Einleitung

DRIVE/WINDOWS ermöglicht mit SQL-Anweisungen den Zugriff auf das Datenbanksystem UDS/SQL. Dieses Lexikon beschreibt in Kurzform die genaue Syntax der DRIVE-SQL-Anweisungen für UDS. Eine ausführliche Beschreibung der SQL-Anweisungen für UDS finden Sie im Handbuch "SQL für UDS/SQL V1.0 Sprachbeschreibung" [13].

Dieses Lexikon beschreibt die DRIVE-SQL-Anweisungen für SESAM in der DRIVE/WINDOWS-Version 1.1 für BS2000 und SINIX.

Komplexe Anweisungsteile, die sowohl in DRIVE- als auch in SQL-Anweisungen enthalten sind, werden gesondert beschrieben im Kapitel "Metavariablen" des Benutzerhandbuches "DRIVE-WINDOWS: Lexikon der DRIVE-Anweisungen" [3]. Davon abweichende DRIVE-SQL-Metavariablen werden in diesem Lexikon in einem eigenen Kapitel "UDS-Metavariablen" beschrieben. Der Begriff "Betriebsmodus mit TP-Monitor" bezeichnet im BS2000 den UTM-Betrieb.

SQL-Returncodes werden vom Datenbanksystem UDS/SQL übernommen und von DRIVE/WINDOWS als Fehlermeldung der Form DRI9xxx ausgegeben. Sie können nachgeschlagen werden im Handbuch "SQL für UDS/SQL (BS2000) V1.0: Sprachbeschreibung" [13].

UDS/SQL-Anweisungen

CLOSE Cursor schließen

CLOSE schließt einen geöffneten Cursor, den Sie mit der Anweisung DECLARE für einen Cursor vereinbart und mit OPEN geöffnet haben.

Die Deklaration des Cursors bleibt erhalten. Vor einem erneuten Zugriff auf den Cursor mit der Anweisung FETCH müssen Sie den Cursor mit OPEN oder RESTORE eröffnen. Ein geöffneter Cursor wird auch bei Transaktionsende geschlossen.

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

```
CLOSE cursor
```

cursor Name des Cursors, den Sie schließen wollen.

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

COMMIT WORK Transaktion beenden

COMMIT WORK beendet eine Transaktion und schreibt die während der Transaktion durchgeführten Änderungen in der Datenbank fest.
Die erste fehlerfreie SQL-Anweisung nach COMMIT WORK eröffnet 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.

Empfehlung:
Im DRIVE-Programm sollte unmittelbar nach den Deklarationen die Anweisung COMMIT WORK gegeben werden, um die Deklarationen in der Datenbank festzuschreiben.

Innerhalb einer Schleife mit Cursorverarbeitung ist COMMIT WORK nur erlaubt, wenn der Cursor mit STORE gespeichert und mit RESTORE wiederhergestellt wurde (siehe Beispiel).

```
                  [display     ]
COMMIT WORK [WITH {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 nicht angegeben, wird die Transaktion beendet und der Programmmlauf im Folge-Teilprogramm ohne Bildschirmausgabe fortgesetzt.

WITH darf nur im Programmmodus angegeben werden.

display — Ausgeben eines Formats. Folgende Anweisungen dürfen verwendet werden:

- DISPLAY screenformat (siehe Handbuch "DRIVE/WINDOWS: Lexikon der DRIVE-Anweisungen" [3], Anweisung DISPLAY screenformat).
- DISPLAY formatname (siehe Handbuch "DRIVE/WINDOWS: Lexikon der DRIVE-Anweisungen" [3], Anweisung DISPLAY formatname).
- DISPLAY FORM (siehe Handbuch "DRIVE/WINDOWS: Lexikon der DRIVE-Anweisungen" [3], Anweisung DISPLAY FORM).

send message
: Ausgeben von Meldungen (siehe Handbuch "DRIVE/WINDOWS: Lexikon der DRIVE-Anweisungen" [3], Anweisung SEND MESSAGE).

stop
: Beenden des DRIVE-Laufs (siehe Handbuch "DRIVE/WINDOWS: Lexikon der DRIVE-Anweisungen" [3], Anweisung STOP).

Regeln

- Wird die Transaktion nicht explizit vom Programmierer mit COMMIT WORK, ROLLBACK WORK oder mit UTM-Sprachmitteln beendet, wird bei Ende der Ausführungseinheit (RUN UNIT) die Transaktion implizit zurückgesetzt. Der Anwender erhält eine Fehlermeldung von DRIVE.
- Falls die aktuelle Transaktion mit einer SET TRANSACTION-Anweisung eröffnet wurde, wird mit COMMIT WORK der darin vereinbarte Transaktionsstatus bei Transaktionsende auf den Standardwert zurückgesetzt.

Beispiel für Cursorverarbeitung in einer Schleife:

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

CREATE TEMPORARY VIEW View deklarieren

CREATE TEMPORARY VIEW deklariert einen View, d.h. eine gespeicherte und benannte Datenbank-Abfrage.

Der Gültigkeitsbereich eines Views wird aufgehoben bei
- Programmende
- 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 view [(satzelement,...)] AS select-ausdruck
```

view
: Name des Views.
Für *view* darf nicht PLAM_DIRECTORY angegeben werden.

  Alle Namen von Views müssen innerhalb der Übersetzungseinheit eindeutig sein. Ein View darf nicht denselben Namen wie eine Basistabelle haben.
Auf einer Programmebene bzw. im Dialog-Modus können zu einem Zeitpunkt keine zwei Views mit identischem Namen deklariert sein.

(satzelement,...)
: vereinbart neue Namen für die Satzelemente innerhalb des Views.

  Die Anzahl der angegebenen Satzelemente muß mit der Anzahl der von *select-ausdruck* gelieferten Satzelemente übereinstimmen. Wird für einen Vektor ein neuer Name vereinbart, dann muß der neue Name in der Form *vektor [(index1[..index2])]* verwendet werden, wenn man sich auf den View bezieht. Geben Sie strukturierte Satzelemente im *select-ausdruck* an, können Sie deren Elemente im View mit den Namen aus der Basistabelle ansprechen.

  satzelement müssen Sie angeben, wenn die von *select-ausdruck* gelieferten Namen der Satzelemente innerhalb des Views nicht eindeutig sind (z.B. durch Join-Attribute) oder kein Name vorhanden ist, wie bei Rechenausdrücken, Mengenfunktionen oder Konstanten.

  Fehlt die Angabe, gelten die Namen der Satzelemente, die *select-ausdruck* liefert.

select-ausdruck

wählt aus bestehenden Tabellen die Ergebnistabelle des Views aus (siehe Metavariable *select-ausdruck*).

Folgende Einschränkungen gelten für den *select-ausdruck*:

– In *select-ausdruck* der CREATE TEMPORARY VIEW-Anweisung darf keine *variable* angegeben werden.

– In der FROM-Klausel dürfen für *tabelle* nur Namen von Basistabellen angegeben werden.

Änderbarer View

Nur wenn der View änderbar ist, kann mit dem View in der zugehörigen Basistabelle geändert werden. Der View ist änderbar, wenn *select-ausdruck* änderbar ist. Der Inhalt eines änderbaren Views ist somit der entsprechende Ausschnitt aus der zugehörigen Basistabelle.

Regel

– Der View muß im Programmtext vor allen nicht-deklarativen Anweisungen deklariert sein und vor allen Deklarationen, die den View verwenden. Innerhalb der aktuellen Transaktion darf vor der Deklaration dieses Views kein View gleichen Namens freigegeben worden sein. Die Viewdeklaration ist nur innerhalb der Übersetzungseinheit gültig.

DECLARE... CURSOR FOR... Cursor deklarieren

DECLARE deklariert einen Cursor und weist dem Cursor ggf. eine *cursor-spezifikation* zu, die die Cursortabelle beschreibt. Die Deklaration muß im Deklarationsteil erfolgen. Sie ist innerhalb einer ganzen Übersetzungseinheit gültig.

Mit dem Cursor können Sie auf die Sätze der Cursortabelle einzeln zugreifen. Der aktuelle Satz, auf den der Cursor zeigt, kann gelesen, gelöscht oder geändert werden. Ändern oder Löschen der Cursortabelle bedeutet immer auch Ändern oder Löschen in der zugrundeliegenden Basistabelle.

Empfehlung:
Im DRIVE-Programm sollte unmittelbar nach den Deklarationen die Anweisung COMMIT WORK gegeben werden, um die Deklarationen in der Datenbank festzuschreiben. Steuerungsanweisungen für Transaktionen auf verschiedenen Programm-Ebenen sind zwar erlaubt, können aber zu schwer durchschaubarem Verhalten im Ablauf des Programms führen.

Gültigkeitsbereich eines Cursors:
Der Gültigkeitsbereich eines Cursors wird aufgehoben bei
- Programmende
- 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 Cursor-Definition erhalten, die Cursor-Position hingegen nicht (Sie können die Cursorposition allerdings mit STORE speichern).

