleichen:
   Aus der Tabelle kunde die Kunden mit Kundeninformation heraussuchen, die aus München kommen:

```
DECLARE ... CURSOR FOR SELECT firma, kundeninfo, ort
   FROM kunde
   WHERE ort = 'Muenchen';

...
```

firma	kundeninfo	ort
Siemens AG	Elektro	Muenchen
Login GmbH	PC Netzwerke	Muenchen
Plenzer Trading	Fruechtehandel	Muenchen

2. Vergleich mit Unterabfrage, die genau einen Wert liefert:
   Aus der Tabelle verwendung den Artikel heraussuchen, für den die größte Menge des Bestandteils 501 benötigt wird.

```
SELECT artnr
   FROM verwendung
   WHERE bestandteil = 501
   AND
   anzahl = (SELECT MAX(anzahl)
         FROM verwendung
         WHERE bestandteil = 501);
```

   Da das Maximum immer ein eindeutiger Wert ist, liefert die Unterabfrage genau einen Wert und darf im Vergleich als Operand angegeben werden.

artnr
200

Vergleich mit einer Ergebnisspalte

sqlausdruck wird mit den Werten einer Ergebnisspalte verglichen, die das Ergebnis einer Unterabfrage ist.

Dieses Prädikat ist nur innerhalb von SQL-Anweisungen möglich.

---

sqlausdruck { = | < | > | <= | >= | <> } { ANY | SOME | ALL } unterabfrage

---

sqlausdruck
Operand für den Vergleich.

Der Wert von *sqlausdruck* muß einfach sein.

Vergleichsoperatoren

= Vergleich auf gleich

< Vergleich auf kleiner

> Vergleich auf größer

<= Vergleich auf kleiner oder gleich

>= Vergleich auf größer oder gleich

<> Vergleich auf ungleich

unterabfrage
Unterabfrage, die eine einspaltige Tabelle liefert.

sqlausdruck und die Ergebniswerte der Unterabfrage müssen verträgliche Datentypen haben.

Ergebnis

ANY
SOME
Wahr, wenn der Vergleich mit mindestens einem Wert der Ergebnisspalte **wahr** ergibt.

Falsch, wenn die Ergebnisspalte leer ist, oder der Vergleich mit allen Werten der Ergebnisspalte **falsch** ergibt.

Unbestimmt, sonst.

ALL
Wahr, wenn die Ergebnisspalte leer ist, oder der Vergleich mit allen Werten der Ergebnisspalte **wahr** ergibt.

Falsch, wenn der Vergleich mit mindestens einem Wert **falsch** ergibt.

Unbestimmt, sonst.

Beispiel

Aus der Tabelle verwendung die Artikel heraussuchen, die einen Bestandteil enthalten, dessen Anzahl höher ist als die Anzahl aller Bestandteile des Artikels mit der Artikelnummer 1.

```
DECLARE ... CURSOR FOR SELECT artnr
    FROM verwendung
    WHERE anzahl
    > ALL (SELECT anzahl
        FROM verwendung
        WHERE artnr = 1);
```

```
artnr
 120
 200
```

Bereichsabfrage

Es wird geprüft, ob der Wert des ersten *sqlausdruck* in dem durch den zweiten und dritten *sqlausdruck* angegebenen Wertebereich liegt.

‘‘

sqlausdruck1 [NOT] BETWEEN sqlausdruck2 AND sqlausdruck3

---

sqlausdruck1, *sqlausdruck2*, *sqlausdruck3*
Operanden für den Vergleich.

Der Wert des ersten *sqlausdruck* muß einfach sein oder der Name einer multiplen Spalte. Ist *sqlausdruck* eine multiple Spalte, darf die Spaltenangabe keine Außenreferenz sein (siehe Metavariable *mengenfunktion*).

Die Werte des zweiten und dritten *sqlausdruck* müssen einfach sein.

Die Operanden müssen verträgliche Datentypen haben.

Ergebnis

sqlausdruck1 einfach:

Ohne NOT:
identisch mit:
(*sqlausdruck1* >= *sqlausdruck2*) AND (*sqlausdruck1* <= *sqlausdruck3*)

Mit NOT:
identisch mit:
NOT (*sqlausdruck1* BETWEEN *sqlausdruck2* AND *sqlausdruck3*)

sqlausdruck1 multiple Spalte:

– Die Bereichsabfrage wird für jede Ausprägung von *sqlausdruck1* durchgeführt.
– Die Einzelergebnisse werden mit OR verknüpft.

Beispiel

Wenn X eine multiple Spalte mit 3 Elementen ist, ist die Bereichsabfrage

X(1-3) BETWEEN 13 AND 20

zu folgenden Bereichsabfragen äquivalent:

X(1) BETWEEN 13 AND 20 OR
X(2) BETWEEN 13 AND 20 OR
X(3) BETWEEN 13 AND 20

Beispiele

1. Numerischen Bereich abprüfen:
   Aus der Tabelle artikel alle Artikel mit Artikelbezeichnung heraussuchen, deren Preis zwischen 50.- und 100.- DM liegt.

```
SELECT artnr, artbez, preis
  FROM artikel
  WHERE preis BETWEEN 50.00 AND 100.00;
```

```
artnr  artbez      preis
 200  Lenkstange  60.00
```

2. Datumsbereich abprüfen:
   Aus der Tabelle auftrag die Aufträge mit Auftragsnummer, Kundennummer, Auftragsdatum und Auftragstext heraussuchen, die im Dezember 1990 gestellt wurden:

```
DECLARE ... CURSOR FOR SELECT anr, knr, atext, adatum
  FROM auftrag
  WHERE adatum
     BETWEEN DATE(1990-12-01) AND DATE(1990-12-31);

...
```

```
anr  knr  atext                  adatum
210  106  Kunden-Verwaltung       1990-12-13
211  106  Datenbankentwurf KUNDEN  1990-12-29
```

Elementabfrage

Es wird geprüft, ob ein Wert unter den Elementen einer Menge vorkommt.

---

sqlausdruck1 [NOT] IN { unterabfrage | (sqlausdruck2,sqlausdruck3,...) }

---

sqlausdruck1
Ausdruck für den Vergleich.

Der Wert von *sqlausdruck1*muß entweder einfach sein oder der Name einer multiplen Spalte. Ist *sqlausdruck1* eine multiple Spalte, darf die Spaltenangabe keine Außenreferenz sein (siehe Metavariable *mengenfunktion*), und als zweiten Operanden dürfen Sie keine Unterabfrage angeben.

unterabfrage
Unterabfrage, die eine einspaltige Tabelle liefert.

sqlausdruck1 und die folgenden Ausdrücke *sqlausdruck2*, *sqlausdruck3*,... bzw. die Ergebniswerte der Unterabfrage müssen verträgliche Datentypen haben.

(*sqlausdruck*2, *sqlausdruck3*,....**)** darf keine multiplen Ausdrücke enthalten.

Ergebnis

sqlausdruck1 einfach:

Ohne NOT:
Wahr, wenn der Vergleich auf gleich mit mindestens einem Ausdruck bzw. einem Ergebniswert der Unterabfrage **wahr** ergibt.

Falsch, wenn der Vergleich mit allen Ausdrücken bzw. allen Ergebniswerten der Unterabfrage **falsch** ergibt, oder wenn die Ergebnisspalte der Unterabfrage leer ist.

Unbestimmt, sonst.

Mit NOT:
identisch mit:

NOT (*sqlausdruck* IN *unterabfrage*) bzw. NOT (*sqlausdruck1* IN **(***sqlausdruck2*,...**)**)

sqlausdruck1 multiple Spalte:

– Die Elementabfrage wird für jede Ausprägung von *sqlausdruck1* durchgeführt.
– Die Einzelergebnisse werden mit OR verknüpft.

Beispiel

Wenn X eine multiple Spalte mit 3 Elementen ist, ist die Elementabfrage

```
X(1-3) IN (13, 20, 30)
```

zu folgenden Elementabfragen äquivalent:

```
X(1) IN (13, 20, 30) OR
X(2) IN (13, 20, 30) OR
X(3) IN (13, 20, 30)
```

Beispiele

1. Elementabfrage mit alphanumerischen Werten:
   Aus der Tabelle kunde alle Kunden mit Kundeninformation heraussuchen, die aus München oder Berlin sind:

```
DECLARE ... CURSOR FOR SELECT firma, kinfo, ort
    FROM kunde
    WHERE ort IN ('Muenchen','Berlin');

...
```

```
firma           kinfo          ort
Siemens AG      Elektro        Muenchen
Login GmbH      PC-Netzwerke   Muenchen
Plenzer Trading Fruechtehandel Muenchen
Jonas Fischladen Einzelhandel  Berlin
```

2. Elementabfrage mit Ergebnisspalte:
   Aus den Tabellen auftrag und leistung die Aufträge heraussuchen, bei denen keine Schulung durchgeführt wurde.

```
SELECT knr
  FROM auftrag
  WHERE anr
  NOT IN (SELECT anr
      FROM leistung
      WHERE ltext = 'Schulung');
```

```
knr
106
```

Mustervergleich

Es wird geprüft, ob ein alphanumerischer Wert zu einem angegebenen Muster paßt. Ein Muster ist eine Zeichenkette, die außer normalen Zeichen auch Platzhalter und Entwertungszeichen enthalten kann.

Ein Platzhalter steht für ein oder mehrere andere Zeichen. Platzhalter können auch als normales Zeichen im Muster erscheinen, wenn sie mit dem Entwertungszeichen entwertet werden. Das Entwertungszeichen können Sie mit der ESCAPE-Klausel definieren.

Dieses Prädikat ist nur innerhalb von SQL-Anweisungen erlaubt.

```
[ tabelle . ] { spalte | spalte(posnummer) | spalte(min-max) }
                [ NOT ] LIKE  muster [ESCAPE zeichen ]

  muster::= wert

  zeichen::= wert
```

---

tabelle
Name einer Tabelle, die *spalte* enthält. Statt des Tabellennamens geben Sie den Korrelationsnamen an, wenn für die Tabelle ein Korrelationsname definiert ist.

spalte
Name einer Spalte, aus der die Werte genommen werden. Der Datentyp der Spalte muß alphanumerisch sein.

posnummer
Vorzeichenlose Ganzzahl.

spalte ist eine multiple Spalte. *spalte* darf nicht Spalte eines übergeordneten Abfrageausdrucks sein.

Der Wert ist der Wert des (*posnummer*-sp_{min}+1)-ten Elements von *spalte*.

Ist *spalte* keine multiple Spalte, *posnummer* kleiner als sp_{min} oder *posnummer* größer als sp_{max}, erfolgt eine SESAM-Fehlermeldung.

sp_{min} bzw. sp_{max} ist die kleinste bzw. größte Positionsnummer der multiplen Spalte.

min-*max*
Vorzeichenlose Ganzzahlen.

spalte ist eine multiple Spalte. *spalte* darf nicht Spalte eines übergeordneten Abfrageausdrucks sein.

Der Wert ist das Aggregat der Spaltenelemente (*min*-sp_{min}+1) bis (*max*-sp_{min}+1).

Ist *spalte* keine multiple Spalte, *min* nicht kleiner als *max*, *min* kleiner als sp_{min} oder *max* größer als sp_{max}, erfolgt eine SESAM-Fehlermeldung.

sp_{min} bzw. sp_{max} ist die kleinste bzw. größte Positionsnummer der multiplen Spalte.

posnummer bzw. *min*-*max* nicht angegeben:
spalte darf keine multiple Spalte sein.

muster
Alphanumerischer Wert, zu dem der Wert aus *spalte* passen soll. *muster* darf enthalten:

– normale Zeichen (d.h. ohne Platzhalter und Entwertungszeichen)

– Platzhalter

Platzhalter	**Bedeutung**
_ (Unterstrich)	ein beliebiges Zeichen
%	beliebige (auch leere) Folge von Zeichen

Tabelle 4: Platzhalter im Mustervergleich

– Entwertungszeichen (gefolgt von Platzhalter oder Entwertungszeichen)

Leerzeichen in *muster*, auch am Anfang oder Ende, gehören zum Muster.

ESCAPE-Klausel
Mit der ESCAPE-Klausel können Sie ein Entwertungszeichen definieren. Entwertungszeichen vor Platzhaltern bewirken, daß die Platzhalter ihre Funktion als Platzhalter verlieren und statt dessen als normale Zeichen interpretiert werden. Mit dem Entwertungszeichen können Sie auch das Entwertungszeichen entwerten und als normales Zeichen verwenden.

zeichen
Alphanumerischer Wert der Länge 1.
zeichen gilt in diesem Vergleich als Entwertungszeichen.

ESCAPE-Klausel nicht angegeben:
Es ist kein Entwertungszeichen definiert.

Ergebnis

spalte einfach:

Unbestimmt, wenn der Wert von *spalte*, *muster* oder *zeichen* der NULL-Wert ist, sonst:

Ohne NOT:

Wahr, wenn es eine Belegung der Platzhalter in *muster* gibt, sodaß der entstehende Wert gleich dem Wert von *spalte* ist und die gleiche Länge hat.

Falsch, sonst.

Mit NOT:

Wahr, wenn es keine Belegung der Platzhalter in *muster* gibt, so daß der entstehende Wert gleich dem Wert von *spalte* ist und die gleiche Länge hat.

Falsch, sonst.

spalte multiple Spalte:

– Der Mustervergleich wird für jede Ausprägung von *spalte* durchgeführt.

– Die Einzelergebnisse werden mit OR verknüpft.

Beispiele

1. Aus der Tabelle kontakt alle Kontaktpersonen heraussuchen, deren Vorname mit Ro beginnt:

```
DECLARE ... CURSOR FOR SELECT vorname, nachname
  FROM kontakt
  WHERE vorname LIKE 'Ro%';

...
```

```
vorname    nachname
Roland     Loetzerich
Robert     Heinlein
```

2. Die folgende Anweisung sucht aus einer alphanumerischen Spalte sp einer Tabelle tab alle Zeichenfolgen, die mit Unterstrich beginnen und mit mindestens einem Leerzeichen enden:

```
SELECT * FROM tab
  WHERE sp LIKE '@_% ' ESCAPE '@'
```

Vergleich auf NULL-Wert

Es wird geprüft, ob eine Spalte den NULL-Wert enthält.

[tabelle .] { spalte | spalte(posnummer) | spalte(min-max) }
IS [NOT] NULL

---

tabelle
Name einer Tabelle, die *spalte* enthält. Statt des Tabellennamens geben Sie den Korrelationsnamen an, wenn für die Tabelle ein Korrelationsname definiert ist.

spalte
Name einer Spalte, aus der die Werte genommen werden.

posnummer
Vorzeichenlose Ganzzahl.

spalte ist eine multiple Spalte. *spalte* darf nicht Spalte eines übergeordneten Abfrageausdrucks sein.

Der Wert ist der Wert des (*posnummer*-sp_{min}+1)-ten Elements von *spalte*.

Ist *spalte* keine multiple Spalte, *posnummer* kleiner als sp_{min} oder *posnummer* größer als sp_{max}, erfolgt eine SESAM-Fehlermeldung.

sp_{min} bzw. sp_{max} ist die kleinste bzw. größte Positionsnummer der multiplen Spalte.

min-*max*
Vorzeichenlose Ganzzahlen.

spalte ist eine multiple Spalte. *spalte* darf nicht Spalte eines übergeordneten Abfrageausdrucks sein.

Der Wert ist das Aggregat der Spaltenelemente (*min*-sp_{min}+1) bis (*max*-sp_{min}+1).

Ist *spalte* keine multiple Spalte, *min* nicht kleiner als *max*, *min* kleiner als sp_{min} oder *max* größer als sp_{max}, erfolgt eine SESAM-Fehlermeldung.

sp_{min} bzw. sp_{max} ist die kleinste bzw. größte Positionsnummer der multiplen Spalte.

posnummer bzw. *min*-*max* nicht angegeben:
spalte darf keine multiple Spalte sein.

Ergebnis

spalte einfach:

Ohne NOT:
Wahr, wenn der Wert von *spalte* der NULL-Wert ist.

Falsch, sonst.

Mit NOT:
Wahr, wenn der Wert von *spalte* nicht der NULL-Wert ist.

Falsch, sonst.

spalte multiple Spalte:

Ohne NOT:
Wahr, wenn mindestens eine Ausprägung von *spalte* der NULL-Wert ist.

Falsch, sonst.

Mit NOT:
Wahr, wenn mindestens eine Ausprägung von *spalte* nicht der NULL-Wert ist.

Falsch, sonst.

Beispiel

Aus der Tabelle auftrag die Aufträge mit Auftragstext und Soll-Termin heraussuchen, die noch nicht fertiggestellt sind, d.h. für die der Ist-Termin der NULL-Wert ist.

```
DECLARE ... CURSOR FOR SELECT anr, atext, fertigsoll
    FROM auftrag
    WHERE fertigist IS NULL;

...
```

```
anr  atext                fertigsoll
250  Serienbrief-Einweisung  1991-03-01
251  Kunden-Verwaltung       1991-05-01
300  Netzwerk-Test/Vergleich
305  Mitarbeiterschulung     1991-02-27
```

Die angegebene Form des Prädikats ist nur in SQL-Anweisungen möglich. In DRIVE-Anweisungen kann aber statt der Spaltenreferenz ein *wert* angegeben werden (siehe DRIVE-Lexikon [3], Metavariable *ausdruck*).

Existenzabfrage

Es wird geprüft, ob eine Ergebnistabelle leer ist.

Dieses Prädikat ist nur innerhalb von SQL-Anweisungen möglich.

EXISTS unterabfrage

---

unterabfrage
Unterabfrage, die eine Ergebnistabelle liefert.

Ergebnis

Wahr, wenn die Ergebnistabelle nicht leer ist.

Falsch, wenn die Ergebnistabelle leer ist.

Beispiel

Aus der Tabelle kunde alle Kunden heraussuchen, die keinen Auftrag vergeben haben:

```
DECLARE ... CURSOR FOR SELECT firma
    FROM kunde
    WHERE NOT EXISTS
    (SELECT anr
        FROM auftrag
        WHERE auftrag.knr = kunde.knr);
...
```

```
firma
Siemens AG
Plenzer Trading
Jonas Fischladen
Externa & Co Kg
```

select_ausdruck

```
select_ausdruck::=

  SELECT [ { ALL | DISTINCT } ] select_liste

    FROM tabellenangabe,...

    [ WHERE bedingung ]

    [ GROUP BY spalte,... ]

    [ HAVING bedingung ]


  select_liste ::= { * | { tabelle.* | sqlausdruck [ [ AS ] spalte ] },... }
```

ALL

ALL ist Voreinstellung.
Doppelte Sätze in der Ergebnistabelle bleiben erhalten.

DISTINCT

Doppelte Sätze werden entfernt (Duplikatelimination).

DISTINCT darf nur einmal in **derselben** Ebene einer SELECT-Abfrage verwendet werden. Sie dürfen zum Beispiel nicht angeben:

```
SELECT DISTINCT      COUNT(DISTINCT ...) ...
```

Über die FROM-Klausel und gegebenenfalls WHERE-Klausel können Sie einen Join bilden, d.h. zwei oder mehrere Tabellen miteinander verknüpfen und das kartesische Produkt aus allen beteiligten Tabellen bilden (siehe auch Metavariable *join_ausdruck*).

Für alle Klauseln gilt:

– Die angegebene Reihenfolge der Klauseln muß eingehalten werden.

– Spaltennamen müssen eindeutig sein. Kommt ein Spaltenname in mehreren Tabellen vor, so müssen Sie den Spaltennamen mit dem Tabellennamen qualifizieren. Wenn Sie eine Tabelle mit einem Korrelationsnamen für die Dauer der SELECT-Anweisung umbenennen (siehe Metavariable *tabellenangabe*), dürfen Sie nur noch den Korrelationsnamen verwenden.

Beispiel

```
SELECT a.knr, l.lsatz
FROM auftrag a, leistung l
WHERE a.anr=l.anr;
```

Auswertung von SELECT-Ausdrücken

SELECT-Ausdrücke werden in folgender Reihenfolge ausgewertet:

1. Aus allen Tabellenangaben in der FROM-Klausel wird das kartesische Produkt gebildet (siehe Metavariable *tabellenangabe*: es gibt Basistabellen, Viewtabellen, temporäre Viewtabellen, Ergebnistabellen und Jointabellen).
2. Ist eine WHERE-Klausel angegeben, wird die WHERE-Bedingung auf alle Sätze des kartesischen Produkts angewendet. Es werden die Sätze ausgewählt, für die die Bedingung den Wahrheitswert **wahr** ergibt.
3. Ist eine GROUP BY-Klausel angegeben, werden die in Punkt 2 bzw. Punkt 1 (falls keine WHERE-Klausel angegeben ist) bestimmten Sätze zu Gruppen zusammengefaßt. Jede Gruppe wird größtmöglich bezüglich der Eigenschaft gebildet, daß alle ihre Sätze für jede in der GROUP BY-Klausel angegebene Spalte einen identischen, gemeinsamen Wert haben.
4. Ist eine HAVING-Klausel angegeben, wird die HAVING-Bedingung auf alle Gruppen angewendet. Es werden die Gruppen ausgewählt, die die Bedingung erfüllen. Ist keine GROUP BY-Klausel angegeben, gelten alle bisher ausgewählten Sätze als eine Gruppe.
5. Enthält die *select_liste* eine Mengenfunktion und ist die Ergebnistabelle noch nicht gruppiert, werden alle Sätze der Ergebnistabelle zu einer Gruppe zusammengefaßt.
6. Ist die Ergebnistabelle gruppiert (eine oder mehrere Gruppen), wird die *select_liste* für jede Gruppe ausgewertet.

   Ist die Ergebnistabelle nicht gruppiert, wird die *select_liste* für jeden Ergebnissatz ausgewertet.
7. Ist DISTINCT angegeben, so werden doppelte Sätze entfernt.

Die resultierenden Sätze bilden die Ergebnistabelle des *select_ausdruck*.

select_liste - Ergebnisspalten auswählen

Mit *select_liste* legen Sie die Spalten der Ergebnistabelle fest.

---

select_liste ::= { * | { tabelle.* | sqlausdruck [[AS] spalte] },... }

---

*

Alle Spalten auswählen. Die Reihenfolge und die Namen der Spalten der in der FROM-Klausel angegebenen Tabellen werden übernommen. Bei mehreren Tabellen gilt die Reihenfolge der Tabellen in der FROM-Klausel.

tabelle.*

Alle Spalten der Tabelle *tabelle* auswählen. Die Tabelle *tabelle* muß in der FROM-Klausel enthalten sein. Die Reihenfolge und die Namen der Spalten von *tabelle* werden übernommen.

sqlausdruck

Bezeichnet eine Ergebnisspalte. Enthält *sqlausdruck* eine Spaltenangabe, muß die Tabelle, zu der die Spalte gehört, in der FROM-Klausel dieses oder eines übergeordneten *select_ausdruck* enthalten sein.

Die Namen der Spalten in *select_liste* müssen eindeutig sein. Wenn Sie Tabellen verbinden und diese Basistabellen Spalten mit identischen Namen haben, müssen Sie zur eindeutigen Identifikation die jeweiligen Namen der Tabellen bzw. deren Korrelationsnamen voranstellen.

Wenn in einer Spaltenauswahl eine Mengenfunktion (AVG, COUNT, MAX, MIN, SUM) vorkommt, gilt folgende Einschränkung:

In *select_liste* dürfen nur Spaltennamen vorkommen, die in der GROUP BY-Klausel aufgeführt sind oder Argument einer Mengenfunktion sind.

[AS] *spalte*

Name für die Ergebnisspalte, die mit *sqlausdruck* angegeben ist.

Beispiel

```
SELECT anr AS auftrags_nr, COUNT(*) AS anzahl FROM auftrag GROUP BY anr

auftrags_nr     anzahl
...             ...
```

spalte nicht angegeben:
Wenn *sqlausdruck* ein Spaltenname ist, erhält die Ergebnisspalte diesen Namen, andernfalls ist der Spaltenname nicht definiert.

Beispiel

```
SELECT anr, COUNT(*) FROM auftrag GROUP BY anr
```

```
anr
...          ...
```

Spalten der Ergebnistabelle

Die Reihenfolge der Spalten in der Ergebnistabelle entspricht der Reihenfolge der Spalten in *select_liste*.

Die Attribute einer Ergebnisspalte (Datentyp, Länge, Genauigkeit, Nachkommastellen) werden entweder von der zugrundeliegenden Spalte übernommen oder ergeben sich aus dem angegebenen Ausdruck.

Für eine Ergebnisspalte ist der NULL-Wert erlaubt, wenn eine der folgenden Bedingungen gilt:

– für die Ausgangsspalte ist der NULL-Wert erlaubt
– *sqlausdruck* enthält eine Variable, eine Unterabfrage, SYSTEM USER oder die Mengenfunktion AVG, MAX, MIN oder SUM.

FROM-Klausel - Tabellen angeben

In der FROM-Klausel geben Sie die Tabellen an, aus denen Daten ausgewählt werden sollen.

Um Sätze in den angegebenen Tabellen lesen zu können, müssen Sie entweder Eigentümer dieser Tabellen sein oder das SELECT-Zugriffsrecht besitzen.

FROM tabellenangabe,...

---

tabellenangabe

Angabe einer Tabelle, aus der Daten gelesen werden. Sie dürfen nur Tabellen derselben Datenbank angeben. Alle Tabellennamen in der FROM-Klausel können daher mit höchstems einem gemeinsamen Datenbanknamen qualifiziert sein. Werden nicht alle Tabellennamen mit einem solchen gemeinsamen Datenbanknamen qualifiziert, so kann für die Qualifizierung nur der voreingestellte Datenbankname verwendet werden (siehe OPTION CATALOG für statische und SET CATALOG sowie SET SCHEMA für dynamische SQL-Anweisungen).

Ein Tabellenname *tabelle* in einer *tabellenangabe* heißt synonymfrei, falls in der *tabellenangabe* kein *korrelationsname* zu *tabelle* angegeben ist. In der FROM-Klausel müssen alle Korrelationsnamen und alle synonymfreien Tabellennamen unterschiedlich sein.

WHERE-Klausel - Ergebnissätze auswählen

In der WHERE-Klausel geben Sie eine Bedingung an, um Sätze für die Ergebnistabelle auszuwählen. Die Ergebnistabelle enthält nur die Sätze, die die angegebene Bedingung erfüllen (d.h. die Bedingung ist **wahr**). Sätze, für die die Bedingung **falsch** oder **unbestimmt** ergibt, werden nicht in die Ergebnistabelle aufgenommen.

WHERE bedingung

---

bedingung

Bedingung, die die auszuwählenden Sätze erfüllen müssen (siehe Metavariable *bedingung*).

GROUP BY-Klausel - Ergebnissätze gruppieren

Mit Hilfe der GROUP BY-Klausel werden Tabellensätze zu Gruppen zusammengefaßt. Zwei Sätze gehören zu derselben Gruppe, wenn für jede Gruppierungsspalte die Werte in beiden Sätzen nach den Vergleichsregeln (siehe Metavariable *praedikat*) gleich sind oder beide der NULL-Wert sind.

Die Ergebnistabelle enthält für jede Gruppe einen Satz.

GROUP BY spalte,...

---

spalte

Gruppierungsspalte. *spalte* muß Teil einer Tabelle sein, die in der FROM-Klausel angegeben wurde. Nicht eindeutige Spaltennamen müssen mit dem Tabellennamen qualifiziert werden. Haben Sie in der FROM-Klausel einen Korrelationsnamen für die betroffene Tabelle vereinbart, muß dieser Name zur Qualifizierung verwendet werden.

Multiple Spalten dürfen nicht als Gruppierungsspalte verwendet werden.

Auswirkung der GROUP BY-Klausel

Wenn Sie die GROUP BY-Klausel angeben, dürfen in *select_liste* nur noch Spaltennamen vorkommen, die bei GROUP BY aufgeführt oder Argument einer Mengenfunktion sind.

Mengenfunktionen für Spalten einer gruppierten Tabelle werden für jede Gruppe ausgewertet.

Wie werden die Gruppen gebildet?

– Die Sätze, die in allen angegebenen Gruppierungsspalten einen nach den Vergleichsregeln (s.o.) gleichen Wert enthalten, bilden eine Gruppe.