Im Programm-Modus von DRIVE bleibt ein Cursor, der mit PERMANENT definiert wurde, über das Ende eines mit CALL gerufenen Programms hinaus mit seiner Cursor-Position 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] CURSOR [FOR cursor-spezifikation]

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

cursor-spezifikation::= select-ausdruck

                                [ORDER BY {ganzzahl|satzelement} [{ASCENDING|DESCENDING}]},...]
```

cursor — Name des Cursors.
Alle Cursornamen müssen innerhalb einer Übersetzungseinheit eindeutig sein. Auf einer Programmebene bzw. im Dialog-Modus können zu einem Zeitpunkt keine zwei Cursor mit identischem Namen deklariert sein.

PERMANENT — PERMANENT darf nur angegeben werden 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. Programms hinaus erhalten. Die Cursor-Position wird nur aufgehoben, wenn der Cursor explizit mit CLOSE geschlossen wird bzw. beim Übergang in den Dialog-Modus.

i 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 — TEMPORARY darf nur angegeben werden 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).

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 Dekla-

ration mit EXEC müssen Sie die FOR-Klausel angeben. Bis auf die FOR-Klausel müssen die statische und die dynamische Deklaration eines Cursors gleich sein. Alle anderen Cursor-Anweisungen (OPEN, FETCH, DROP, CYCLE) können für den variablen Cursor statisch angegeben werden, wodurch die Performance steigt (siehe DRIVE-Programmiersprache [2] Kapitel Dynamische SQL-Anweisungen).

select-ausdruck

bestimmt die Cursortabelle durch Auswahl von Sätzen und Satzelementen aus bestehenden Basistabellen oder Views. Variablen werden beim Öffnen des Cursors (OPEN) ausgewertet.

ORDER BY Sortieren der Sätze der Cursortabelle auf- oder absteigend nach den Werten der durch *ganzzahl* oder *satzelement* angegebenen Satzelemente.
Bei der Sortierung mit der ORDER BY-Klausel wird ein NULL-Wert kleiner als ein Nicht-NULL-Wert interpretiert. ORDER BY kann nur angegeben werden, wenn *select-ausdruck* änderbar ist.

Sind mehrere Satzelemente angegeben, wird zuerst nach den Werten des ersten angegebenen Satzelements sortiert.

Die angegebenen Satzelemente dürfen weder Primärschlüssel noch Fremdschlüssel bezeichnen. Außerdem dürfen sie nicht strukturiert sein.

Innerhalb einer ORDER BY-Klausel darf entweder nur aufsteigend oder nur absteigend sortiert werden.

satzelement Name eines Satzelements (siehe Metavariable *satzelement*), nach dem sortiert werden soll. Das *satzelement* muß in *select-liste* von *select-ausdruck* vorkommen und darf nicht mit *tabelle* qualifiziert sein.

ganzzahl bezeichnet die Position des Satzelements in der Cursortabelle, nach dem sortiert werden soll (1 ≤ ganzzahl ≤ Anzahl der Satzelemente). Die Satzelemente sind entsprechend den Angaben in *select-liste* von *select-ausdruck* von links nach rechts, beginnend mit 1, durchnummeriert. Auf diese Weise können Sie auch nach Satzelementen sortieren, die, z.B. als Ergebnis von Berechnungen, keinen Namen haben.

ganzzahl muß zwischen 1 und der Anzahl der Satzelemente der Ergebnistabelle liegen. Die Reihenfolge der Satzelemente wird mit *select-ausdruck* festgelegt.

ASCENDING sortiert die Werte des zugehörigen Satzelements aufsteigend.

DESCENDING sortiert die Werte des zugehörigen Satzelements absteigend.

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 *cursor-spezifikation* von der Form *select-ausdruck* ist und *select-ausdruck* änderbar ist (siehe Metavariable *select-ausdruck*).

Regeln

- Der Cursor muß im DRIVE-Programmtext vor allen nicht-deklarativen Anweisungen deklariert sein und vor allen deklarativen Anweisungen, die den Cursor verwenden. Außerdem dürfen auf derselben Programmebene bzw. im Dialog-Modus nicht gleichzeitig Cursor mit identischen Namen vereinbart sein. Der Cursor kann nur innerhalb seines Gültigkeitsbereichs verwendet werden (siehe oben, Gültigkeitsbereich eines Cursors).
- Ein Cursor kann nur innerhalb derselben Transaktion geöffnet, mit FETCH gelesen und geschlossen werden, da COMMIT WORK bzw. ROLLBACK WORK alle offenen Cursor schliessen (Sie können aber auch mit STORE die Cursorposition für einen erneuten Zugriff speichern und den Cursor entsprechend mit RESTORE öffnen).
- Satzelemente, die Ergebnis von Berechnungen sind, erhalten keinen Namen. Ihr Datentyp ergibt sich aus der Rechenoperation und den Datentypen der Operanden (siehe Metavariable *sqlausdruck*).

DELETE... WHERE bedingung Sätze löschen

DELETE... WHERE *bedingung* löscht Sätze aus einer Basistabelle. Diese Anweisung können Sie auch für einen änderbaren View angeben.

```
DELETE FROM tabelle [WHERE bedingung]
```

tabelle — bezeichnet eine Basistabelle oder einen änderbaren View (siehe Metavariable *tabelle*).

bedingung — Für jeden Satz wird geprüft, ob *bedingung* (siehe Metavariable *bedingung*) erfüllt ist. Sätze, die die angegebene *bedingung* erfüllen, werden gelöscht.

Fehlt die Angabe, so werden alle Sätze der angegebenen Basistabelle oder die mit dem View ausgewählten Sätze gelöscht.

Regeln

- Falls es keinen zu löschenden Datensatz gibt, wird &SQL_CODE=100 gesetzt.
- Tritt bei der Abarbeitung der Anweisung DELETE... WHERE *bedingung* beim n-ten Satz (n>1) ein Fehler auf, wird die Transaktion vom Datenbanksystem zurückgesetzt. Tritt der Fehler beim ersten Satz auf, dann wird nur die Durchführung der aktuellen Anweisung zurückgesetzt.

DELETE... WHERE CURRENT OF... Aktuellen Satz des Cursors löschen

DELETE... WHERE CURRENT OF... löscht den aktuellen Satz des Cursors aus der zugrundeliegenden Basistabelle.

Anschließend ist der Cursor vor dem nachfolgenden Satz positioniert. Erst durch eine nachfolgende FETCH-Anweisung wird der Cursor auf den nächsten Satz der Cursortabelle positioniert.

```
DELETE FROM tabelle WHERE CURRENT OF cursor
```

tabelle — bezeichnet eine Basistabelle oder einen änderbaren View (siehe Metavariable *tabelle*).
Der angegebene Name der Tabelle muß übereinstimmen mit dem Tabellennamen aus der FROM-Klausel der zugrundeliegenden Deklaration von *cursor*.

cursor — Name des Cursors.

Regeln

- Der Cursor muß mit DECLARE vorher deklariert worden sein.
- Der Cursor muß
  - mit der OPEN-Anweisung geöffnet und mit FETCH auf einen Satz positioniert sein oder
  - mit einer RESTORE-Anweisung geöffnet sein. In diesem Fall zeigt der Cursor auf denselben Satz, auf den er zum Zeitpunkt der STORE-Anweisung gezeigt hatte.
- Der Cursor muß änderbar sein.

DROP CURSOR Corsor 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 diese Anweisung im Programm-Modus für variable Cursor erlaubt. Beim DROP auf einen variablen Cursor wird die variable Suchbedingung (FOR-Klausel) gelöscht. D.h. Sie können die Suchbedingung des Cursors nach einer DROP-Anweisung erneut nachdeklarieren. Zwischen DROP und Nachdeklaration sollte ein COMMIT WORK stehen.

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

```
      ⎡CURSOR cursor⎤
DROP  ⎨             ⎬
      ⎣CURSORS      ⎦
```

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

CURSORS — Alle im Dialog definierten Cursor bzw. alle auf derselben Programmebene mit EXECUTE definierten Cursor werden freigegeben. Die Anweisung DROP CURSORS ist äquivalent zu einer Reihe von Anweisungen DROP CURSOR *cursor*, in der die Cursornamen explizit angegeben werden.

Beispiel

```
DCL c1 CURSOR;
EXEC 'DCL c1 CURSOR FOR' || &SUCH1;
...
...
DROP CURSOR c1;
COMMIT WORK;
EXEC 'DCL c1 CURSOR FOR' || &SUCH2;
```

DROP TEMPORARY VIEW VIEW freigeben

DROP TEMPORARY VIEW bzw. DROP TEMPRARY VIEWS gibt im Dialog-Modus von DRIVE und innerhalb von EXECUTE (nur, wenn der View auch innerhalb von EXECUTE deklariert wurde) Views frei.
DROP TEMPORAY VIEW(S) unterliegt der Transaktionssicherung, d.h. wenn Sie ROLLBACK WORK angeben, werden die bei DROP TEMPORARY VIEW(S) angegebenen Views nicht freigegeben.