– Sätze, die in den gleichen Gruppierungsspalten den NULL-Wert und in den restlichen Gruppierungsspalten gleiche Werte enthalten, werden zu einer Gruppe zusammengefaßt.

Beispiel

Für jede Auftragsnummer den durchschnittlichen Mehrwertsteuersatz bilden:

```
DECLARE ... CURSOR FOR SELECT anr, AVG(mwsatz) AS mwst
FROM leistung
GROUP BY anr;

...
```

```
anr     mwst
200     0.14
211     0.06
250     0.07
```

HAVING-Klausel - Gruppen auswählen

In der HAVING-Klausel geben Sie Bedingungen an, um Gruppen auszuwählen. Die Ergebnistabelle enthält den Satz für eine Gruppe, wenn die Gruppe die angegebene Bedingung erfüllt. Ist keine GROUP BY-Klausel angegeben, gelten alle Sätze als eine Gruppe.

HAVING bedingung

---

bedingung
Bedingung, die eine Gruppe erfüllen muß (siehe Metavariable *bedingung*).

Im Unterschied zur WHERE-Bedingung, die für jeden Satz einer Tabelle ausgewertet wird, wird die HAVING-Bedingung einmal pro Gruppe ausgewertet.

Für einen Spaltennamen in *bedingung* muß eine der folgenden Bedingungen gelten:

– Der Spaltenname ist in der GROUP BY-Klausel aufgeführt.

– Der Spaltenname ist Argument einer Mengenfunktion (AVG(), COUNT(), MAX(), MIN(), SUM()). Wenn der Spaltenname auch in *select_liste* erscheint, darf er dort auch nur als Argument einer Mengenfunktion vorkommen.

– Der Spaltenname kommt in einer Unterabfrage vor. Wenn der Spaltenname sich auf die Tabellen in der FROM-Klausel bezieht, muß er in der GROUP BY-Klausel aufgeführt oder Argument einer Mengenfunktion sein.

– Der Spaltenname ist Teil einer Tabelle aus einem übergeordneten *select_ausdruck*.

Beispiel

Für jeden Auftrag soll die zuletzt erbrachte Leistung angezeigt werden, sofern sie nach dem 1.1.1991 erfolgt ist:

```
SELECT anr, MAX(ldatum)
  FROM leistung
  GROUP BY anr
  HAVING MAX(ldatum) > DATE(1991-01-01);
```

spaltenbedingung

Bei der Erzeugung oder Änderung einer Basistabelle (CREATE TABLE, ALTER TABLE) können in der Spaltendefinition für einzelne Spalten Spaltenbedingungen angegeben werden. Die Spalte darf keine multiple Spalte sein.

Eine Spaltenbedingung ist eine Integritätsbedingung, die sich auf eine Spalte bezieht. Alle Werte der Spalte müssen die Integritätsbedingung erfüllen.

---

spaltenbedingung::=

```
{ NOT NULL |
  UNIQUE |
  PRIMARY KEY |
  CHECK ( bedingung ) |
  REFERENCES tabelle [ ( spalte ) ]
```

NOT NULL
: Nicht-NULL-Bedingung.
Die Spalte darf keine NULL-Werte enthalten.

  Die Nicht-NULL-Bedingung wird als Check-Bedingung (*spalte* IS NOT NULL) abgespeichert.

UNIQUE
: Eindeutigkeitsbedingung.
Die Spaltenwerte, die ungleich dem NULL-Wert sind, müssen eindeutig sein.

  Die Spalte darf nicht länger als 256 Zeichen sein.

PRIMARY KEY
: Primärschlüsselbedingung.
Die Spalte ist der Primärschlüssel der Tabelle. Die Werte der Spalte müssen eindeutig sein. Für jede Tabelle kann nur ein Primärschlüssel definiert werden.

  Die Spalte darf nicht vom Datentyp VARCHAR sein. Die Länge der Spalte muß zwischen 4 und 256 Zeichen sein.

  Für eine Primärschlüsselspalte gilt implizit auch die Nicht-NULL-Bedingung.

CHECK (*bedingung*)

Check-Bedingung.
Jeder Wert der Spalte muß die Bedingung *bedingung* erfüllen. Für *bedingung* gelten folgende Einschränkungen:

- *bedingung* darf keine Variablen enthalten.
- *bedingung* darf keine Mengenfunktion enthalten.
- *bedingung* darf keine Unterabfrage enthalten, also kann sich *bedingung* nur auf die Spalte der Tabelle beziehen, zu der die Spaltenbedingung gehört.
- *bedingung* darf keine Zeitfunktion enthalten.
- *bedingung* darf weder USER, CURRENT USER noch SYSTEM USER enthalten.

REFERENCES

Referenzbedingung.
Die Spalte der referenzierenden Tabelle darf einen vom NULL-Wert verschiedenen Wert nur dann enthalten, wenn derselbe Wert in der referenzierten Spalte der referenzierten Tabelle enthalten ist.

Der aktuelle Berechtigungsschlüssel muß das Privileg REFERENCES für die referenzierten Spalten besitzen.

tabelle

Name der referenzierten Basistabelle.
Die referenzierte Basistabelle muß eine SQL-Tabelle sein. Der einfache Name der referenzierten Basistabelle kann durch einen Datenbank- und Schemanamen qualifiziert werden. Der Datenbankname muß mit dem Datenbanknamen der referenzierenden Tabelle übereinstimmen.

(*spalte*)

Name der referenzierten Spalte.
Die referenzierte Spalte muß mit UNIQUE oder PRIMARY KEY definiert sein. Die referenzierte Spalte darf keine multiple Spalte sein. Referenzierende Spalte und referenzierte Spalte müssen genau denselben Datentyp besitzen.

(*spalte*) nicht angegeben:
Der Primärschlüssel der referenzierten Tabelle wird als referenzierte Spalte verwendet. Referenzierende Spalte und referenzierte Spalte müssen genau denselben Datentyp besitzen.

spaltendefinition

Die Spaltendefinition legt bei der Erzeugung oder Änderung einer Basistabelle (CREATE TABLE, ALTER TABLE) den Namen und die Eigenschaften einer Spalte fest.

SESAM/SQL unterscheidet zwischen einfachen und multiplen Spalten. Bei einer einfachen Spalte kann pro Satz genau ein Wert gespeichert werden. Bei einer multiplen Spalte können pro Satz mehrere Werte desselben Datentyps gespeichert werden.

Eine Basistabelle kann max. 26134 Spalten eines Datentyps außer VARCHAR und max. 1000 Spalten mit Datentyp VARCHAR enthalten.

```
spaltendefinition::=

  spalte [ ( dimension ) ] { grunddatentyp | FLOAT ( stellenanzahl ) }
                                  [ voreinstellung ]

  [ [ CONSTRAINT integritätsbedingungsname ] spaltenbedingung ... ]


  dimension::= vorzeichenlose_ganzzahl

  voreinstellung::= DEFAULT {  literal |
                    CURRENT DATE |
                    CURRENT TIME |
                    CURRENT TIMESTAMP |
                    [ CURRENT ] USER |
                    SYSTEM USER |
                    NULL }
```

---

spalte
Name der Spalte. Der Spaltenname muß innerhalb der Basistabelle eindeutig sein.

dimension
Vorzeichenlose Ganzzahl zwischen 1 und 255. *dimension* gibt die Anzahl der Spaltenelemente einer multiplen Spalte an.
dimension nicht angegeben: Die Spalte ist eine einfache Spalte.
Bei VARCHAR (*länge*) und CHAR VARYING (*länge*) ist die Angabe von *dimension* nicht erlaubt.

grunddatentyp
Datentyp für die Spalte (siehe DRIVE-Lexikon [3], Metavariable *grunddatentyp*).
Beim Datentyp CHAR muß die Länge kleiner als 257 sein.
Bei den Datentypen NUMERIC und DECIMAL darf *stellenanzahl* max. 31 sein.
Die Vorbelegung von *stellenanzahl* ist 1.
Beim Datentyp SMALLINT ist die Abkürzung SMINT nicht erlaubt.
Beim Datentyp NUMERIC ist die Abkürzung NUM nicht erlaubt.
Der Datentyp EXTENDED DECIMAL bzw. XDEC ist nicht erlaubt.
FLOAT bezeichnet den Datentyp FLOAT(1) (und nicht DOUBLE PRECISION).
Der Datentyp TIME ist nicht erlaubt (nur TIME(3)).
Der Datentyp INTERVAL ist nicht erlaubt.
usertyp ist nicht erlaubt.

FLOAT (*stellenanzah*l)
Beim Datentyp FLOAT bezeichnet *stellenanzahl* die binäre Mantissenlänge. Sie muß größer als 0 und kleiner als 54 sein. Der Wertebereich von FLOAT entspricht dem von REAL, falls *stellenanzahl* kleiner oder gleich 21 ist. Andernfalls entspricht er dem von DOUBLE PRECISION. Die Vorbelegung von *stellenanzahl* ist 1.

voreinstellung
Legt einen SQL-Defaultwert fest, der in die Spalte eingetragen wird, wenn ein Satz eingefügt oder geändert wird und für die Spalte kein Wert und auch nicht der NULL-Wert angegeben ist.

- *spalte* darf keine multiple Spalte sein.
- *spalte* darf keine CALL-DML-Spalte sein.
- *voreinstellung* muß den Zuweisungsregeln für Defaultwerte genügen (siehe SESAM/SQL-Server SQL-Sprachbeschreibung Teil1 [18] Abschnitt „Werte in Tabellenspalten eintragen“ auf Seite 86).
- Der voreingestellte Wert muß die Spaltenbedingung erfüllen.

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

voreinstellung nicht angegeben:
Es gibt keinen SQL-Defaultwert.
Bei Spalten ohne Nicht-NULL-Bedingung wird der NULL-Wert eingetragen.

Aus Kompatibilitätsgründen mit der SQL-Norm wird empfohlen, für den zu CURRENT USER oder SYSTEM USER gehörenden Datentyp CHAR(*länge*) oder VARCHAR(*länge*) *länge* größer oder gleich 128 zu wählen.

[CONSTRAINT *integritätsbedingungsname*] *spaltenbedingung*
Definiert eine Integritätsbedingung für die Spalte. Sie dürfen Integritätsbedingungen nicht für multiple Spalten angeben.

[CONSTRAINT *integritätsbedingungsname*] *spaltenbedingung* nicht angegeben:
Keine Spaltenbedingung definiert.

CONSTRAINT *integritätsbedingungsname*
Vergibt einen Namen für die Integritätsbedingung. 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 folgender Regel:

{ UN | PK | FK | CH } integritätsbedingungsnummer

wobei das zweistellige Präfix UN für UNIQUE, PK für PRIMARY KEY, FK für FOREIGN KEY und CH für CHECK steht. *integritätsbedingungsnummer* ist eine 16stellige Ganzzahl (Zeitstempel). Die Nicht-NULL-Bedingung wird als Check-Bedingung abgespeichert.

spaltenbedingung
Gibt eine Integritätsbedingung an, die die Spalte erfüllen muß (siehe Metavariable *spaltenbedingun*g).

sqlausdruck

sqlausdruck ergibt einen Wert. *sqlausdruck* kann vorkommen in:

– Spaltenauswahl (*select_ausdruck*, SELECT-Anweisung)

– Prädikaten von bedingung (z.B. WHERE-, HAVING-Klausel)

– Zuweisungen (INSERT-, UPDATE-Anweisung).

sqlausdruck besteht aus Operanden und eventuell Operatoren. Ist ein Operand der NULL-Wert, ist das Gesamtergebnis auch der NULL-Wert. Ansonsten werden die Operatoren auf die Ergebnisse der Operanden angewendet. Das Ergebnis der Auswertung ist ein alphanumerischer Wert, ein numerischer Wert oder ein Zeitwert.

Die Reihenfolge der Auswertung der Operanden ist nicht festgelegt. In bestimmten Fällen wird ein Teilausdruck nicht berechnet, wenn er für die Berechnung des Gesamtergebnisses nicht benötigt wird.

```
sqlausdruck::=
        { wert |
          [ tabelle. ] { spalte | spalte(posnummer) | spalte(min-max) } |
          { + | - } sqlausdruck |
          sqlausdruck { * | / | + | - | || } sqlausdruck |
          ( sqlausdruck ) |
          unterabfrage |
          mengenfunktion |
          zeitfunktion |
          [ CURRENT ] USER | SYSTEM USER }


  posnummer::= vorzeichenlose_ganzzahl

  min::= vorzeichenlose_ganzzahl

  max::= vorzeichenlose_ganzzahl
```

---

wert
: Alphanumerischer Wert, numerischer Wert oder Zeitwert (siehe Metavariable *wert*).

tabelle
: Name einer Tabelle, die *spalte* enthält. Statt des Tabellennamens müssen Sie den Korrelationsnamen angeben, wenn für die Tabelle ein Korrelationsname definiert ist (siehe Metavariable *tabellenangabe*).

spalte

Name einer Spalte, aus der die Werte genommen werden (Spaltenreferenz).

posnummer

Vorzeichenlose Ganzzahl.

Der Wert wird aus dem (*posnummer*-sp_{min}+1)-ten Spaltenelement der multiplen Spalte *spalte* genommen und kann wie ein einfacher Wert verwendet werden.

Ist *spalte* keine multiple Spalte, *posnummer* kleiner als sp_{min} oder *posnummer* größer als sp_{max}, erfolgt eine SESAM-Fehlermeldung.

sp_{min} bzw. sp_{max} ist die kleinste bzw. größte Positionsnummer der multiplen Spalte.

min-*max*

Vorzeichenlose Ganzzahlen.

Der Wert ist das Aggregat aus den Spaltenelementen (*min*-sp_{min}+1) bis (*max*-sp_{min}+1) der multiplen Spalte *spalte*.

Ist *spalte* keine multiple Spalte, *min* nicht kleiner als *max*, *min* kleiner als sp_{min} oder *max* größer als sp_{max}, erfolgt eine SESAM-Fehlermeldung.

sp_{min} bzw. sp_{max} ist die kleinste bzw. größte Positionsnummer der multiplen Spalte.

posnummer bzw. *min*-*max* nicht angegeben:
spalte darf keine multiple Spalte sein.

– *sqlausdruck*, +*sqlausdruck*

„–“ bewirkt einen Wechsel des Vorzeichens, d.h. der Wert von *sqlausdruck* wird negiert. „+“ läßt den Wert von *sqlausdruck* unverändert. *sqlausdruck* muß numerisch sein und darf kein multipler Wert mit Dimension > 1 sein. *sqlausdruck* darf nicht mit „+“ oder „–“ anfangen.
Erlaubt sind damit insbesondere folgende Varianten für numerische Werte bzw. Ausdrücke:
{ + | - } { spalte | wert | mengenfunktion | (sqlausdruck) }

sqlausdruck { + | - | * | / | || } *sqlausdruck*

bezeichnet die Rechenarten Addition, Subtraktion, Multiplikation und Division sowie die Konkatenation. Bei den Rechenarten +, -, * und / müssen beide Operanden numerisch sein, für die Konkatenation || müssen beide Ausdrücke alphanumerisch sein. Kein Operand darf eine strukturierte Variable oder ein Aggregat mit mehr als einer Komponente sein.

Wenn a und b vom Datentyp CHARACTER sind, ist das Ergebnis vom Datentyp CHARACTER mit der Länge l_a+l_b, höchstens jedoch 256.

Wenn a oder b vom Datentyp VARCHAR sind, ist das Ergebnis vom Datentyp VARCHAR mit der Länge l_a+l_b, höchstens jedoch 32000.
l_a und l_b sind die Längen von a und b.

Ist das Ergebnis vom Typ CHARACTER länger als 256 Zeichen, erfolgt eine SESAM-Fehlermeldung.

Ist das Ergebnis vom Typ VARCHAR länger als 32000 Zeichen, wird die Zeichenkette rechts auf die Länge 32000 gekürzt. Wenn Zeichen entfernt werden, die keine Leerzeichen sind, erfolgt eine SESAM-Fehlermeldung.

a ***** b

Multiplikation von a mit b.

Die Ausdrücke a und b müssen numerisch sein.

Wenn a und b Ganz- oder Festpunktzahlen sind, ist das Ergebnis eine Ganz- oder Festpunktzahl mit der Gesamtstellenzahl g_a+g_b, höchstens jedoch 31. Die Nachkommastellenzahl beträgt n_a+n_b, höchstens jedoch 31.
g_a und g_b sind die Gesamtstellenzahlen von a und b.
n_a und n_b sind die Nachkommastellenzahlen von a und b.

Wenn a oder b Gleitpunktzahlen sind, ist das Ergebnis eine Gleitpunktzahl mit der Gesamtstellenzahl 24 Bit bei REAL bzw. 56 Bit bei DOUBLE PRECISION.

Ist der Betrag des Ergebnisses für den Ergebnisdatentyp zu groß, erfolgt eine SESAM- Fehlermeldung. Ist die Gesamtstellenzahl zu groß, wird das Ergebnis gerundet.

a **/** b

Division von a durch b.

Die Ausdrücke a und b müssen numerisch sein.

Wenn a und b Ganz- oder Festpunktzahlen sind, ist das Ergebnis eine Ganz- oder Festpunktzahl mit der Gesamtstellenzahl 31. Die Nachkommastellenzahl beträgt 31-v_a-n_b, mindestens jedoch 0.
v_a ist die Vorkommastellenzahl von a.
n_b ist die Nachkommastellenzahl von b.

Wenn a oder b Gleitpunktzahlen sind, ist das Ergebnis eine Gleitpunktzahl mit der Gesamtstellenzahl 24 Bit bei REAL bzw. 56 Bit bei DOUBLE PRECISION.

Ist der Betrag des Ergebnisses für den Ergebnisdatentyp zu groß, oder ist der Wert von **b** gleich 0, erfolgt eine SESAM-Fehlermeldung. Ist die Gesamtstellenzahl zu groß, wird das Ergebnis gerundet.

a **+** *b*
Addition von *a* und *b*.

Die Ausdrücke *a* und *b* müssen numerisch sein.

Wenn *a* und *b* Ganz- oder Festpunktzahlen sind, ist das Ergebnis eine Ganz- oder Festpunktzahl mit der Gesamtstellenzahl v_{max}+n_{max}+1, höchstens jedoch 31. Die Nachkommastellenzahl beträgt n_{max}.
v_{max} ist die größere der beiden Vorkommastellenzahlen von *a* und *b*.
n_{max} ist die größere der beiden Nachkommastellenzahlen von *a* und *b*.

Wenn *a* oder *b* Gleitpunktzahlen sind, ist das Ergebnis eine Gleitpunktzahl mit der Gesamtstellenzahl 24 Bit bei REAL bzw. 56 Bit bei DOUBLE PRECISION.

Ist der Betrag des Ergebnisses für den Ergebnisdatentyp zu groß, erfolgt eine SESAM-Fehlermeldung. Ist die Gesamtstellenzahl zu groß, wird das Ergebnis gerundet.

a **-** *b*
Subtraktion b von a.

Die Ausdrücke *a* und *b* müssen numerisch sein.

Wenn *a* und *b* Ganz- oder Festpunktzahlen sind, ist das Ergebnis eine Ganz- oder Festpunktzahl mit der Gesamtstellenzahl v_{max}+n_{max}+1, höchstens jedoch 31. Die Nachkommastellenzahl beträgt n_{max}.
v_{max} ist die größere der beiden Vorkommastellenzahlen von *a* und *b*.
n_{max} ist die größere der beiden Nachkommastellenzahlen von *a* und *b*.

Wenn *a* oder *b* Gleitpunktzahlen sind, ist das Ergebnis eine Gleitpunktzahl mit der Gesamtstellenzahl 24 Bit bei REAL bzw. 56 Bit bei DOUBLE PRECISION.

Ist der Betrag des Ergebnisses für den Ergebnisdatentyp zu groß, erfolgt eine SESAM-Fehlermeldung. Ist die Gesamtstellenzahl zu groß, wird das Ergebnis gerundet.

a **||** *b*
Konkatenation von *a* mit *b*.

Die Ausdrücke *a* und *b* müssen alphanumerisch sein.

Wenn *a* und *b* vom Datentyp CHARACTER sind, ist das Ergebnis vom Datentyp CHARACTER mit der Länge l_a+l_b, höchstens jedoch 256.

Wenn *a* oder *b* vom Datentyp VARCHAR sind, ist das Ergebnis vom Datentyp VARCHAR mit der Länge l_a+l_b, höchstens jedoch 32000.
l_a und l_b sind die Längen von *a* und *b*.

Ist das Ergebnis vom Typ CHARACTER länger als 256 Zeichen, erfolgt eine SESAM-Fehlermeldung.

Ist das Ergebnis vom Typ VARCHAR länger als 32000 Zeichen, wird die Zeichenkette rechts auf die Länge 32000 gekürzt. Wenn Zeichen entfernt werden, die keine Leerzeichen sind, erfolgt eine SESAM-Fehlermeldung.

(*sqlausdruck*)
Mit Klammern können Sie Teile von Ausdrücken zu einer Einheit zusammenfassen, um die Reihenfolge der Auswertung von Rechenausdrücken zu verändern. Klammern müssen nach algebraischen Regeln gesetzt werden.

unterabfrage
Unterabfrage (siehe Metavariable *unterabfrage*), die genau einen Wert liefert.

mengenfunktion
Mengenfunktion (siehe Metavariable *mengenfunktion*). *sqlausdruck* bezeichnet dann den Wert, den diese Funktion liefert.
mengenfunktion darf nur in der *select_liste* einer SELECT-Anweisung oder eines *select-ausdruck* vorkommen.

zeitfunktion
Zeitfunktion (siehe Metavariable *zeitfunktion*).

[CURRENT] USER | SYSTEM USER
Spezifikation von SESAM-Userfunktionen. Das Ergebnis von [CURRENT] USER ist der aktuelle Berechtigungsschlüssel (vgl. CREATE USER-Anweisung). Er ist ein alphanumerisches Literal vom Typ CHARACTER (18).
Das Ergebnis von SYSTEM USER ist der Name des aktuellen Systembenutzers. Der Name wird zusammengesetzt aus dem Rechnernamen, dem UTM-Anwendungsnamen (bzw. Leerzeichen) und der UTM- bzw. BS2000-Benutzerkennung (vgl. CREATE SYSTEM_USER-Anweisung). Er ist ein alphanumerisches Literal vom Typ CHARACTER (24).

Beispiel

```
CREATE TABLE current_users (counter INTEGER PRIMARY KEY,
                    sqluser CHARACTER (18),
                    systemuser CHARACTER (24),
                    stamptime TIMESTAMP (3));

COMMIT WORK;

INSERT INTO current_users VALUES (*,
                        CURRENT USER,
                        CURRENT SYSTEM USER,
                        CURRENT TIMESTAMP)
                        RETURN INTO &counter;
```

Prioritäten

– Klammerausdrücke haben die höchste Priorität.

– Die einstelligen Operatoren haben Vorrang vor den zweistelligen.

– Die multiplikativen Operatoren * und / haben höhere Priorität als die additiven Operatoren + und -.

– Die multiplikativen Operatoren haben gleiche Priorität.

– Die additiven Operatoren haben gleiche Priorität.

– Operatoren gleicher Priorität werden von links nach rechts angewendet.

Enthält *sqlausdruck* keine Spaltenreferenzen, Mengenfunktionen, Unterabfragen und Funktionen, so ist diese Metavariable identisch mit dem entsprechenden Teil der DRIVE-Metavariablen *ausdruck* (vgl. DRIVE-Lexikon [3]). Nichtsdestotrotz wird *sqlausdruck* innerhalb einer SQL-Anweisung immer von SESAM berechnet. DRIVE berechnet nur *ausdruck* innerhalb von DRIVE-Anweisungen.

tabellenangabe

Mit *tabellenangabe* wird eine Tabelle angegeben.

```
tabellenangabe::= { tabelle [ [ AS ] korrelationsname [ ( spalte,... ) ] ] |
                  unterabfrage [ AS ] korrelationsname [ ( spalte,... ) ] |
                  join_ausdruck }


tabelle::= { [ [ catalog. ] einf_schemaname. ] einf_basistabellenname |
           [ [ catalog. ] einf_schemaname. ] einf_viewname |
             temp_viewname }
```

---

tabelle

Name einer Basistabelle, eines View oder temporären View.

catalog

Name für eine Datenbank (Catalog-Space *catalog*.CATALOG und Anwender-Spaces *catalog.spacename*). Ein Datenbankname darf max. 18 Zeichen lang sein (siehe CREATE CATALOG-Anweisung im SESAM-Handbuch [19]).

einf_schemaname

Name für ein Schema. Der einfache Schemaname muß innerhalb der Schemanamen der Datenbank eindeutig sein. Ein einfacher Schemaname darf max. 31 Zeichen lang sein (siehe CREATE SCHEMA-Anweisung).

einf_basistabellenname

Name für eine Basistabelle. Der einfache Name einer Basistabelle muß innerhalb der Basistabellen- und (persistenten) Viewnamen ihres Schemas eindeutig sein. Ein einfacher Basistabellenname darf max. 31 Zeichen lang sein (siehe CREATE TABLE-Anweisung).

einf_viewname

Name für einen persistenten View. Der einfache Name eines Views muß innerhalb der Basistabellen- und Viewnamen seines Schemas eindeutig sein. Ein einfacher Viewname darf max. 31 Zeichen lang sein (siehe CREATE VIEW-Anweisung).

temp_viewname

Name für einen temporären View. Der Name eines temporären Programm-Views muß innerhalb der Übersetzungseinheit (DRIVE-Programm) eindeutig sein, der eines temporären Dialog-Views innerhalb der DRIVE-Sitzung oder des UTM-Vor-

gangs. Ein temporärer Viewname darf max. 31 Zeichen lang sein, der Name eines statischen temporären Programm-Views max. 24 Zeichen (siehe CREATE TEMPORARY VIEW-Anweisung).

Dieselbe Tabelle kann mehrmals in einer Tabellenangabe in *abfrageausdruck* vorkommen. Um unterschiedliche Vorkommen derselben Tabelle unterscheiden zu können, müssen Korrelationsnamen verwendet werden.

korrelationsname

Tabellenname, der innerhalb von *abfrageausdruck* eine Umbenennung für eine Tabelle ist (Korrelationsname oder Synonym). Ein Korrelationsname darf max. 18 Zeichen lang sein.
Bei jeder Spaltenangabe, die sich auf diese Angabe der Tabelle bezieht, müssen Sie den Spaltennamen mit dem neuen Namen *korrelationsname* qualifizieren.

Der neue Name muß eindeutig sein, d.h. *korrelationsname* darf nur einmal in einer Tabellenangabe dieses *abfrageausdruck* vorkommen.

Sie müssen eine Tabelle umbenennen, wenn in *abfrageausdruck* die Spalten der Tabelle ohne Umbenennung nicht eindeutig angegeben werden können.

Außerdem können Sie eine Tabelle umbenennen, um durch entsprechende Namen *abfrageausdruck* verständlicher zu formulieren oder um lange Namen abzukürzen.

Beispiel

Tabelle mit sich selbst verknüpfen:

```
SELECT a.firma, b.firma          /* Kunden abfragen, die in */
  FROM kunde AS a, kunde AS b
  WHERE a.ort = b.ort            /* demselben Ort wohnen */
  AND a.knr < b.knr;             /* und gleiche Paare vermeiden */
```

spalte,...

Spaltenname, der innerhalb von *abfrageausdruck* eine Umbenennung für die Spalte der dazugehörigen Tabelle ist.

Wenn Sie eine Spalte umbenennen, müssen Sie allen Spalten der Tabelle einen neuen Namen zuweisen.

spalte ist der neue Name der Spalte, der innerhalb der mit *korrelationsname* angegebenen Tabelle eindeutig sein muß. Die Spalte darf in diesem *abfrageausdruck* nur mit dem neuen Namen angesprochen werden.

Die Spalten einer Ergebnistabelle müssen umbenannt werden, wenn die Spaltennamen der zugrundeliegenden Tabellen nicht eindeutig sind oder wenn Ergebnisspalten mit intern vergebenen Namen angesprochen werden sollen.