```
DROP TEMPORARY {VIEW view}
               {         }
               {VIEWS    }
```

view — der angegebene View *view* wird freigegeben sowie alle mit ihm assoziierten Cursor, soweit sie im Dialog-Modus oder dynamisch definiert wurden.

VIEWS — Alle im Dialog definierten Views bzw. alle auf derselben Programmebene mit EXECUTE definierten Views und die mit ihnen assoziierten Cursor werden freigegeben. Die Anweisung DROP TEMPORARY VIEWS ist äquivalent zu einer Reihe von Anweisungen DROP TEMPORARY VIEW *view*, in der die Viewnamen explizit angegeben werden.

FETCH Cursor positionieren und Variablen mit Werten aus Satzelementen versorgen

FETCH positioniert den Cursor auf den nächsten Satz in der Cursortabelle und macht diesen zum aktuellen Satz. Im Programm-Modus werden die Werte der Satzelemente dieses aktuellen Satzes in die angegebene(n) Variable(n) übertragen. Im Dialog-Modus werden die Werte der Satzelemente dieses aktuellen Satzes am Datensichtgerät angezeigt.

Der aktuelle Satz kann mit den nachfolgenden Anweisungen UPDATE... WHERE CURRENT OF... bzw. DELETE ... WHERE CURRENT OF ... geändert oder gelöscht werden, falls der Cursor änderbar ist (siehe DECLARE... CURSOR FOR...).

```
FETCH cursor INTO variable,...                (Programm-Modus)
```

```
FETCH cursor                                  (Dialog-Modus)
```

cursor	Name des Cursors.
INTO	die Werte der Satzelemente des aktuellen Satzes werden in die Variablen übertragen.
variable	Name einer Variablen (siehe Metavariable *variable*). Der ersten Variablen wird der Wert des ersten Satzelements zugewiesen, der zweiten Variablen der Wert des zweiten Satzelements usw..

Regeln

- Der Cursor muß mit DECLARE vorher deklariert sein.
- Der Cursor muß mit der OPEN- oder RESTORE-Anweisung geöffnet worden sein.
- Die Anzahl der Variablen muß gleich der Anzahl der Satzelemente in *cursor-spezifikation* der CURSOR-Deklaration sein. Außerdem müssen die Datentypen der Variablen und der entsprechenden Satzelemente verträglich sein.
- Im Fehlerfall erhält &SQL_CODE einen negativen Wert. Die einzelnen Variablen erhalten keinen neuen Wert, d.h. der alte Inhalt bleibt erhalten.

INSERT Satz einfügen

INSERT nimmt einen Satz in eine bestehende Basistabelle auf. Diese Anweisung können Sie auch für einen änderbaren View (siehe CREATE TEMPORARY VIEW) angeben.

```
INSERT INTO tabelle [(satzelement,...)]

        VALUES ( { wert } ,... )
               { NULL }
               { *    }

        [RETURN INTO variable]
```

tabelle
: Name einer Basistabelle oder eines änderbaren View (siehe Metavariable *tabelle*).

(satzelement,...)
: Namen der Satzelemente, in die mit der VALUES-Klausel Werte eingetragen werden sollen.

  Fehlt die Angabe, so gelten alle Satzelemente der angegebenen Basistabelle oder des View. Die Reihenfolge der Satzelemente entspricht bei einer Basistabelle der Reihenfolge gemäß der Definition dieser *tabelle*, bei einem View haben Sie die Reihenfolge mit der Deklaration des View festgelegt.

  satzelement darf nicht qualifiziert sein. Die Satzelemente müssen in der angegebenen Tabelle enthalten sein. Geben Sie nicht alle Satzelemente der angegebenen Tabelle an, so erhalten die fehlenden Satzelemente des aufgenommenen Satzes abhängig von der Definition des Schemas den Defaultwert oder den NULL-Wert. Ein systemdefinierter Primärschlüssel (PRIMARY KEY) erhält einen vom Datenbanksystem vergebenen Wert.

  Ein Satzelement darf nicht mehrfach angegeben werden, auch nicht wieder als Element eines strukturierten Satzelements.

VALUES
: weist den Satzelementen die angegebenen Werte zu. Die Werte werden entsprechend ihrer Reihenfolge den einzelnen Satzelementen zugewiesen.

  Die Anzahl der angegebenen Werte muß übereinstimmen mit der Anzahl der angegebenen Satzelemente. Wenn keine Satzelemente angegeben werden, muß die Anzahl der Werte der Anzahl der Satzelemente der obersten Stufe entsprechen. Das bedeutet, daß für eine Struktur nicht

die Werte für die Satzelemente der Struktur einzeln angegeben werden. Es muß für *wert* ein *aggregat* angegeben werden.

wert — gibt den Wert an, den das Satzelement erhalten soll (siehe Metavariable *wert*).

Wenn das Satzelement strukturiert ist, dann muß der Wert gleich strukturiert sein.

NULL — Mit diesem Schlüsselwort weisen Sie dem entsprechenden Satzelement den NULL-Wert zu.

* — gibt an, daß der Wert für einen systemdefinierten Primärschlüssel vom Datenbanksystem vergeben wird.

* muß und darf nur angegeben werden, wenn das entsprechende Satzelement der systemdefinierte Primärschlüssel ist.

RETURN INTO — vereinbart, daß der Primärschlüsselwert des aufgenommenen Satzes in die Variable übertragen wird. Es dürfen keine signifikanten Stellen verloren gehen.

RETURN INTO ist nur erlaubt, wenn für die angegebene Tabelle ein systemdefinierter Primärschlüssel (PRIMARY KEY) existiert.

variable — Name einer Variablen (siehe Metavariable *variable*).

Die *variable* muß numerisch deklariert sein.

OPEN Cursor öffnen

OPEN öffnet einen Cursor.

OPEN wertet die Cursor-Spezifikation aus, die Sie bei der Vereinbarung des Cursors mit DECLARE angegeben haben. Dabei verwendet OPEN die aktuellen Werte der Variablen, die in *cursor-spezifikation* vorkommen. OPEN positioniert den Cursor vor den ersten Satz der Cursortabelle.

Ein Cursor wird durch eine der folgenden Anweisungen geschlossen:
- CLOSE *cursor*
- COMMIT WORK
- DROP CURSOR bzw. DROP CURSORS
- DROP TEMPORARY VIEW bzw. DROP TEMPORARY VIEWS, falls der Cursor sich auf einen View bezieht.

```
OPEN cursor
```

cursor Name des Cursors.

Regeln

- Der Cursor muß mit DECLARE vorher deklariert sein.
- Der Cursor muß geschlossen sein.
- Eine innerhalb einer vorhergehenden Transaktion mit STORE gespeicherte Cursorposition für diesen Cursor wird ungültig.

PERMIT Benutzeridentifikation angeben

PERMIT gibt die Benutzeridentifikation für ein verwendetes relationales Schema einer UDS-Datenbank an.

Werden die Benutzeridentifikation und Zuordnung einer paßwortgeschützten Datenbank während des Ablaufs eines DRIVE-Programms benötigt, so muß die PERMIT-Anweisung vor dem Aufruf dieses Programms angegeben werden.

Maskengesteuerte Angabe

Im UTM-Vorlauf (nach Angabe des TAC) und wenn nicht PERMIT OFF angegeben ist, wird automatisch eine PERMIT-Bildschirm-Eingabemaske ausgegeben, deren Eingabefelder mit Leerzeichen gefüllt sind.
Wird diese Maske ganz oder teilweise ausgefüllt, wird intern eine PERMIT-Anweisung mit den angegebenen Werten bzw. Standardwerten erzeugt.
Die PERMIT-Maske kann beliebig oft ausgefüllt werden, wenn in der Fußzeile 'R' angegeben ist (Voreinstellung). Wird 'N' angegeben, wird eine PERMIT-Anweisung erzeugt und ausgeführt und anschließend der Bildschirm gelöscht. Wird 'S' angegeben, wird die Maske ohne Erzeugen einer PERMIT-Anweisung beendet.

Standardvereinbarungen

Solange Sie für ein Schema keine PERMIT-Anweisung angegeben haben, gilt die Standardvereinbarung

- Benutzergruppe: SQLGRP
- Benutzername: SQLUSR
- Paßwort: SQLPWD

```
PERMIT SCHEMA={schema  }
              {        }
              {variable}

       [USERGROUP=benutzergruppe]

       [USERNAME=benutzername]