Beispiel

Die Spalten der Tabelle LAGER sollen mit neuen, aussagekräftigeren Namen versehen werden:

```
SELECT * FROM lager l (artikelnummer, aktueller_bestand, lagerort)
  WHERE lagerort = 'Teilelager Ost';
```

spalte,... nicht angegeben:
Es gelten die Spaltennamen der zugehörigen Tabelle. Dies können intern vergebene Namen sein, die in *abfrageausdruck* nicht angesprochen werden können.

unterabfrage
Die Tabelle ist die Ergebnistabelle der Unterabfrage (siehe Metavariable *unterabfrage)*.

join_ausdruck
Join-Ausdruck, der die Tabellen benennt, aus denen Daten durch einen Join ausgewählt werden (siehe Metavariable *join_ausdruck*).

Zugrundeliegende Tabellen

Abhängig von der Angabe in der Tabellenangabe sind die zugrundliegenden Tabellen wie folgt definiert:

Angabe in Tabellenangabe	**zugrundeliegende Tabelle(n)**
Basistabelle	Basistabelle
View oder temporärer View	Basistabellen, auf die sich der (temporäre) View direkt oder indirekt bezieht
Unterabfrage	die der Unterabfrage zugrundeliegende Tabelle (siehe Metavariablen *select_ausdruck* und *abfrageausdruck*)
Join-Ausdruck	die den Tabellenangaben des *join_ausdruck* zugrundliegenden Tabellen

tabellenbedingung

Bei der Erzeugung oder Änderung einer Basistabelle (CREATE TABLE, ALTER TABLE) können Tabellenbedingungen angegeben werden. Eine Tabellenbedingung ist eine Integritätsbedingung, die sich auf mehr als eine Spalte der Basistabelle beziehen kann. Keine der Spalten darf eine multiple Spalte sein.

---

tabellenbedingung::=

```
{ UNIQUE ( { spalte,... } ) |
  PRIMARY KEY ( { spalte,... } ) |
  FOREIGN KEY ( { spalte,... } ) REFERENCES tabelle [ ( { spalte },... ) ] |
  CHECK ( bedingung )
```

UNIQUE (*spalte*,...)

Eindeutigkeitsbedingung.
Die Kombination der Spaltenwerte, die ungleich dem NULL-Wert sind, muß eindeutig sein.

Die Summe der Spaltenlängen plus die Gesamtzahl der Spalten darf 256 nicht überschreiten.

PRIMARY KEY (*spalte*,...)

Primärschlüsselbedingung.
Die angegebenen Spalten bilden zusammen den Primärschlüssel der Tabelle. Die Kombination der Spaltenwerte muß eindeutig sein. Für jede Tabelle kann nur ein Primärschlüssel definiert werden.

Keine der Spalten darf vom Datentyp VARCHAR sein. Die Summe der Spaltenlängen muß zwischen 4 und 256 Zeichen liegen.

Für Primärschlüsselspalten gilt implizit auch die Nicht-NULL-Bedingung.

FOREIGN KEY ... REFERENCES

Referenzbedingung.
Die referenzierenden Spalten (FOREIGN KEY-Klausel) dürfen eine Wertekombination, die keinen NULL-Wert enthält, nur dann enthalten, wenn die Wertekombination in den referenzierten Spalten (REFERENCES-Klausel) ebenfalls vorkommt.
Sie müssen die gleiche Anzahl Spalten der referenzierenden und der referenzierten Tabelle angeben. Die Datentypen der korrespondierenden Spalten müssen genau gleich sein.

Der aktuelle Berechtigungsschlüssel muß das Privileg REFERENCES für die referenzierten Spalten besitzen.

FOREIGN KEY (*spalte*,...)
Spalten der referenzierenden Tabelle, deren Wertekombination in der referenzierten Basistabelle enthalten sein soll.

REFERENCES *tabelle*
Name der referenzierten Basistabelle.
Die referenzierte Basistabelle muß eine SQL-Tabelle sein. Der einfache Name der referenzierten Basistabelle kann durch einen Datenbank- und Schemanamen qualifiziert werden. Der Datenbankname muß mit dem Datenbanknamen der referenzierenden Tabelle übereinstimmen.

(*spalte*,...)
Namen der referenzierten Spalten.
Für diese Spalten muß eine Eindeutigkeits- oder Primärschlüsselbedingung definiert sein, die dieselben Spalten und dieselbe Reihenfolge verwendet. Keine der Spalten darf eine multiple Spalte sein.

(*spalte*,...) nicht angegeben:
Der Primärschlüssel der referenzierten Tabelle wird als referenzierte Spalte verwendet.

CHECK (*bedingung*)
Check-Bedingung.
Für jeden Datensatz der Tabelle muß die Bedingung *bedingung* erfüllt sein. Für *bedingung* gelten folgende Einschränkungen:

- *bedingung* darf keine Variablen enthalten.
- *bedingung* darf keine Mengenfunktion enthalten.
- *bedingung* darf keine Unterabfrage enthalten, also kann sich *bedingung* nur auf Spalten der Tabelle beziehen, zu der die Tabellenbedingung gehört.
- *bedingung* darf keine Zeitfunktion enthalten.
- *bedingung* darf weder USER , CURRENT USER noch SYSTEM USER enthalten.

unterabfrage

Eine Unterabfrage ist die Spezifikation einer Tabelle als Ergebnistabelle eines *abfrageausdruck*, der in folgenden Fällen verwendet werden kann:

– In Ausdrücken:
  Die Unterabfrage muß eine einspaltige Ergebnistabelle mit höchstens einem Satz liefern. Der Wert der Unterabfrage ist dann der Wert in der Ergebnistabelle bzw. der NULL-Wert, wenn die Ergebnistabelle leer ist.

– In Prädikaten:
  In den Prädikaten ANY, SOME, ALL und IN muß die Unterabfrage eine einspaltige Ergebnistabelle liefern. Im Prädikat EXISTS kann die Unterabfrage eine beliebige Ergebnistabelle liefern.

– In der FROM-Klausel von SELECT-Ausdrücken:
  Die Unterabfrage liefert eine Ergebnistabelle.

– In Join-Ausdrücken:
  Die Unterabfrage liefert eine Ergebnistabelle.

Eine Unterabfrage wird immer in runde Klammern eingeschlossen.

---

unterabfrage::= (abfrageausdruck)

---

abfrageausdruck
  Abfrageausdruck, der die Ergebnistabelle liefert.

  Bei Unterabfragen, die nicht im Prädikat EXISTS oder in einer FROM-Klausel oder in einem *join_ausdruck* angegeben sind, darf die Ergebnistabelle nur einfache Spalten oder multiple Spalten mit Dimension 1 enthalten.

variable

variable referenziert eine einfache Variable oder eine Komponente einer strukturierten Variablen. Die Liste aller Komponenten, die auf der nächsten Stufe liegen, kann mit der Teilqualifizierung „.*“ angegeben werden. *variable* hat in DRIVE/WINDOWS folgende Schreibweise (siehe auch DRIVE-Lexikon [3]):

```
variable::= { &vamame1 [ suffix ] |
        &varname2 { ( index1, index2 ) |
        ( index1, bereich2 ) |
        ( bereich1, index2 ) }

    suffix::= { gruppenkomponente | indexkomponente }

    gruppenkomponente::= . { * | komponente [ suffix ] }

    indexkomponente::= { ( { index | bereich } ) }
    index::= vorzeichenlose_ganzzahl
    bereich::= index1 - index2
```

---

variable

Dient dazu, in einer SQL-Anweisung Werte aus der Datenbank aufzunehmen (Ausgabe-Variable) oder in die Datenbank zu speichern und Werte bereitzustellen, die in Berechnungen und Bedingungen benötigt werden (Eingabe-Variable). Datentypen von Spalten und zugehörigen Variablen müssen miteinander verträglich sein.

varnname1

Name der einfachen Variablen oder der ersten Qualifizierung der Komponente der strukturierten Variablen. *varname1* darf maximal 31 Zeichen lang sein.
Die ausführliche Syntaxbeschreibung finden Sie bei der Metavariablen *variable* im DRIVE- Lexikon [3].

suffix

Weitere Qualifizierungen der Komponente der strukturierten Variablen.

.*

Abkürzende Schreibweise für die Liste aller Variablenkomponenten, die als Komponenten auf der nächst tieferen Stufe liegen

komponente

Name einer Komponenten, d.h. eines Teils einer strukturierten Variablen, oder einfacher Name einer Tabellenspalte (siehe Anweisung DECLARE VARIABLE... LIKE CURSOR/TABLE). *komponente* darf maximal 31 Zeichen lang sein.

varname2

Name einer Matrix. *varname2* darf maximal 31 Zeichen lang sein.

DRIVE/WINDOWS bietet Ihnen die Möglichkeit, sich durch DECLARE VARIABLE ... LIKE Variablen anzulegen, deren Struktur und Namen denen einer Tabelle bzw. eines Cursors entsprechen.

wert

Ein Wert kann entweder als Literal oder über eine Variable angegeben werden. *wert* legt dann den Datenwert für eine Variable oder für eine Komponente einer Variablen fest. Jeder Variablen und jeder Tabellenspalte kann der NULL-Wert zugeordnet werden, falls nicht bei der Definition (DECLARE VARIABLE, CREATE/ALTER TABLE) eine Nicht-NULL-Bedingung vereinbart wurde (siehe DRIVE-Anweisung SET und SQL-Anweisungen INSERT und UPDATE).

---

wert::= { literal | variable | aggregat }

aggregat::= < { wert | NULL }, ... >

---

literal

Alphanumerisches Literal, numerisches Literal oder Zeitliteral (siehe Metavariable *literal*).

variable

Name einer Variablen, die den Wert enthält (siehe Metavariable *variable*).

aggregat

spezifiziert ein Aggregat, d.h. einen strukturierten Wert, dessen Komponenten durch *wert* bzw. NULL festgelegt sind. *aggregat* darf nicht mehr als 255 Komponenten enthalten. Ein Aggregat heißt auch multipler Wert, weil es die Angabe für Werte von multiplen Spalten ist. Entsprechend heißt die Anzahl der Komponenten eines Aggregats auch Dimension (siehe auch Metavariable *spaltendefinition*).

Beispiel

```
CREATE TABLE demo (demo (5) INTEGER);

COMMIT WORK;

INSERT INTO demo (demo (2-4)) VALUES (<13,NULL,67>);
```

Die multiple Spalte demo enthält dann den multiplen Wert <NULL,13,NULL,67,NULL>.

wert

Für *wert* dürfen Literale, und Variablen angegeben werden, d.h. geschachtelte Aggregate sind nicht erlaubt. Bei strukturierten Variablen darf nur die unterste Struktur, aber nicht die gesamte Struktur angesprochen werden, bei Vektoren darf nur eine Komponente angesprochen werden.

NULL
Der jeweiligen Komponente von *aggregat* wird der NULL-Wert zugewiesen. Jeder Variablen (SET-Anweisung) und jeder Spalte einer Tabelle (Anweisungen INSERT und UPDATE) kann der NULL-Wert zugeordnet werden.

zeitfunktion

Zeitfunktionen ermitteln das aktuelle Datum und/oder die aktuelle Uhrzeit:

---

zeitfunktion::= { CURRENT DATE | CURRENT TIME | CURRENT TIMESTAMP }

CURRENT DATE

Liefert das aktuelle Datum. **Datentyp**: DATE.

CURRENT TIME

Liefert die aktuelle Uhrzeit. **Datentyp**: TIME(3).

CURRENT TIMESTAMP

Liefert den aktuellen Zeitstempel. **Datentyp**: TIMESTAMP(3).

Kommen Zeitfunktionen innerhalb einer SQL-Anweisung mehrmals vor, werden diese simultan von SESAM ausgeführt. Das gilt auch für alle Zeitfunktionen, die als Folge der Anweisung ausgewertet werden:

– Zeitfunktionen in der DEFAULT-Klausel der Spaltendefinition, wenn die Voreinstellung verwendet wird.

– Zeitfunktionen, die im SELECT-Ausdruck eines View oder temporären View vorkommen, wenn der View bzw. der temporäre View angesprochen wird.

Zeitfunktionen innerhalb von DRIVE-Anweisungen werden von DRIVE/WINDOWS ausgeführt.

Zeitfunktionen in dynamisch formulierten Anweisungen und in Cursorbeschreibungen werden zum Zeitpunkt der EXECUTE-Anweisung ausgewertet.

Alle zurückgelieferten Werte enthalten gleiches Datum und/oder gleiche Uhrzeit. Daher können Sie Zeitfunktionen nicht dazu verwenden, Ausführungszeitpunkte innerhalb einer SQL- oder DRIVE-Anweisung festzustellen. Dagegen können Sie Zeitfunktionen dazu verwenden, **näherungsweise** die Ausführungsdauer einzelner Anweisungen oder Anweisungsblöcke zu bestimmen.