       [PASSWORD=passwort]
```

schema Name des UDS-Schemas.

variable Variable, die den Namen des Schemas bzw. der SESAM-Datenbank enthält.

USERGROUP=benutzergruppe
Wertzuweisung für eine Anwendergruppe (siehe Metavariable *wert*).
benutzergruppe muß als Variable, als alphanumerisches oder als sedezimales Literal angegeben werden (max. 8 Zeichen).
benutzergruppe muß den UDS-Konventionen genügen.
In Programmen muß *benutzergruppe* als Variable angegeben werden.

USERNAME=benutzername
Wertzuweisung für einen Anwendernamen (siehe Metavariable *wert*).
benutzername muß als Variable, als alphanumerisches oder als sedezimales Literal angegeben werden (max. 24 Zeichen).
benutzergruppe muß den UDS-Konventionen genügen.
In Programmen muß *benutzername* als Variable angegeben werden.

PASSWORD=passwort
Wertzuweisung für ein Paßwort (siehe Metavariable *wert*). *passwort* muß als Variable, als alphanumerisches oder als sedezimales Literal angegeben werden.
passwort darf maximal 48 Zeichen lang sein. Das Paßwort muß den SESAM- bzw. UDS-Konventionen für Paßwörter genügen.
In Programmen muß *passwort* als Variable angegeben werden.

Regeln

- Innerhalb einer Transaktion darf pro UDS-Datenbank nur ein relationales Schema angesprochen werden.
- Die PERMIT-Anweisung muß innerhalb einer Transaktion vor allen anderen SQL-Anweisungen ausgeführt werden, die sich auf dasselbe relationale Schema beziehen.
- Innerhalb einer Transaktion ist eine zweite PERMIT-Anweisung für ein relationales Schema nur dann zulässig, wenn noch nicht auf das relationale Schema zugegriffen wurde.
- Wenn Sie mit mehreren relationalen Schemata aus verschiedenen UDS-Datenbanken arbeiten, müssen Sie für jedes Schema eine PERMIT-Anweisung angeben, wenn die Benutzeridentifikation von der Standardvereinbarung abweicht.

- Die Angabe gilt
  - bis zum Ende der Ausführungseinheit (RUN UNIT) bzw. des UTM-Vorgangs,
  - bis zur Ausführung der nächsten PERMIT-Anweisung für dasselbe Schema oder
  - bis die Transaktion mit der ROLLBACK-Anweisung zurückgesetzt wird. Nach dem Zurücksetzen gilt wieder die zuletzt gültige Benutzeridentifikation für dieses Schema oder die Standardvereinbarung.

PERMIT OFF Unterdrücken einer UTM-Eingabemaske

PERMIT OFF ist nur in der UTM-Startprozedur zulässig.

Die Anweisung unterdrückt das standardmäßige Ausgeben einer Bildschirm-Eingabemaske für die Benutzeridentifikation. (PERMIT) im UTM-Vorlauf.

```
PERMIT OFF
```

RESTORE Cursor wiederherstellen

RESTORE stellt einen mit STORE gespeicherten Cursor wieder her.

Dies hat folgende Wirkung:

- Der Cursor wird geöffnet. Im Gegensatz zur OPEN-Anweisung werden nicht die aktuellen Werte von Variablen übernommen. Es gelten die Werte zum Zeitpunkt der letzten OPEN-Anweisung für diesen Cursor.
- Der Cursor zeigt auf denselben Satz, auf den er zum Zeitpunkt der Ausführung der STORE-Anweisung in einer vorausgegangenen Transaktion gezeigt hatte.
- Für den Cursor ist danach keine Cursorposition mehr gespeichert.
  Ein erneuter RESTORE ist daher ohne zwischenzeitlichen STORE nicht mehr möglich.

```
RESTORE cursor
```

cursor Name des Cursors. Die Cursorposition muß mit STORE in einer vorhergehenden Transaktion desselben Vorgangs gespeichert worden sein.

Regeln

- Sie können den Cursor nicht wiederherstellen, wenn der gespeicherte Cursor nach der STORE-Anweisung mit einer der folgenden SQL-Anweisungen ungültig gemacht wurde:
  OPEN Cursor öffnen
  RESTORE Cursor wiederherstellen
  Dies gilt auch dann, wenn die Transaktion zurückgesetzt wird, in der die OPEN- bzw. RESTORE-Anweisung ausgeführt wurde.
- Die mit STORE abgespeicherte Cursorposition kann verlorengehen, wenn in der Zwischenzeit in eigenen oder fremden Transaktionen
  - der aktuelle Satz gelöscht wurde
  - der aktuelle Satz so geändert wurde, daß er die WHERE *bedingung* in *cursorspezifikation* der zugehörigen Cursor-Deklaration nicht mehr erfüllt.
- Eine erneute STORE-Anweisung für denselben Cursor überschreibt einen zuvor gespeicherten Cursor.
- Nach einem RESTORE kann der aktuelle Satz mit UPDATE... WHERE CURRENT OF... geändert oder mit DELETE... WHERE CURRENT OF... gelöscht werden. Mit einer FETCH-Anweisung wird der auf den aktuellen Satz folgende Satz gelesen.

ROLLBACK WORK Transaktion zurücksetzen

ROLLBACK WORK beendet eine Transaktion und macht alle Veränderungen in der Datenbank seit dem Beginn der Transaktion rückgängig.
Im UTM-Betrieb wird nach ROLLBACK WORK der Sicherstellungsbereich auf den letzten Konsistenzpunkt zurückgesetzt.

Ein vor Beginn der Transaktion gespeicherter Cursor wird nicht wiederhergestellt, wenn der Cursor in der mit ROLLBACK WORK zurückgesetzten Transaktion mit OPEN oder RESTORE bearbeitet wurde. In diesem Fall muß der Cursor zur erneuten Bearbeitung wieder mit OPEN eröffnet werden.

Alle innerhalb der Transaktion geöffneten Cursor werden geschlossen.

Ist innerhalb des dynamischen Programmablaufs vor Programmende eine offene Transaktion noch nicht beendet worden, wird von DRIVE ein ROLLBACK WORK durchgeführt und das Programm mit einer Fehlermeldung abgebrochen.

Die erste fehlerfreie SQL-Anweisung nach der ROLLBACK WORK-Anweisung eröffnet 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. Der Inhalt der DRIVE-Variablen wird auf die Werte zurückgesetzt, die zum Zeitpunkt des COMMIT WORK galten. Außerdem werden alle nach diesem COMMIT WORK geöffneten Fenster geschlossen (siehe Handbuch "DRIVE/WINDOWS: Programmiersprache" [2], Kapitel Transaktionssteuerung, Abschnitt Transaktionssicherung für Window-4GL-Anwendungen). 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.

Meldet das Datenbanksystem, daß es implizit (selbständig) die aktuelle Transaktion zurückgesetzt hat, so werden sowohl die Datenbank- als auch die DRIVE-Transaktion zurückgesetzt.

Regel

– Falls in der aktuellen Transaktion eine PERMIT-Anweisung angegeben wurde, gilt nach dem Zurücksetzen wieder die zuletzt gültige Benutzeridentifikation für das Schema oder die Standardvereinbarung.

SELECT Daten abfragen

SELECT wählt einen Satz aus Basistabellen oder Views aus. Dabei können Sie Sätze aus mehreren Basistabellen miteinander verbinden (Join).
Im Programm-Modus überträgt SELECT die Werte der Satzelemente in die mit INTO angegebenen Variablen.
Im Dialog-Modus gibt SELECT die Werte der angegebenen Satzelemente auf den Bildschirm aus.

Die Anzahl der ausgewählten Sätze (Ergebnissätze) der SELECT-Anweisung darf nicht größer als 1 sein. Wollen Sie mehr Sätze lesen, müssen Sie einen Cursor verwenden.

```
SELECT [ALL] select-liste

       [INTO variable,...]

       FROM {tabellenangabe},...

       [WHERE bedingung]
```

ALL
: Standardwert.
Dieser Operand liefert alle mit WHERE *bedingung* ausgewählten Sätze. Auch doppelte Sätze (Duplikate) werden ausgewählt. Beachten Sie, daß beim Vorliegen von Duplikaten somit in jedem Fall ein Fehler auftritt, da als Ergebnismenge der SELECT-Anweisung maximal ein Satz erlaubt ist.

select-liste
: Mit der *select-liste* spezifizieren Sie die Satzelemente der Ergebnistabelle (siehe Metavariable *select-ausdruck*).

INTO
: Mit INTO weisen Sie den Variablen die Werte der Satzelemente aus der (einzeiligen) Ergebnistabelle zu.
INTO gilt nur im Programm-Modus.

  variable
  : Name einer Variablen (entspricht dem Begriff *benutzervariable* im Handbuch "SQL für ISO/SQL (BS2000) Sprachbeschreibung" [11]).
  Der ersten Variablen wird der Wert des ersten Satzelements des Treffersatzes zugewiesen, der zweiten Variablen der Wert der zweiten Satzelements usw..

    Die Anzahl Satzelemente in der *select-liste* muß gleich der Anzahl der Variablen sein. Strukturierten Satzelementen müssen Variablen zugewiesen werden, die gleich strukturiert sind.

Die Datentypen der Variablen und der entsprechenden Satzelemente in *select-liste* müssen verträglich sein (siehe Handbuch "DRIVE/WINDOWS: Programmiersprache" [2], Kapitel Variablen und Konstanten einsetzen, Abschnitt Zuweisungsverträglichkeit bei der Datenkonvertierung).

Wenn kein Treffersatz ermittelt werden konnte, erhält &SQL_CODE den Wert 100, und den Variablen werden keine Werte zugewiesen.

FROM In der FROM-Klausel geben Sie die Basistabellen oder einen View an, aus denen Daten für die Ergebnistabelle ausgewählt werden (siehe Metavariable *select-ausdruck*).

WHERE In der WHERE-Klausel geben Sie die Bedingungen an, um Sätze für die Ergebnistabelle auszuwählen. Die Ergebnistabelle enthält nur Sätze, die die angegebenen Bedingungen erfüllen (siehe Metavariable *select-ausdruck*).

SET TRANSACTION Konsistenzlevel festlegen

SET TRANSACTION legt den Konsistenzlevel für eine Transaktion fest.
Je höher der Konsistenzlevel ist, desto geringer ist der Grad der Transaktionsparallelität.

```
SET TRANSACTION CONSISTENCY LEVEL {2}
                                  {3}
```

Wird die SET TRANSACTION-Anweisung für eine Transaktion nicht angegeben, so gilt der Konsistenzlevel 2 für diese Transaktion.

Generell ist garantiert:

- Eine Transaktion wird entweder komplett durchgeführt oder komplett nicht durchgeführt.
- Keine Änderungen gehen verloren, d.h. "lost updates" sind ausgeschlossen. Unter "lost updates" versteht man folgende Situation: Wenn Transaktion 1 einen Satz ändert, der anschließend von Transaktion 2 geändert wird, kann beim Zurücksetzen von Transaktion 1 auch die zweite Änderung verlorengehen);
- in keiner Transaktion werden die Änderungen anderer, noch nicht beendeter Transaktionen sichtbar, d.h. "dirty read" ist nicht möglich.

Abhängig vom Konsistenzlevel können folgende Phänomene auftreten, wenn zwei Transaktionen parallel auf einen Satz zugreifen:

- Nichtwiederholbares Lesen - "non-repeatable read"
  In einer Transaktion wird ein Satz gelesen, der anschließend von einer zweiten Transaktion geändert wird. Die zweite Transaktion wird beendet und die Änderung festgeschrieben. Liest die erste Transaktion den Satz erneut, dann ist dieser Satz nicht mehr im ursprünglichen Zustand.
- Phantome - "phantoms"
  In einer Transaktion werden Sätze gelesen, anschließend nimmt eine zweite Transaktion weitere Sätze auf. Wiederholt die erste Transaktion die gleiche Suchfrage, kann die Anzahl der Ergebnissätze größer sein.

Die folgende Tabelle zeigt, welche Phänomene bei den einzelnen Konsistenzleveln auftreten können:

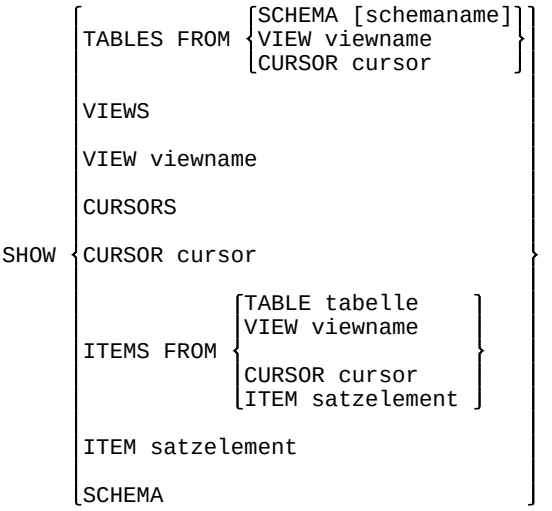
Konsistenzlevel	non-repeatable read	phantoms
2	x	x
3		x

Regel

– Die SET TRANSACTION-Anweisung muß statisch die erste von PERMIT verschiedene Anweisung einer Transaktion sein.

SHOW Ausgeben von Informationen über Metadaten

Mit SHOW kann man sich am Bildschirm Informationen über Basistabellen, Views, Cursor, Satzelemente und Schemanamen ausgeben lassen.



TABLES FROM

Für alle Basistabellen des angegebenen Schemas, Views oder Cursors wird der Name der Tabelle ausgegeben.

Wird *schemaname* nicht angegeben, wird *schemaname* von PARAMETER DYNAMIC genommen.

VIEWS Für alle Views wird der Name des Views und "änderbar/nicht änderbar" ausgegeben.

VIEW viewname

Für *viewname* wird der Name des Views und "änderbar/nicht änderbar" ausgegeben.

CURSORS Für alle Cursor wird der Name des Cursors und "änderbar/ nicht änderbar" ausgegeben.

CURSOR cursor

Für *cursorname* wird der Name des Cursors und "änderbar/ nicht änderbar" ausgegeben.

ITEMS FROM

Für alle Satzelemente der angegebenen Tabelle, des Views oder Cursors wird der Name, der Datentyp mit Stellenanzahl und Nachkommastellen ausgegeben; zusätzlich:

- der Wiederholungsfaktor bei Vektoren oder Wiederholungsgruppen
- die Kennzeichnung eines Schlüssels
- die Nullwertbedingung
- der Name der referenzierten Tabelle (UDS-Basistabelle)

ITEM satzelement

Für *satzelement* wird dieselbe Information wie bei ITEMS FROM ausgegeben, jedoch nur für das angegebene Satzelement.

satzelement muß mit *prefix* und bei Tabellen mit *schemaname* angegeben werden (siehe Metavariable *satzelement*).

SCHEMA Bei UDS-Datenbanken werden alle Schemanamen ausgegeben, für die zu diesem Zeitpunkt Zugriffsrechte vereinbart sind.

STORE Cursorposition speichern

STORE speichert die Cursorposition über eine Transaktion hinaus.

Am Ende einer Transaktion werden alle Cursor geschlossen. Mit STORE kann die Cursorposition vor Transaktionsende gespeichert werden. In einer folgenden Transaktion kann mit RESTORE der Cursor wiederhergestellt werden. Der Cursor hat nach RESTORE dieselbe Position, die er zum Zeitpunkt des STORE hatte.

```
STORE cursor
```

cursor Name des Cursors.

Regeln

- Der Cursor muß mit DECLARE vorher deklariert sein.
- Die gespeicherte Cursorposition bleibt maximal bis zum Ende des UTM-Vorgangs bzw. des Anwenderprogrammlaufs erhalten.
- Der Cursor muß mit der OPEN- oder RESTORE-Anweisung geöffnet und auf einen Satz positioniert sein.
- Eine erneute STORE-Anweisung für denselben Cursor überschreibt die zuvor gespeicherte Cursorposition.

UPDATE... WHERE bedingung
Werte von Satzelementen in ausgewählten Sätzen ändern

UPDATE ... WHERE *bedingung* ändert Werte von Satzelementen in ausgewählten Sätzen einer Basistabelle. Die Anweisung können Sie auch für einen änderbaren View angeben.