Beispiele

1. Beispiel

```
DCL VAR &vorher TIMESTAMP(3),
        &nachher TIMESTAMP(3),
        &dauer INTERVAL FRACTIONS;
SET &vorher=CURRENT TIMESTAMP;
INSERT INTO demo VALUES (<1,2,3,4,5>);
SET &nachher=CURRENT TIMESTAMP;
SET &dauer=&nachher - &vorher;
DISPLAY FORM 'Ausführungsdauer der INSERT-Anweisung', NL 1,
  'in Millisekunden:', &dauer;
```

2. Beispiel

```
DCL VAR &vorher1 TIMESTAMP(3),
        &vorher2 TIMESTAMP(3),
        &nachher1 TIMESTAMP(3),
        &nachher2 TIMESTAMP(3),
        &dauer1 INTERVAL FRACTIONS,
        &dauer2 INTERVAL FRACTIONS,
        &lauf SMALLINT,
        &SCHLUESSEL CHAR(6),
        ARTNAM CHAR(10);
DCL C1 CURSOR FOR S SCHLUESSEL, ARTNAM FROM FIRMA;
DCL C2 CURSOR PREFETCH 200
          FOR S SCHLUESSEL, ARTNAM FROM FIRMA;
DCL VAR &V LIKE CURSOR C1;
CYCLE FOR &lauf = 1 TO 10;
  SET &vorher1 = CURRENT TIMESTAMP;
  CYCLE C1 INTO &V.*;
  END CYCLE;
  SET &nachher1 = CURRENT TIMESTAMP,
```

```
  SET &dauer1 = &dauer1 + (&nachher1 - vorher1);
  END CYCLE;
SET &dauer1 = &dauer1/10;
CYCLE FOR &lauf = 1 TO 10;
  SET &vorher2 = CURRENT TIMESTAMP;
  CYCLE C2 INTO &V.*;
  END CYCLE;
  SET &nachher2 = CURRENT TIMESTAMP,
  SET &dauer2 = &dauer2 + (&nachher2 - vorher2);
END CYCLE;
SET &dauer2 = &dauer2/10;
DISPLAY FORM 'mittlere Abarbeitungsdauer für Cursortabelle', NL1,
  'mit ', &anzahl, 'Sätzen der Länge 16', NL1,
' a) ohne PREFETCH: ', &dauer1, 'Millisekunden', NL1,
' b) mit PREFETCH 200: ', &dauer2, 'Millisekunden';
```

5 Syntaxübersicht

5.1 Anweisungen

ALTER TABLE - Basistabelle ändern

```
ALTER TABLE tabelle

 { ADD [ COLUMN ] spaltendefinition |

   ALTER [ COLUMN ] spalte
         { DROP DEFAULT |
           SET grunddatentyp |
           SET voreinstellung } |

   ADD [ CONSTRAINT integritätsbedingungsname ] tabellenbedingung |

   DROP CONSTRAINT integritätsbedingungsname RESTRICT }



 voreinstellung::= DEFAULT { literal |
                   CURRENT DATE |
                   CURRENT TIME |
                   CURRENT TIMESTAMP |
                   [ CURRENT ] USER | SYSTEM USER |
                   NULL }
```

CLOSE - Cursor schließen

CLOSE cursor

COMMIT WORK -Transaktion beenden

COMMIT [WORK] [WITH { display | send message | stop }]

CREATE SCHEMA - Schema erzeugen

CREATE SCHEMA

{ [catalog .] schema [AUTHORIZATION berechtigungsschlüssel] |
AUTHORIZATION berechtigungsschlüssel }

[{ create_table_definition |
create_view_definition |
grant_definition } ...]

CREATE TABLE - Basistabelle erzeugen

CREATE TABLE tabelle

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

[USING SPACE space]

CREATE TEMPORARY VIEW - Temporären View deklarieren

```
CREATE TEMPORARY VIEW temp_viewname [ ( { spalte },... ) ]
   AS abfrageausdruck
```

CREATE VIEW - View erzeugen

```
CREATE VIEW tabelle [ ( spalte,... ) ]
      AS abfrageausdruck
      [ WITH CHECK OPTION ]
```

DECLARE - Cursor vereinbaren

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


      cursorbeschreibung::=

            abfrageausdruck

            [ ORDER BY { { spalte | spalte(posnr) | spaltennummer }
            [ { ASCENDING | DESCENDING } ] },... ]

            [ FOR UPDATE [ OF { spalte },... ] ]


      n::= vorzeichenlose_ganzzahl

      posnr::= vorzeichenlose_ganzzahl

      spaltennummer::= vorzeichenlose_ganzzahl
```

DELETE - Sätze löschen

DELETE FROM tabelle [WHERE { bedingung | CURRENT OF cursor }]

DROP CURSOR - Cursor Deklaration freigeben

DROP { CURSOR cursor | CURSORS }

DROP SCHEMA -Schema löschen

DROP SCHEMA [catalog .] schema RESTRICT

DROP TABLE - Basistabelle löschen

DROP TABLE tabelle RESTRICT

DROP TEMPORARY VIEW - Temporären View löschen

DROP TEMPORARY { VIEW tabelle | VIEWS }

DROP VIEW - View löschen

```
DROP VIEW tabelle RESTRICT
```

FETCH - Cursor positionieren und Satz lesen

```
FETCH [ { NEXT | PRIOR | FIRST | LAST | RELATIVE n | ABSOLUTE n  } ]
      [ FROM ] cursor

      [ INTO  { variable },... ]

      n::=  { [ { + | - } ] vorzeichenlose_ganzzahl | variable }
```

GRANT - Privilegien vergeben

GRANT-Format für Tabellen- und Spalten-Privilegien:

```
GRANT { ALL PRIVILEGES | { tabellen_und_spalten_privileg },... }

ON [ TABLE ] tabelle

TO { PUBLIC | { berechtigungsschlüssel },... }

[ WITH GRANT OPTION ]
tabellen_und_spalten_privileg::= { SELECT | DELETE | INSERT |
                          UPDATE [ ( { spalte },... ) ] |
                          REFERENCES [ ( { spalte },... ) ]
```

GRANT-Format für Sonder-Privilegien:

```
GRANT { ALL SPECIAL PRIVILEGES | CREATE SCHEMA }

ON CATALOG catalog

TO { PUBLIC | { berechtigungsschlüssel },... }

[ WITH GRANT OPTION ]
```

INSERT - Sätze in Tabelle einfügen

```
INSERT INTO tabelle

  { [ ( { spalte | spalte(posnr) | spalte(min-max) },... ) ]

    { VALUES ( { sqlausdruck | DEFAULT | NULL | * },... ) |
      VALUES { sqlausdruck | DEFAULT | NULL | * } |
      abfrageausdruck } |

    DEFAULT VALUES }


      [ RETURN INTO variable ]
```

OPEN - Cursor öffnen

```
OPEN cursor
```

PERMIT - Benutzeridentifikation für Old-Style angeben

```
PERMIT SCHEMA = { tabelle | variable } [ PASSWORD = wert ]
```

PRAGMA - Pragmaklauseln vereinbaren

```
PRAGMA literal

literal::= '{ pragma_klausel },...'

pragma_klausel::= { PREFETCH n |
                  EXPLAIN INTO datei |
                  IGNORE INDEX indexname |
                  OPTIMIZATION LEVEL n |
                  SIMPLIFICATION { ON | OFF } |
                  ISOLATION LEVEL
                     { READ UNCOMMITTED |
                       READ COMMITTED |
                       REPEATABLE READ |
                       SERIALIZABLE } |
                  DATA TYPE OLDEST |
                  CHECK { ON | OFF } }
```

RESTORE - Cursor wiederherstellen

```
RESTORE cursor
```

REVOKE - Privilegien entziehen

REVOKE-Format für Tabellen- und Spalten-Privilegien:

```
REVOKE { ALL PRIVILEGES | { tabellen_und_spalten_privileg },... }

ON [ TABLE ] tabelle

FROM { PUBLIC | { berechtigungsschlüssel },... } RESTRICT

tabellen_und_spalten_privileg::= { SELECT |
                                   DELETE |
                                   INSERT |
                                   UPDATE [ ( { spalte },...) ] |
                                   REFERENCES [ ( { spalte },... ) ] }
```

REVOKE-Format für Sonder-Privilegien:

```
REVOKE { ALL SPECIAL PRIVILEGES | CREATE SCHEMA }

ON CATALOG catalog

FROM { PUBLIC | { berechtigungsschlüssel },... }  RESTRICT
```

ROLLBACK WORK - Transaktion zurücksetzen

```
ROLLBACK [ WORK ] [ WITH RESET ]
```

SELECT - Einzelnen Satz lesen

SELECT [{ ALL | DISTINCT }] select_liste

[INTO { variable },...]

FROM tabellenangabe,...

[WHERE bedingung]

[GROUP BY spalte,...]

[HAVING bedingung]

SET CATALOG - Datenbanknamen voreinstellen

SET CATALOG voreingest_catalog

voreingest_catalog::= { alphanumerisches_literal | :variable }

SET SCHEMA - Schemanamen voreinstellen

SET SCHEMA voreingest_schema

voreingest_schema::= { alphanumerisches_literal | variable }

SET SESSION AUTHORIZATION - Berechtigungsschlüssel festlegen

SET SESSION AUTHORIZATION neuer_berechtigungsschlüssel

neuer_berechtigungsschlüssel::= { alphanumerisches_literal | variable }

SET TRANSACTION - Transaktionseigenschaften festlegen

SET TRANSACTION { level [[,] transaktionsmodus] |
transaktionsmodus [[,] level] }

level::= { ISOLATION LEVEL { READ UNCOMMITTED | READ COMMITTED |
REPEATABLE READ | SERIALIZABLE } |
CONSISTENCY LEVEL konsistenzlevel }

transaktionsmodus::= { READ ONLY | READ WRITE }

STORE - Cursorposition speichern

STORE cursor

UPDATE - Spaltenwerte ändern

```
UPDATE tabelle SET {
        { spalte | spalte(posnr) | spalte(min-max) }
         = { sqlausdruck | DEFAULT | NULL }
             },...
    [ WHERE { bedingung | CURRENT OF cursor } ]
```

WHENEVER - Fehlerbehandlung definieren

```
WHENEVER &DML_STATE [ IN ( status,... ]

            { CONTINUE | CALL subprogname | BREAK }
```

5.2 Metavariablen

abfrageausdruck

abfrageausdruck::= { select_ausdruck | join_ausdruck | (abfrageausdruck) }
[UNION [ALL] abfrageausdruck]

bedingung

bedingung::= { [bedingung AND] { [NOT] { praedikat | (bedingung) } } |
bedingung OR { [NOT] { { praedikat | (bedingung) } } }

join_ausdruck

join_ausdruck::=
{ tabellenangabe [{ INNER | { LEFT | RIGHT | FULL } [OUTER] }]
JOIN tabellenangabe ON bedingung |
(join_ausdruck) }

literal

literal::=

{ charliteral | numliteral | datumzeitliteral }

charliteral::= 'string'

string::= [zeichen] ...

numliteral::= { ganzzahl | festpunktzahl | gleitpunktzahl | $PI }

ganzzahl::= [{ + | - }] vorzeichenlose_ganzzahl

festpunktzahl::= [{ + | - }] vorzeichenlose_ganzzahl [.vorzeichenlose_ganzzahl]

gleitpunktzahl::= festpunktzahl E [{ + | - }] vorzeichenlose_ganzzahl

vorzeichenlose_ganzzahl::= ziffer...

datumzeitliteral::=

```
{ DATE (jahr-monat-tag) |
  TIME (stunde:minute:sekunde [.bruchteil ] ) |
  TIMESTAMP (jahr-monat-tag stunde:minute:sekunde [.bruchteil ] ) }
```

mengenfunktion

```
mengenfunktion::= { { AVG | COUNT | MAX | MIN | SUM }
                    ( [ { ALL | DISTINCT } ] sqlausdruck ) |

                  COUNT(*) }
```

praedikat

praedikat::=

{ sqlausdruck { = | < | > | <= | >= | <> } sqlausdruck |

sqlausdruck { = | < | > | <= | >= | <> } { ANY | SOME | ALL } unterabfrage |

sqlausdruck [NOT] BETWEEN sqlausdruck AND sqlausdruck |

sqlausdruck [NOT] IN { unterabfrage | (sqlausdruck,sqlausdruck,...) } |

[tabelle .] { spalte | spalte(posnummer) | spalte(min-max) } [NOT] LIKE
muster [ESCAPE zeichen] |

[tabelle .] { spalte | spalte(posnummer) | spalte(min-max) }
IS [NOT] NULL |

EXISTS unterabfrage }

select_ausdruck

select_ausdruck::=

SELECT [{ ALL | DISTINCT }] select_liste

FROM tabellenangabe,...

[WHERE bedingung]

[GROUP BY spalte,...]

[HAVING bedingung]

select_liste ::= { * | { tabelle.* | sqlausdruck [[AS] spalte] },... }

spaltenbedingung

spaltenbedingung::=

{ NOT NULL |
UNIQUE |
PRIMARY KEY |
CHECK (bedingung) |
REFERENCES tabelle [(spalte)]

spaltendefinition

spaltendefinition::=

spalte [(dimension)] { grunddatentyp | FLOAT (stellenanzahl) }
[voreinstellung]

[[CONSTRAINT integritätsbedingungsname] spaltenbedingung ...]

dimension::= vorzeichenlose_ganzzahl

voreinstellung::= DEFAULT { literal |
CURRENT DATE |
CURRENT TIME |
CURRENT TIMESTAMP |
[CURRENT] USER |
SYSTEM USER |
NULL }

sqlausdruck

```
sqlausdruck::=
        { wert |
         [ tabelle. ] { spalte | spalte(posnummer) | spalte(min-max) } |
         { + | - } sqlausdruck |
         sqlausdruck { * | / | + | - | || } sqlausdruck |
         ( sqlausdruck ) |
         unterabfrage |
         mengenfunktion |
         zeitfunktion |
         [ CURRENT ] USER | SYSTEM USER }


  posnummer::= vorzeichenlose_ganzzahl

  min::= vorzeichenlose_ganzzahl

  max::= vorzeichenlose_ganzzahl
```

tabellenangabe

```
tabellenangabe::= { tabelle [ [ AS ] korrelationsname [ ( spalte,... ) ] ] |
                    unterabfrage [ AS ] korrelationsname [ ( spalte,... ) ] |
                    join_ausdruck }


tabelle::= { [ [ catalog. ] einf_schemaname. ] einf_basistabellenname |
           [ [ catalog. ] einf_schemaname. ] einf_viewname |
             temp_viewname }
```

tabellenbedingung

tabellenbedingung::=