```
UPDATE tabelle SET {satzelement=⎧sqlausdruck⎫
                               ⎨          ⎬},...[ WHERE bedingung]
                               ⎩NULL      ⎭
```

tabelle — Name der Basistabelle oder des änderbaren View (siehe Metavariable *tabelle*).

SET — weist den Satzelementen die neuen Werte zu, die mit *sqlausdruck* oder dem Schlüsselwort NULL angegeben werden.

Der Datentyp des Ausdrucks und der Datentyp des Satzelements müssen verträglich sein.

satzelement — gibt das zu ändernde Satzelement an (siehe Metavariable *satzelement*). Das Satzelement muß in der angegebenen Tabelle enthalten sein. *satzelement* darf nicht mit *tabelle* qualifiziert werden.

Ein Satzelement darf nicht mehrfach angegeben werden, auch nicht wieder als Element eines strukturierten Satzelements.
Das Satzelement darf nicht den Primärschlüssel bezeichnen.

Die Werte der nicht angegebenen Satzelemente bleiben unverändert.

sqlausdruck — bezeichnet einen Ausdruck (siehe Metavariable *sqlausdruck*), dessen Wert dem entsprechenden Satzelement zugeordnet wird. Kommt in *sqlausdruck* ein *satzelement* von *tabelle* vor, so gelten die Werte dieses *satzelement* vor einer möglichen Änderung mit UPDATE.

Wenn das Satzelement strukturiert ist, dann muß für *sqlausdruck* entweder eine gleich strukturierte *variable* oder ein *aggregat* angegeben werden (siehe Metavariable *wert*).

NULL — Mit diesem Schlüsselwort weisen Sie dem Satzelement den NULL-Wert zu.

bedingung wählt die Sätze aus, die geändert werden sollen (siehe Metavariable *bedingung*).

Fehlt die Angabe, so werden alle Sätze der angegebenen Basistabelle oder des angegebenen Views geändert.

Regeln

- Falls es keinen Satz gibt, der der Bedingung genügt, wird &SQL_CODE auf 100 gesetzt.
- Tritt bei der Abarbeitung der Anweisung UPDATE ... WHERE *bedingung* beim n-ten Satz (n>1) ein Fehler auf, wird die Transaktion vom Datenbanksystem zurückgesetzt. Tritt der Fehler beim ersten Satz auf, dann wird nur die Durchführung der aktuellen Anweisung zurückgesetzt.
- Ein *sqlausdruck* darf keine *mengenfunktion* enthalten.

UPDATE... WHERE CURRENT OF...
Werte von Satzelementen im aktuellen Satz des Cursors ändern

UPDATE... WHERE CURRENT OF... ändert Werte von Satzelementen des Satzes, auf den der Cursor positioniert ist.

Die Position des Cursors bleibt unverändert.

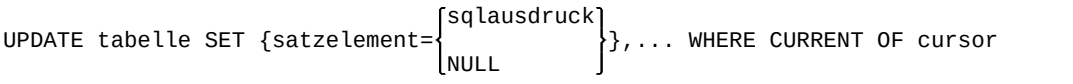


tabelle — bezeichnet die Basistabelle oder einen änderbaren View (siehe Metavariable *tabelle*).
Der angegebene Name der Tabelle muß übereinstimmen mit dem Tabellennamen aus der FROM-Klausel der zugrundeliegenden Cursordeklaration.

SET — weist den Satzelementen die neuen Werte zu, die mit *sqlausdruck* oder NULL angegeben werden.

Der Datentyp des Ausdrucks und der Datentyp des entsprechenden Satzelements müssen verträglich sein.

satzelement — gibt das zu ändernde Satzelement an (siehe Metavariable *satzelement*).
Das Satzelement muß in der angegebenen Tabelle enthalten sein.
satzelement darf nicht mit *tabelle* qualifiziert werden.

Ein Satzelement darf nicht mehrfach angegeben werden, auch nicht wieder als Element eines strukturierten Satzelements.
Das Satzelement darf nicht den Primärschlüssel bezeichnen.

Die Werte der nicht angegebenen Satzelemente bleiben unverändert.

sqlausdruck — bezeichnet einen Ausdruck (siehe Metavariable *sqlausdruck*), dessen Wert dem entsprechenden Satzelement zugeordnet wird. Kommt in *sqlausdruck* ein *satzelement* von *tabelle* vor, so gelten die Werte dieses *satzelement* vor einer möglichen Änderung mit UPDATE.

Wenn das Satzelement strukturiert ist, dann muß für *sqlausdruck* eine gleich strukturierte *variable* oder ein *aggregat* angegeben werden (siehe Metavariable *wert*).

NULL Mit diesem Schlüsselwort weisen Sie dem Satzelement den NULL-Wert zu.

cursor gibt den Namen eines Cursors an. Der Cursor muß änderbar sein.

Regeln

- Der Cursor muß mit DECLARE vorher deklariert sein.
- Der Cursor muß
  - mit der OPEN-Anweisung geöffnet und mit FETCH auf einen Satz positioniert sein oder
  - mit einer RESTORE-Anweisung geöffnet sein. In diesem Fall zeigt der Cursor auf denselben Satz, auf den er zum Zeitpunkt der STORE-Anweisung gezeigt hatte.
- *sqlausdruck* darf keine *mengenfunktion* enthalten.

bedingung Bedingungen vereinbaren

Eine Bedingung besteht aus einem oder mehreren logischen Ausdrücken und den logischen Operatoren AND, OR oder NOT.
Erfüllt ein Satz die vereinbarte Bedingung, wird der Satz in die Ergebnistabelle übernommen (CREATE, DECLARE) bzw. die entsprechende Aktion ausgeführt (DELETE, UPDATE).

Bei Bedingungen wird unterschieden zwischen Join- und Auswahlbedingungen. Mit einer Joinbedingung verbinden Sie Basistabellen miteinander. Eine Auswahlbedingung wählt Sätze aus einer Tabelle aus.

Folgende Auswahlbedingungen können gestellt werden:

- Vergleichen eines Satzelements mit einem Ausdruck
- Vergleichen eines Satzelements mit einem Wertebereich
- Vergleichen eines Satzelements mit einer Liste von Werten
- Vergleichen eines Satzelements mit dem NULL-Wert
- Vergleichen eines Satzelements mit einem teilweise unbekannten Wert

NULL-Werte in Bedingungen

Wenn NULL-Werte in Bedingungen vorkommen, kann das Ergebnis von Bedingungen neben erfüllt und nicht erfüllt auch unbestimmt sein. Eine ausführliche Beschreibung, wann das Ergebnis einer Bedingung erfüllt, nicht erfüllt oder unbestimmt ist, finden Sie im Handbuch "SQL für UDS/SQL V1.0" [13] unter dem Syntaxelement *bedingung*.
Falls das Ergebnis von WHERE *bedingung* unbestimmt ist, dann reagiert das Datenbanksystem so, als ob die Bedingung nicht erfüllt wäre.

Es gilt folgende Syntax für *bedingung*:

```
                ⎧joinbedingung [AND auswahlbedingung]...              ⎫
                ⎪                                                     ⎪
bedingung ::=   ⎨                      ⎧AND⎫                          ⎬
                ⎪auswahlbedingung1[    ⎨   ⎬ auswahlbedingung2]...    ⎪
                ⎩                      ⎩OR ⎭                          ⎭
```

```
joinbedingung ::= satzelement = satzelement [AND satzelement = satzelement]...
```

```
                              ⎧             ⎧= ⎫                                  ⎫
                              ⎪             ⎪< ⎪                                  ⎪
                              ⎪             ⎪> ⎪                                  ⎪
                              ⎪sqlausdruck. ⎨<>⎬ sqlausdruck                      ⎪
                              ⎪             ⎪<=⎪                                  ⎪
                              ⎪             ⎩>=⎭                                  ⎪
                              ⎪                                                   ⎪
                              ⎪satzelement [NOT] BETWEEN sqlausdruck1             ⎪
                              ⎪                                AND sqlausdruck2   ⎪
auswahlbedingung ::= [NOT]    ⎨satzelement [NOT] IN (wert,...)                    ⎬
                              ⎪                                                   ⎪
                              ⎪satzelement IS [NOT] NULL                          ⎪
                              ⎪                                                   ⎪
                              ⎪satzelement [NOT] LIKE muster                      ⎪
                              ⎪            [ESCAPE escape-zeichen]                ⎪
                              ⎪                                                   ⎪
                              ⎪                        ⎧AND⎫                      ⎪
                              ⎪(auswahlbedingung1[     ⎨   ⎬ auswahlbedingung2]...)⎪
                              ⎩                        ⎩OR ⎭                      ⎭
```

Die folgenden Ergebnisse von *bedingung* sind für AND, OR und NOT möglich:

AND logisches UND: beide mit AND verknüpften Bedingungen müssen erfüllt sein, damit die gesamte Bedingung erfüllt ist.

OR logisches ODER: mindestens eine der beiden mit OR verknüpften Bedingungen muß erfüllt sein, damit die gesamte Bedingung erfüllt ist.

NOT Negation: die mit NOT verknüpfte Bedingung darf nicht erfüllt sein, damit die gesamte Bedingung erfüllt ist.
Ist das Ergebnis der mit NOT verknüpften Bedingung unbestimmt, so ist auch das Ergebnis der gesamten Bedingung unbestimmt.

joinbedingung

Mit einer Joinbedingung verbinden Sie Basistabellen miteinander (Join). Die Satzelemente, die Sie in einer Joinbedingung angeben, werden auch als JOIN-Felder bezeichnet.

Eine Joinbedingung formulieren Sie mit einer Vergleichsbedingung, indem Sie ein Satzelement einer Basistabelle mit dem Satzelement einer zweiten Basistabelle durch den Vergleichsoperator "=" vergleichen.
Die Datentypen der Satzelemente, die miteinander verglichen werden, müssen verträglich sein.

Wenn ein Satzelement einen Fremdschlüssel bezeichnet, muß das zweite Satzelement der Primärschlüssel derjenigen Basistabelle sein, auf die der Fremdschlüssel verweist. Sie verknüpfen dann die Basistabellen über einen Fremdschlüssel und den zugehörigen Primärschlüssel miteinander.

Ein Join für mehrere Basistabellen entsteht, wenn die Joinbedingung aus mehreren Vergleichsbedingungen besteht, die durch den logischen Operator AND miteinander verknüpft sind. Dieser Join wird multipler Join genannt.

Durch die erste Vergleichsbedingung einer Joinbedingung wird die Verknüpfung zweier Basistabellen festgelegt.
Durch jede folgende Vergleichsbedingung wird eine neue Basistabelle spezifiziert. Diese Basistabelle ist mit genau einer Basistabelle verknüpft, die durch eine der vorhergehenden Vergleichsbedingungen festgelegt wurde. In einer Vergleichsbedingung muß deshalb genau ein Satzelement aus einer Basistabelle auftreten, die bereits in einer vorhergehenden Vergleichsbedingung angesprochen wurde.

Ein Join über drei Basistabellen wird mit zwei Vergleichsbedingungen und AND formuliert. Ein Join über vier Basistabellen wird mit drei Vergleichsbedingungen formuliert usw.

Bei einer besonderen Form des Joins wird eine Basistabelle mit sich selbst verbunden. Dann stammen beide Satzelemente aus der gleichen Basistabelle. Da jedoch Basistabellen in der FROM-Klausel unterschiedliche Namen haben müssen, muß für mindestens eine dieser Basistabellen ein Synonym vereinbart werden.

auswahlbedingung

Mit der Auswahlbedingung werden Sätze aus einer Tabelle ausgewählt.

auswahlbedingung1 OR auswahlbedingung2

Alle Satzelemente müssen in einer Tabelle enthalten sein. Falls sich die FROM-Klausel auf einen Join-View bezieht, dürfen nur Satzelemente spezifiziert werden, die bei der Viewdefinition derselben Basistabelle zugeordnet waren.

auswahlbedingung1 AND auswahlbedingung2 [AND...]

Innerhalb von *auswahlbedingung1* , *auswahlbedingung2* usw. müssen alle Satzelemente aus derselben Tabelle stammen. Falls sich die FROM-Klausel auf einen Join-View bezieht, dann dürfen innerhalb von *auswahlbedingung1*, *auswahlbedingung2* usw. nur Satzelemente spezifiziert werden, die bei der Viewdefinition derselben Basistabelle zugeordnet waren.

Auswahlbedingungen können Sie mit den logischen Operatoren AND, OR und NOT verknüpfen. Durch das Setzen von Klammern können Sie Einheiten zusammenfassen.

Regel

Wenn Sie die logischen Operatoren AND, OR und NOT kombinieren, so gelten die üblichen Vorrangregeln für die Auswertung:
NOT vor AND vor OR.
Wenn Sie die beschriebene Reihenfolge ändern wollen, so müssen Sie entsprechend Klammern setzen. Operatoren innerhalb der Klammern haben Vorrang.

Die folgenden Seiten beschreiben die verschiedenen Formate für *auswahlbedingung* in den einzelnen Funktionen und erläutern die Angaben.

Nicht beschrieben sind in *bedingung* die Angaben für:

- *sqlausdruck* siehe Metavariable *sqlausdruck*
- *satzelement* siehe Metavariable *satzelement*

Vergleichen von Ausdrücken mit Vergleichsoperatoren

Mit Vergleichsoperatoren können Sie die Werte zweier Ausdrücke miteinander vergleichen.

```
                  =
                  <
                  >
sqlausdruck1      {   } sqlausdruck2
                  <=
                  >=
                  <>
```

Vergleichsoperator	Bedeutung
=	gleich
<	kleiner
>	größer
<=	kleiner oder gleich
>=	größer oder gleich
<>	ungleich

Die Bedingung ist erfüllt, wenn der Vergleich zutrifft.
Das Ergebnis der Bedingung ist unbestimmt, wenn mindestens ein Ausdruck den NULL-Wert annimmt.

Regeln

- Mindestens ein Ausdruck muß ein Satzelement sein. Der zweite Ausdruck kann entweder ein Satzelement oder ein Wert sein.
- Innerhalb einer Auswahlbedingung dürfen zur Bildung eines Vergleichswertes nur Satzelemente verwendet werden, die alle aus derselben Tabelle stammen.
- Wenn ein Ausdruck einen Fremdschlüssel oder Primärschlüssel bezeichnet, muß der Vergleichsoperator "=" oder "<>" sein.
- Die Ausdrücke müssen entweder beide numerisch, alphanumerisch oder gleich strukturiert sein.
- Wenn ein Ausdruck ein strukturiertes Satzelement bezeichnet, dann muß der Vergleichsoperator "=" oder "<>" sein, und der zweite Ausdruck muß gleich strukturiert sein.
- Alphanumerische Ausdrücke werden zeichenweise miteinander verglichen. Sind die Ausdrücke ungleich lang, wird der kürzere Ausdruck mit Leerzeichen aufgefüllt.

Vergleichen eines Satzelements mit einem Wertebereich

Es wird geprüft, ob der Wert des Satzelements innerhalb oder außerhalb des Wertebereichs liegt.

```
satzelement [NOT] BETWEEN sqlausdruck1 AND sqlausdruck2
```

BETWEEN ... AND

Das Ergebnis der Bedingung ist dasselbe wie für die Bedingung *sqlausdruck1* <= *satzelement* AND *satzelement* <= *sqlausdruck2*.

Die Bedingung ist erfüllt, wenn der Wert von *satzelement* innerhalb des Wertebereichs liegt.

NOT BETWEEN ... AND

Das Ergebnis der Bedingung ist dasselbe wie für die Bedingung *satzelement* < *sqlausdruck1* OR *satzelement* > *sqlausdruck2*.

Die Bedingung ist erfüllt, wenn der Wert von *satzelement* außerhalb des Wertebereichs liegt.

Regeln

- Das Satzelement und die Ausdrücke dürfen nicht strukturiert und müssen entweder alle alphanumerisch oder alle numerisch sein.
- Jeder Ausdruck bezeichnet einen *wert*.

Vergleichen eines Satzelements mit einer Liste von Werten

Das Satzelement wird mit einer Liste von angegebenen Werten verglichen.

```
satzelement [NOT] IN (wert,...)
```

IN
- Die Bedingung ist erfüllt, wenn der Wert von *satzelement* nicht der NULL-Wert ist und mit mindestens einem der nach IN angegebenen Werte übereinstimmt.
- Die Bedingung ist nicht erfüllt, wenn weder der Wert von *satzelement* noch einer der nach IN angegebenen Werte der NULL-Wert ist und der Wert von *satzelement* mit keinem der nach IN angegebenen Werte übereinstimmt.

Andernfalls ist das Ergebnis der Bedingung unbestimmt.

NOT IN
- Die Bedingung ist erfüllt, wenn weder der Wert von *satzelement* noch einer der nach IN angegebenen Werte der NULL-Wert ist und der Wert von *satzelement* mit keinem der nach IN angegebenen Werte übereinstimmt.
- Die Bedingung ist nicht erfüllt, wenn der Wert von *satzelement* nicht der NULL-Wert ist und mit mindestens einem der nach IN angegebenen Werte übereinstimmt.

<menge> Andernfalls ist das Ergebnis der Bedingung unbestimmt.

Regeln

- Das Satzelement und die Werte müssen entweder alle numerisch, alphanumerisch oder gleich strukturiert sein.
- Sie müssen nach IN eine Liste von Werten angeben, d.h. mehr als einen Wert.

Vergleichen eines Satzelements mit dem NULL-Wert

```
satzelement IS [NOT] NULL
```

IS NULL — Die Bedingung ist erfüllt, wenn der Wert von *satzelement* der NULL-Wert ist. Andernfalls ist die Bedingung nicht erfüllt.

IS NOT NULL — Die Bedingung ist erfüllt, wenn der Wert von *satzelement* nicht der NULL-Wert ist. Andernfalls ist die Bedingung nicht erfüllt.

Regel

– Das Satzelement muß einen Fremdschlüssel bezeichnen.

Vergleich eines Satzelements mit einem teilweise unbekannten Wert

Der Wert eines Satzelements wird mit einem vorgegebenen Muster verglichen.

```
satzelement [NOT] LIKE muster
            [ESCAPE escape-zeichen]
```

LIKE — Die Bedingung ist erfüllt, wenn der Wert von *satzelement* mit *muster* übereinstimmt.

NOT LIKE — Die Bedingung ist erfüllt, wenn der Wert von *satzelement* mit *muster* nicht übereinstimmt.

Das Ergebnis der Bedingung ist unbestimmt, wenn der Wert von *satzelement* oder von *muster* der NULL-Wert ist.

muster — Für *muster* geben Sie eine alphanumerische Konstante *konst* oder eine *variable* an.
Das Muster darf aus folgenden Zeichen bestehen:

muster enthält	Die Bedingung ist erfüllt, wenn das satzelement an derselben Stelle ...
zeichen	genau das Zeichen enthält
_ (Tiefstrich)	ein beliebiges Zeichen enthält
%	eine beliebige Zeichenfolge oder nichts enthält
escape-zeichen%	% enthält
escape-zeichen_	_ enthält
escape-zeichenescape-zeichen	escape-zeichen enthält

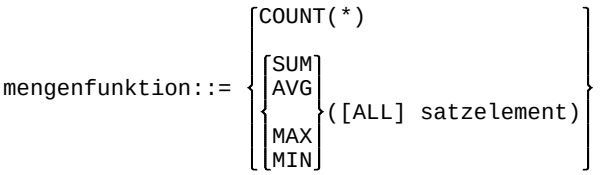
escape-zeichen
vereinbart ein Zeichen, mit dem Sie die Zeichen "%", "_" oder das *escape-zeichen* als Musterzeichen entwerten können. Sie können damit prüfen, ob das Satzelement diese Zeichen enthält.
Für *escape-zeichen* geben Sie eine alphanumerische Konstante *konst* oder eine alphanumerische *variable* der Länge 1 an.

Regeln

- Das Satzelement muß alphanumerisch sein.
- "%" ohne vorangehendes *escape-zeichen* darf nur am Ende von *muster* angegeben werden.
- *escape-zeichen* muß direkt vor "%", "_" oder einem zweiten *escape-zeichen* stehen.
- *escape-zeichen* und das darauffolgende Zeichen werden in *muster* als ein Zeichen gewertet.
- Die Länge eines Musters, das kein Sonderzeichen "%" und kein *escape-zeichen* enthält, muß übereinstimmen mit der definierten Länge des jeweiligen Satzelements.

mengenfunktion Mengenfunktionen angeben

Eine Mengenfunktion berechnet einen Wert aus einer Menge von Sätzen.



COUNT(*) gibt die Anzahl der Sätze aus. Duplikate werden berücksichtigt.

SUM([ALL] satzelement)
addiert die Werte von *satzelement*. *satzelement* muß numerisch sein.

AVG([ALL] satzelement)
errechnet aus den Werten von *satzelement* den Mittelwert. *satzelement* muß numerisch sein.

MAX([ALL] satzelement)
liefert den größten Wert von *satzelement*. *satzelement* muß numerisch oder alphanumerisch sein.

MIN([ALL] satzelement)
liefert den kleinsten Wert von *satzelement*. *satzelement* muß numerisch oder alphanumerisch sein.

Das Ergebnis der Mengenfunktion hat folgenden Datentyp:

Mengenfunktion	Datentyp des Ergebnisses
COUNT(*)	Datentyp DECIMAL mit der Genauigkeit 15, keine Nachkommastellen.
MIN und MAX	gleicher Datentyp wie angegebenes *satzelement*.
SUM	Datentyp DECIMAL mit der Genauigkeit 15. Die Anzahl der Nachkommastellen entspricht der Anzahl der Nachkommastellen des angegebenen *satzelement*.
AVG	Datentyp DECIMAL mit der Genauigkeit 15. Die Anzahl der Nachkommastellen entspricht 15 minus der Anzahl der Stellen vor dem Komma des angegebenen *satzelement*

Regeln

- Mengenfunktionen sind nur innerhalb der *select-liste* einer SELECT-Anweisung oder eines *select-ausdruck* zulässig.
- Das Satzelement darf nicht strukturiert sein.
- NULL-Werte im Satzelement werden bei der Berechnung von SUM, AVG, MAX und MIN nicht berücksichtigt.
- Wenn das Ergebnis von WHERE *bedingung* des *select-ausdruck* bzw. der SELECT-Anweisung keinen Ergebnissatz liefert, so ist das Ergebnis von SUM, AVG, MIN und MAX der NULL-Wert. Bei COUNT(*) ist das Ergebnis "0".

satzelement Satzelemente angeben

Für *satzelement* können Sie folgendes angeben:

- einfaches Satzelement
- Vektor
- Struktur
- Vektor mit strukturierten Elementen