```
{ UNIQUE ( { spalte,... } ) |
  PRIMARY KEY ( { spalte,... } ) |
  FOREIGN KEY ( { spalte,... } ) REFERENCES tabelle [ ( { spalte },... ) ] |
  CHECK ( bedingung )
```

unterabfrage

unterabfrage::= (abfrageausdruck)

variable

variable::= { &vamame1 [suffix] |

&varname2 { (index1, index2) |

(index1, bereich2) |

(bereich1, index2) }

suffix::= { gruppenkomponente | indexkomponente }

gruppenkomponente::= . { * | komponente [suffix] }

indexkomponente::= { ({ index | bereich }) }

index::= vorzeichenlose_ganzzahl

bereich::= index1 - index2

wert

wert::= { literal | variable | aggregat }

aggregat::= < { wert | NULL }, ... >

zeitfunktion

zeitfunktion::= { CURRENT DATE | CURRENT TIME | CURRENT TIMESTAMP}

6 Fehlermeldungen

Dieses Kapitel beschreibt

– die Abbildung der SQLCODEs von SESAM V2 auf den &DML_STATE von DRIVE/WINDOWS. Diese Abbildung ist kompatibel zu SESAM V1 bzw. DRIVE/WINDOWS V1.1 und DRIVE V6.x

– die für DRIVE/WINDOWS relevanten SQLSTATE-Klassen von SESAM V2.

Detaillierte Hinweise zur Fehlerbehandlung finden Sie bei der Anweisung WHENEVER, im Abschnitt „Pragmas“ auf Seite 62 und in der DRIVE-Programmiersprache [2], Abschnitt "Fehler abfangen, Endekriterien".

6.1 Abbildung der SQLCODEs von SESAM auf &DML_STATE

In diesem Abschnitt werden die SQLCODEs von SESAM V2 aufwärtskompatibel auf den &DML_STATE von DRIVE/WINDOWS abgebildet und die zugeordneten SQLSTATEs aufgeführt.

SQLCODEs aus der Sprachdefinition

Bedeutung	**SQLCODE**	**&DML_STATE**	**SQLSTATEs**
table-end-reached	100	TABLE END	02000
warning	50	OK	01xxx
dirty-read	10	DIRTY READ	01SA1
sql-ok	0	OK	00000
access-right-conflict	-22	SQL ERROR	42SQC
syntax-errror	-110	<drive sys error>	42xxx (36 mögliche SQLSTATEs)
schema-name-ambiguous	-118	SQL ERROR	42SND, 42SNN
table-not-primitive	-121	SQL ERROR	42S01, 42SQJ

Bedeutung	SQLCODE	&DML_STATE	SQLSTATEs
table-not-defined	-127	SQL ERROR	42SH3, 42SNI, 42SQ6, 42SQK
table-name-ambiguous	-128	SQL ERROR	42SA2, 42SA4, 42SQI
col-or-comp-error	-131	SQL ERROR	42SOG, 42SP3, 42SR6
col-or-comp-spec-error	-135	SQL ERROR	42SAK, 42SAS, 42SO2, 42SQ2, 42SQ4, 42SQ5
col-or-comp-not-defined	-137	SQL ERROR	42SNF, 42SNG, 42SNM, 42S00
col-or-comp-ambiguous	-138	SQL ERROR	42SA1, 42SI8, 42SI9, 42SN5
cursor-already-closed	-141	SQL ERROR	24SA2
cursor-already-open	-142	SQL ERROR	24SA1, 24SA6
cursor-not-positioned	-143	SQL ERROR	24SA3
cursor-position-error	-144	CURSOR SQL ERROR	24SA5
cursor-not-defined	-147	SQL ERROR	34SA1, 42SF1
cursor-name-ambiguous	-148	SQL ERROR	42SF0
null-error	-210	SQL ERROR	23SA3, 23SA4
unique-error	-220	SQL ERROR	23SA2, 23SA5
no-indicator-variable	-310	SQL ERROR	22002
more-than-one-hit	-320	SQL ERROR	21000
value-list-error	-330	SQL ERROR	42SAT, 42SOC, 42SQH
value-error	-335	SQL ERROR	22023, 22SA2, 42SC9
set-function-not-allowed	-338	SQL ERROR	42SBR, 42SNH, 42SNO
value-overflow	-340	SQL ERROR	22001, 22003, 22012
type-error	-345	SQL ERROR	07SA6, 22019, 42SAX, 42SAY, 42SBO, 42SBX, 42SBY, 42SN2, 42S0Y
type-mismatch	-350	SQL ERROR	22005, 42SAW, 42SBS, 42SQ7, 42SQ9, 42SR1
argument-error	-365	SQL ERROR	42SAA
like-error	-370	SQL ERROR	22025, 42SAL, 42SAM, 42SBP

Bedeutung	**SQLCODE**	**&DML_STATE**	**SQLSTATEs**
default-value-not-allowed	-372	SQL ERROR	22SA3
search-cond-error	-380	SQL ERROR	42SBU
view-column-list-missing	-390	SQL ERROR	42SOF, 42SR3
group-restriction-error	-420	SQL ERROR	42SNK
into-clause-error	-440	SQL ERROR	42SCP, 42SQT
error-in-sql-stmt-processing	-550	SQL ERROR	42xxx (weit über 200 mögliche SQLSTATEs)
error-in-dml-stmt-processing	-560	SQL ERROR	22018, 23SA0, 23SA1, 28000, 3D000, 3F000, 42SH4...42SH8, 42SNL, 44000
stmt-not-allowed	-600	SQL ERROR	24SA4, 25SA2, 25SA4
fetch-orientation-error	-630	SQL ERROR	42SF2
syntax-value-violated	-650	SQL ERROR	22020, 22021, 42SA8, 42SAB, 42SAD, 42SAE, 42SAH, 42SBW, 42SF3, 42SF4, 42SL0...42SL6, 42SM1...42SM4, 42SN3, 42SOE, 42SOP, 42SR9

Systembedingte SQLCODEs

<session cancelled> bezeichnet den internen &DML_STATE-Eintrag 'SESSION CANCELLED'.

<db not available> bezeichnet den internen &DML_STATE-Eintrag 'DB NOT AVAILABLE'.

<drive sys error> bezeichnet einen Systemfehler von DRIVE/WINDOWS. In diesem Fall gibt DRIVE/WINDOWS die Fehlermeldung DRI0078 aus und erstellt Diagnoseunterlagen.

Kurzfristiges Hindernis

Bedeutung	SQLCODE	&DML_STATE	SQLSTATEs
temp-access-restriction	-700	TEMP SYS ERROR	81SC9
stmt-cancelled	-701	TEMP SYS ERROR	81SA2, 81SD2
sort-option-restriction	-702	TEMP SYS ERROR	81SA3, 81SS0...81SS5, 91SCF, 91SCG
temp-system-limit	-710	TEMP SYS ERROR	81SC7, 91SA0, 91SA2, 91SA4, 91SA6, 91SA8, 91SA9, 91SAA, 91SAB, 91SAC, 91SAG, 91SAJ, 91SAL, 91SC5, 91SC6, 91SC7, 91SCB, 91SCC, 91SCD, 91SCE, 91SCH, 91SCJ, 91SCK, 91SS0...91SS7

Unverträglichkeiten innerhalb des SQL-Schemas

Bedeutung	SQLCODE	&DML_STATE	SQLSTATEs
db-open-error	-775	ACC SYS ERROR	42SC7, 42SN7, 55SA1, 55SAA, 55SAE, 81SA4, 81SA5

Administrator-Eingriff erforderlich

Bedeutung	SQLCODE	&DML_STATE	SQLSTATEs
session-unknown	-740	<session cancelled>	81SP2
access-restriction	-800	ADMIN SYS ERROR	81SA6, 81SB2, 81SCA
update-restriction	-810	ADMIN SYS ERROR	25SA1, 81SB0, 81SB1
system-abnormally-down	-830	<db not available>	81SB5, 81SC3
component-not-available	-840	ACC SYS ERROR	81SC6
transaction-start-not-allowed	-850	ADMIN SYS ERROR	81SA1, 81SP3
configuration-file-error	-860	ACC SYS ERROR	81SB4, 81SC0, 81SC1

Programmierfehler im DB-System oder an der DB-Schnittstelle

Bedeutung	SQLCODE	&DML_STATE	SQLSTATEs
ta-stmt-not-allowd	-654	<drive sys error>	42SH2
program-error	-900	<session cancelled>	42SC1, 55SA7, 81SC4, 81SC5
integrity-error	-910	ACC SYS ERROR	81SA7, 81SA8, 81SA9, 91SAD
system-limit-exceeded	-920	ACC SYS ERROR	91SA1, 91SA7, 91SAE, 91SAF, 91SAH, 91SAI, 91SAK, 91SB1...91SB4, 91SC0...91SC4, 91SC8, 91SC9, 91SCA, 91SCI, 91SR0, 91SR2, 91SR3, 91SU0...91SU4
not-implemented	-940	SQL ERROR	42SS0
illegal-order-of-state-ments	-990	SQL ERROR	25SA3, 25SA5
representation-error	-990	<drive sys error>	-
output-too-long	-990	<drive sys error>	-

Internes Rücksetzen der Transaktion ("CANCEL WORK")

Wird die Transaktion vom Datenbanksystem intern zurückgesetzt, so wird ein interner SQL-CODE zwischen -1000 und -2000 erzeugt, indem auf den eigentlichen SQLCODE -1000 addiert wird. In diesem Fall generiert DRIVE/WINDOWS den internen &DML_STATE-Eintrag 'TA CANCELLED' und verhält sich danach wie bei einem externen ROLLBACK WORK WITH RESET. Eine Ausnahme bildet der SQLCODE -1830, bei dem DRIVE/WINDOWS den internen &DML_STATE-Eintrag 'DB NOT AVAILABLE' generiert.

Fehler bei dynamisch ausgeführten SQL-Anweisungen (EXECUTE)

Fehler in dynamischen SQL-Anweisungen (SQLSTATE-Klasse 07) führen in der Regel zu DRIVE-Systemfehlern, weil DRIVE/WINDOWS diese Anweisungen nicht an der Oberfläche anbietet, sondern sie nur intern im Rahmen der DRIVE-EXECUTE-Anweisung erzeugt.

Bedeutung	**SQLCODE**	**&DML_STATE**	**SQLSTATEs**
undefined-descriptor-area-item	100	<drive sys error>	02SA1
error-in-dynamic-stmt	-650	<drive sys error>	-
invalid-stmt-identifier	-651	<drive sys error>	-
exec-or-open-not-possible	-652	<drive sys error>	-
error-in-using-clause	-653	<drive sys error>	-
descriptor-area-name-invalid	-661	<drive sys error>	-
allocated-descriptor-area	-662	<drive sys error>	-
not-allocated-descriptor-area	-663	<drive sys error>	-
error-in-descriptor-area-item	-664	<drive sys error>	-
configuration-error	-690	<drive sys error>	-
session-limit-exceeded	-950	<drive sys error>	-
too-many-host-variables	-960	<drive sys error>	-
error-in-descriptor-area-size	-965	LIMIT REACHED	91SA3, 91SA5

6.2 SQLSTATE-Klassen

Die Abbildung von SQLSTATEs auf &DML_STATE-Einträge ist bei der WHENEVER-Anweisung beschrieben. Die folgenden SQLSTATE-Klassen von SESAM/SQL V2 sind für DRIVE/WINDOWS relevant:

&SQL_STATE	**Bedeutung**
00xxx	erfolgreiche Ausführung
01xxx	Warnung
02xxx	keine Daten
21xxx	Verstoß gegen Mengenbeschränkung
22xxx	Datenfehler
23xxx	Integritätsbedingungen verletzt
24xxx	Cursor-Zustand oder -Operation unzulässig
25xxx	Transaktionszustand unzulässig
26xxx	SQL-Anweisungsname unzulässig oder ungültig
28xxx	Berechtigungsschlüssel unzulässig
2Dxxx	Art der Transaktionsbeendigung unzulässig
34xxx	Cursorname unzulässig oder ungültig
3Dxxx	Katalogname unzulässig
3Fxxx	Schemaname unzulässig
40xxx	Transaktion zurückgesetzt
42xxx	Syntaxfehler oder kein Zugriffsrecht Hinweis: Diese Klasse enthält über 300 Unterklassen.
44xxx	"CHECK OPTION" verletzt
55xxx	Fehlermeldungen des BS2000
56xxx	Einschränkungen des BS2000
81xxx	Fehler in der Umgebung
91xxx	Ressourcenmangel
95xxx	Fehlerhafter Transaktionszustand

Das Handbuch SESAM/SQL-Server "Meldungen" [24] enthält eine Gegenüberstellung aller SQLSTATEs mit zugeordneten SQLCODEs.

Literatur

[1] **DRIVE/WINDOWS** (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** (BS2000)
Programmiersprache
Sprachbeschreibung

Zielgruppe
Anwendungsprogrammierer
Inhalt
Beschreibung der Programmerstellung einschließlich Alpha-Bildschirmformaten sowie Listenformaten mit DRIVE und Report-Generator.

[3] **DRIVE/WINDOWS** (BS2000)
Lexikon der DRIVE-Anweisungen
Referenzhandbuch

Zielgruppe
Anwendungsprogrammierer
Inhalt
Syntax und Funktionsumfang aller DRIVE-Anweisungen. Meldungen und Schlüsselwörter von DRIVE/WINDOWS

[4] **DRIVE/WINDOWS** (BS2000/SINIX)
Lexikon der DRIVE-SQL-Anweisungen für SESAM V1.x
Referenzhandbuch

Zielgruppe
Anwendungsprogrammierer
Inhalt

Syntax und Funktionsumfang aller DRIVE-SQL-Anweisungen für SESAM V1.x in Kurzform.

[5] **DRIVE/WINDOWS** (BS2000/SINIX)
Lexikon der DRIVE-SQL-Anweisungen für SESAM V2.x
Referenzhandbuch

Zielgruppe
Anwendungsprogrammierer
Inhalt
Syntax und Funktionsumfang aller DRIVE-SQL-Anweisungen für SESAM V2.x in Kurzform.

[6] **DRIVE/WINDOWS** (BS2000/SINIX)
Lexikon der DRIVE-SQL-Anweisungen für UDS
Referenzhandbuch

Zielgruppe
Anwendungsprogrammierer
Inhalt
Syntax und Funktionsumfang aller DRIVE-SQL-Anweisungen für UDS in Kurzform.

[7] **DRIVE/WINDOWS V2.0** (MS-Windows)
Software-Produktionsumgebung (SPU)
Benutzerhandbuch

Zielgruppe
Anwendungsprogrammierer
Inhalt
Erläuterung der Funktionen der Software-Produktionsumgebung (Arbeitsplatz). Einsatzvorbereitung für DRIVE/WINDOWS, den Remote-Zugriff auf BS2000- und SINIX-Datenbanken und für Client-Server-Anwendungen.

[8] **DRIVE/WINDOWS V2.0** (MS-Windows)
Programmiersprache
Sprachbeschreibung

Zielgruppe
Anwendungsprogrammierer
Inhalt
Beschreibung der Programmerstellung einschließlich Fenster-Client-Server-Anwendungen.

[9] **DRIVE/WINDOWS V2.0** (MS-Windows)
Lexikon der DRIVE-Anweisungen
Referenzhandbuch

Zielgruppe
Anwendungsprogrammierer

Inhalt
Syntax und Funktionsumfang aller DRIVE-Anweisungen, Meldungen und Schlüsselwörter von DRIVE/WINDOWS.

[10] **DRIVE/WINDOWS** (SINIX)
Software-Produktionsumgebung (SPU)
Benutzerhandbuch

Zielgruppe
Anwendungsprogrammierer
Inhalt
Erläuterung der Funktionen der Software-Produktionsumgebung (Arbeitsplatz) und des Expertenmodus. Einsatzvorbereitung für Remote-Zugriff auf BS2000-Datenbanken, für das Erstellen von Anwendungen für das BS2000 und für DRIVE/WINDOWS allgemein.

[11] **DRIVE/WINDOWS** (SINIX)
Programmiersprache
Sprachbeschreibung

Zielgruppe
Anwendungsprogrammierer
Inhalt
Beschreibung der Programmerstellung einschließlich Grafik- und Alpha-Bildschirmformaten sowie Listenformaten mit DRIVE und Report Generator.

[12] **DRIVE/WINDOWS** (SINIX)
Lexikon der DRIVE-Anweisungen
Referenzhandbuch

Zielgruppe
Anwendungsprogrammierer
Inhalt
Syntax und Funktionsumfang aller DRIVE-Anweisungen. Meldungen und Schlüsselwörter von DRIVE.

[13] **DRIVE/WINDOWS** (BS2000/SINIX)
Lexikon der DRIVE-SQL-Anweisungen für INFORMIX
Referenzhandbuch

Zielgruppe
Anwendungsprogrammierer
Inhalt
Syntax und Funktionsumfang aller DRIVE-SQL-Anweisungen für INFORMIX in Kurzform.

[14] **DRIVE V5.1** (BS2000)
Teil 1: Benutzerhandbuch

Zielgruppe
– DV-Laien in der Fachabteilung
– Anwendungsprogrammierer

Inhalt
– Allgemeiner Überblick über das System DRIVE in OLD-Style
– Erläuterung der DRIVE-Komponenten
– Beschreibung möglicher Anwendungsfälle anhand einführender Beispiele
– Generierung und Administration von DRIVE im UTM-Betrieb

[15] **DRIVE V5.1** (BS2000)
Teil 2: LEXIKON

Zielgruppe
– DV-Laien in der Fachabteilung
– Anwendungsprogrammierer

Inhalt
– Syntax und Funktionsumfang aller DRIVE-Anweisungen in OLD-Style
– Meldungen und Schlüsselwörter von DRIVE

[16] **DRIVE/WINDOWS-COMP** (BS2000)
Benutzerhandbuch

Inhalt
Beschreibung der Sprachabweichungen zu DRIVE/WINDOWS V1.1 und Darstellung des Compilierungsvorganges. Beschreibung des Generierens und Startens von Anwendungen von compilierten DRIVE-Objekten (TIAM- und UTM-Betrieb) unter besonderer Berücksichtigung des Versionsmischbetriebes.

[17] **SQL für SESAM/SQL**
Sprachbeschreibung

Zielgruppe
Programmierer, die mit SQL-Anweisungen auf SESAM-Datenbanken zugreifen wollen.

Inhalt
SQL-Anweisungen für den Zugriff auf SESAM-Datenbanken.

[18] **SESAM/SQL-Server** (BS2000/OSD)
SQL-Sprachbeschreibung Teil 1: SQL-Anweisungen
Benutzerhandbuch

Zielgruppe
Zur Zielgruppe gehören alle Personen, die eine SESAM/SQL-Datenbank mit SQL-Anweisungen bearbeiten.

Inhalt

Das Handbuch beschreibt die Programmeinbettung von SQL-Anweisungen und die SQL-Sprachelemente. In einem alphabetischen Nachschlageteil sind alle SQL-Anweisungen ausführlich dargestellt.

[19] **SESAM/SQL-Server** (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.

[20] **SESAM/SQL-Server** (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 anderen SESAM/SQL-Handbücher.

[21] **SESAM/SQL-Server** (BS2000/OSD)
Utility-Monitor
Benutzerhandbuch

Zielgruppe
Das Handbuch ist für den Datenbankverwalter und 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.

[22] **SESAM/SQL-Server** (BS2000/OSD)
Umstellen von SESAM-Datenbanken u. -Anwendungen auf SESAM/SQL-Server
Benutzerhandbuch

Zielgruppe

Das Handbuch richtet sich an alle, die sich für SESAM/SQL-Server ab der Version 2.0 interessieren.
Inhalt
Dieses Handbuch beschreibt die neuen Konzepte und Funktionen im Überblick. Im Vordergrund steht der Bezug zu Vorgängerversionen, um bisherigen SESAM/SQL-Anwendern den Umstieg in die neue Welt von SESAM/SQL-Server zu erleichtern.

[23] **SESAM/SQL-Server** (BS2000/OSD)
CALL-DML Anwendungen
Benutzerhandbuch

Zielgruppe
Programmierer von SESAM-Anwendungen
Inhalt
- CALL-DML-Anweisungen für die Bearbeitung von SESAM-Datenbanken mittels Anwenderprogrammen
- Teilhaberbetrieb mit UTM und DCAM
- Dienstprogramme SEDI61 und SEDI63 zur Datenwiedergewinnung und Direktänderung
- Hinweise zum Mischbetrieb von CALL-DML- und SQL-Modus

[24] **SESAM/SQL-Server** (BS2000/OSD)
Meldungen
Benutzerhandbuch

Zielgruppe
Zur Zielgruppe gehören alle SESAM/SQL-Anwender.
Inhalt
Das Handbuch enthält sämtliche Meldungen zu SESAM/SQL nach Meldungsnummern sortiert.

[25] **SQL für UDS/SQL**
Sprachbeschreibung

Zielgruppe
Programmierer, die mit SQL-Anweisungen auf UDS-Datenbanken zugreifen wollen.
Inhalt
SQL-Anweisungen für den Zugriff auf UDS-Datenbanken.

[26] **UDS/SQL** (BS2000)
Verwalten und Bedienen
Benutzerhandbuch

Zielgruppe
Datenbankadministrator
Inhalt

– Verwaltungs- und Bedienungsarbeiten wie Sichern der Datenbank,
– Datenbankbetrieb
– Ausgeben von Datenbankinformationen
– Reorganisieren der DatenbankDatenbankinformationen,
– Konsistenzprüfprogramm

Einsatz
Datenbankadministration im laufenden Betrieb

[27] **UDS/SQL** (BS2000)
Aufbauen und Umstrukturieren
Benutzerhandbuch

Zielgruppe
Datenbankadministrator
Inhalt
– Übersicht über die von UDS benötigten Dateien
– UDS-Dienstprogramme, die zum Aufbauen der UDS-Datenbank nötig sind
– Dienstprogramme zum Umstrukturieren

Einsatz
Datenbankadministrator beim Aufbauen einer Datenbank

[28] **IFG für FHS** (TRANSDATA)
Benutzerhandbuch

Zielgruppe
Datenstationsbenutzer, Anwendungsdesigner und Programmierer
Inhalt
Der Interaktive Formatgenerator (IFG) ist ein System zur komfortablen und einfachen Erstellung und Verwaltung von Formaten an Datensichtstationen. Diese Formate können zusammen mit FHS im Verarbeitungsrechner eingesetzt werden. Das Benutzerhandbuch beschreibt, wie die Formate erstellt, geändert und verwaltet werden sowie die neuen Funktionen von IFG V8.1.

[29] **FHS** (TRANSDATA)
Benutzerhandbuch

Zielgruppe
Programmierer
Inhalt
Programmschnittstellen von FHS für TIAM-, DCAM- und UTM-Anwendungen. Erstellen, Einsatz und Verwalten von Formaten.

[30] **UTM** (BS2000/OSD)
Anwendungen generieren und administrieren
Benutzerhandbuch

Zielgruppe
Organisierer, Einsatzplaner und Administratoren von UTM-Anwendungen.
Inhalt
– Installation von UTM
– Einrichten, Bedienen und Verwalten von UTM-Anwendungen
– UTM-Benutzerkommandos

[31] **UTM** (TRANSDATA)
Anwendungen programmieren
Benutzerhandbuch

Zielgruppe
Programmierer von UTM-Anwendungen
Inhalt
– Sprachunabhängige Beschreibung der Programmschnittstelle KDCS,
– Aufbau von UTM-Programmen
– KDCS-Aufrufe
– Testen von UTM-Anwendungen
– Alle Informationen, die der Programmierer von UTM-Anwendungen benötigt
Einsatz
BS2000-Transaktionsbetrieb

[32] **UTM** (SINIX)
Formatierungssystem

Zielgruppe
UTM(SINIX)-Anwender, die mit Formaten arbeiten wollen, C-Programmierer und COBOL-Programmierer
Inhalt
Einsetzen der Formatsteuerung FORMANT in UTM(SINIX)-Teilprogrammen, Erstellen von Formaten, konvertieren von Formaten zwischen BS2000 und SINIX.

[33] **EDT** (BS2000/OSD)
Anweisungen
Benutzerhandbuch

Zielgruppe
EDT-Einsteiger und EDT-Anwender
Inhalt
Bearbeiten von SAM- und ISAM-Dateien und Elementen aus Programm-Bibliotheken und POSIX-Dateien.

[34] **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.

[35] **BS2000/OSD-BC**
Kommandos Band 1 - 3
Benutzerhandbuch

Zielgruppe
Die Handbücher wenden sich sowohl an den nichtprivilegierten Anwender als auch an die Systembetreuung.
Inhalt
Sie enthalten die BS2000/OSD-Kommandos (BS2000/OSD-Grundausbau und ausgewählte Produkte) mit der Funktionalität für alle Privilegien. Die Einleitung gibt Hinweise zur Kommandoeingabe.

[36] **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.

[37] **BS2000/OSD-BC V2.0**
Einführung in das DVS
Benutzerhandbuch

Zielgruppe
Das Handbuch wendet sich an den nichtprivilegierten Anwender und an die Systembetreuung.
Inhalt
Es beschreibt die Dateiverwaltung und -verarbeitung im BS2000.
Themenschwerpunkte:
- Datei- und Katalogverwaltung

- Dateien und Datenträger
- Datei- und Datenschutz
- OPEN-, CLOSE-, EOV-Verarbeitung
- DVS-Zugriffsmethoden (SAM, ISAM,...)

[38] BS2000
Einführung in die Systemanwendung
Benutzerhandbuch

Zielgruppe
BS2000-Anwender
Inhalt
– Einführung ins BS2000
– Beschreibung der meistgebrauchten Benutzerkommandos bis BS2000 V8.5A
– Einführung in die Benutzung der Dienstprogramme und Softwareprodukte EDT, SORT, ARCHIVE, TSOSLNK, LMS, PERCON
– Hinweise für den programmierenden Benutzer
Einsatz
BS2000-Dialogbetrieb und -Stapelbetrieb

[39] **FORMANT** (SINIX)
Beschreibung

Zielgruppe
– C-Programmierer
– COBOL-Programmierer
– Anwendungsplaner
Inhalt
FORMANT ist eine Maskensteuerung für alle SINIX-Systeme. Das Manual enthält:
– Einführung in FORMANT
– Beschreibung von FORMANTGEN
– Beschreibung der Bedienerschnittstelle
– Programmschnittstellen in C und COBOL
– Beispiele zur Programmierung

[40] **OMNIS** (TRANSDATA, BS2000)
Administration und Programmierung
Benutzerhandbuch

Zielgruppe
– OMNIS-Administrator
– Programmierer
Inhalt
Beschreibung der Grundlagen der Administration von OMNIS, der OMNIS-Dienstprogramme sowie der Anwendungsschnittstelle zur Erweiterung des Funktionsumfanges von OMNIS.

[41] **DRIVE/WINDOWS-COMP(SINIX)**
Compiler
Benutzerhandbuch
Zielgruppe
Anwendungsprogrammierer und Systemverwalter
Inhalt
Beschreibung des Compilierungsvorgangs durch den DRIVE-Compiler.

[42] **INFORMIX-NET** (SINIX)
INFORMIX-STAR (SINIX)
Benutzerhandbuch

Zielgruppe
INFORMIX-Benutzer und Systemverwalter
Inhalt
Das Handbuch beschreibt die Arbeit mit den INFORMIX-Netzprodukten INFORMIX-NET und INFORMIX-STAR.
Mit den INFORMIX-Netzprodukten können INFORMIX-Anwendungen von einem lokalen Rechner aus Datenbanken auf fernen Rechnern erstellen und bearbeiten.

[43] **DRIVE/WINDOWS** (SINIX)
Ergänzungsband
Benutzerhandbuch

Zielgruppe
Anwendungsprogrammierer
Inhalt
Der Ergänzungsband enthält die funktionalen Änderungen von DRIVE/WINDOXS (SINIX) V1.1. Die Handbücher der Version 1.0 werden benötigt.

[44] **SESAM/SQL-Server** (BS2000/OSD)
Meldungen
Benutzerhandbuch

Zielgruppe
Zur Zielgruppe gehören alle SESAM/SQL-Anwender.
Inhalt
Das Handbuch enthält sämtliche Meldungen zu SESAM/SQL nach Meldungsnummern sortiert.

[45] **SESAM/SQL-Server** (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.

[46] **SESAM/SQL** (BS2000)
Aufbau und Wartung
Benutzerhandbuch

Zielgruppe
Datenbank-Verwalter
Inhalt
- Aufbau und Wartung von SESAM-Datenbanken mit dem Datenbank-Administrationsmonitor SESASB
- Schattendatenbankbetrieb

Sonstige Literatur

[47] International Organization for Standardization (ISO):
Database Language SQL
ISO/IEC 9075:1992

Bestellen von Handbüchern

Die aufgeführten Handbücher finden Sie mit ihren Bestellnummern im *Druckschriftenverzeichnis* der Siemens Nixdorf Informationssysteme AG. Neu erschienene Titel finden Sie in den *Druckschriften-Neuerscheinungen*.

Beide Veröffentlichungen erhalten Sie regelmäßig, wenn Sie in den entsprechenden Verteiler aufgenommen sind. Wenden Sie sich bitte hierfür an Ihre zuständige Geschäftsstelle. Dort können Sie auch die Handbücher bestellen.

Stichwörter

C

O

R

S

Inhalt

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/SQL 2 in Kurzform. Kurze Einführung in das Arbeiten mit SESAM/SQL 2. Beispiel für die Migration einer SESAM V1-Datenbank in SESAM/SQL 2.

Ausgabe: Februar 1996

Datei: drv_ses2.PDF

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.