```
satzelement ::=

                                                    ⌈einf-satzelement             ⌉
                                                    |                             |
[⌈tabelle⌉   ⌈struktur1                    ⌉      |struktur2                      |
 |       |.][|                             |].]...|                               |
 ⌊synonym⌋   ⌊vektor-strukturiert1(index1)⌋      |vektor[(index1[..index2])]     |
                                                    |                             |
                                                    ⌊vektor-strukturiert[(index1)]⌋
```

---

tabelle Name einer Basistabelle oder eines Views (siehe Metavariable *tabelle*). Zur eindeutigen Qualifikation müssen Sie *satzelement* mit *tabelle* qualifizieren, wenn in einer Anweisung identische Namen für Satzelemente aus verschiedenen Tabellen vorkommen.

synonym Synonym für eine Tabelle.
synonym kann in der FROM-Klausel der SELECT-Anweisung oder des *select-ausdruck* vereinbart werden. *synonym* müssen Sie zur eindeutigen Identifikation angeben, wenn in einer Anweisung identische Namen für Satzelemente aus verschiedenen Tabellen vorkommen und Sie Synonyme dafür vereinbart haben.

struktur1 Name einer Struktur.

vektor-strukturiert1(index1)
Strukturiertes Element eines Vektors *index1* muß größer sein als 0.

einf-satzelement
Name eines einfachen Satzelements.

struktur2 Name einer Struktur.

vektor[(index1[..index2])]
: Name eines Vektors. Sie können nur ein Element eines Vektors (indizierter Vektor) mit *index1* oder einen Teilvektor mit *index1*..*index2* angeben. Die angegebenen Elemente müssen im Vektor enthalten sein.
Für *index1* bzw. *index2* gilt:
index1 und *index2* müssen ganzzahlige numerische Konstanten sein.
index1 muß größer sein als 0 und
index2 muß größer sein als *index1*.

vektor-strukturiert2[(index1)]
: Name eines Vektors mit strukturierten Elementen.
Sie können aber auch nur ein Element des Vektors mit strukturierten Elementen mit *index1* angeben. Das angegebene Element muß im Vektor mit strukturierten Elementen enthalten sein.
Für *index1* gilt:
index1 muß größer sein als 0.

Regeln

- *tabelle* und *synonym* dürfen nicht angegeben werden für *satzelement* in einer ORDER BY-Klausel.
- Geben Sie *satzelement*, *struktur*, *vektor* oder *vektor-strukturiert* als untergeordnetes Satzelement an, muß das übergeordnete Satzelement *struktur* oder *vektor-strukturiert* sein. Die einzelnen Satzelemente müssen durch "." (Punkt) voneinander getrennt werden.
- Ist das Satzelement ein Element eines Vektors mit strukturierten Elementen, so müssen Sie mit *vektor-strukturiert1(index1)* angeben, zu welcher Ausprägung des Vektors mit strukturierten Elementen das angegebene Satzelement gehört.
- Wenn der angegebene Name innerhalb der Anweisung nicht eindeutig ist, muß der Name der Tabelle oder das Synonym für die Tabelle dem Satzelement vorangestellt werden.
- Wenn Sie in einer SELECT-Abfrage einem Satzelement den Namen einer Tabelle voranstellen, müssen Sie diesen Tabellennamen auch in der zugehörigen FROM-Klausel angeben.

select-ausdruck
SELECT innerhalb von SQL-Anweisungen angeben

Der *select-ausdruck* wählt Sätze bzw. Satzelemente aus Basistabellen oder Views aus. Dabei können Sie Sätze aus mehreren Basistabellen miteinander verbinden (Join).

Das Ergebnis ist wieder eine Tabelle, die Ergebnistabelle. Die Ergebnistabelle enthält die über die SELECT-Liste ausgewählten Satzelemente der Sätze, die Sie über die Selektion mit der WHERE-Klausel auswählen.

Wird in weiteren SQL-Anweisungen der View oder der Cursor verwendet, ist die Ergebnistabelle die Grundlage für die Abfrage von Daten und Änderungen in der Datenbank.

Änderbarer SELECT-Ausdruck

Der *select-ausdruck* und damit die Ergebnistabelle kann änderbar oder nicht änderbar sein. *select-ausdruck* ist änderbar, wenn

- in der *select-liste* nur Satzelemente angegeben sind und jedes Satzelement nur einmal angesprochen ist. Ein Satzelement darf nicht einmal als Element eines strukturierten Satzelements angegeben werden und dann noch einmal direkt mit dem Namen des Satzelements.
  Zum Beispiel ist die Angabe des strukturierten Satzelements ADRESSE und dann nochmal die Angabe von ORT als Element von Adresse nicht erlaubt. Teilvektoren dürfen sich nicht überlappen.
- in der FROM-Klausel nur eine Tabelle angegeben ist. Die Tabelle muß eine Basistabelle oder ein änderbarer View sein.

Ein View bzw. ein Cursor ist änderbar, wenn *cursor-spezifikation* von der Form *select-ausdruck* ist und *select-ausdruck* änderbar ist..

```
select-ausdruck::=  SELECT [ALL] select-liste

                           FROM tabellenangabe,...

                           [WHERE bedingung]
```

Die Reihenfolge der Klauseln muß eingehalten werden.

ALL — Standardwert liefert alle mit WHERE *bedingung* ausgewählten Sätze. Auch doppelte Sätze (Duplikate) werden ausgewählt. Die Angabe ALL ist syntaktisch erlaubt, hat aber keine Auswirkung auf das Ergebnis.

select-liste **Satzelemente auswählen**

Mit der *select-liste* spezifizieren Sie die Satzelemente der Ergebnistabelle.

```
select-liste ::= ⎧*                ⎫
                 ⎩sqlausdruck,...  ⎭
```

*	Die Ergebnistabelle enthält alle Satzelemente der in der FROM-Klausel angegebenen Tabelle(n). Die Reihenfolge der Satzelemente wird bestimmt – durch die Reihenfolge der Tabellen in der FROM-Klausel und – innerhalb einer Tabelle durch die im Schema definierte Reihenfolge.
sqlausdruck	Die Ergebnistabelle enthält die mit *sqlausdruck*,... spezifizierten Satzelemente (siehe Metavariable *sqlausdruck*) in der angegebenen Reihenfolge.

Regeln

– Wenn ein *sqlausdruck* eine *mengenfunktion* enthält, muß jedes Satzelement in der *select-liste* innerhalb einer Mengenfunktion stehen.

– Alle Namen von Satzelementen müssen eindeutig sein. Wenn Sie Basistabellen verbinden und diese Basistabellen Satzelemente mit identischen Namen haben, müssen Sie zur eindeutigen Identifizierung die jeweiligen Namen der Basistabellen bzw. deren Synonyme voranstellen.

– Die Eigenschaften eines Satzelements der Ergebnistabelle (Datentyp, Länge, Genauigkeit, Skalenfaktor) werden entweder von der zugrundeliegenden Tabelle übernommen oder ergeben sich aus dem angegebenen Ausdruck.

FROM Tabellen auswählen

In der FROM-Klausel geben Sie die Basistabellen oder einen View an, aus denen Daten für die Ergebnistabelle ausgewählt werden.

```
FROM tabellenangabe,...

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

tabellenangabe::= tabelle [synonym]
```

tabelle — gibt eine Basistabelle oder einen View an, deren Satzelemente im SELECT-Ausdruck verwendet werden (siehe Metavariable *tabelle*). Geben Sie mehr als eine Tabelle an, so müssen es Basistabellen sein. Das Verbinden von Tabellen (Join) ist nur mit Basistabellen möglich. Mit der WHERE-Klausel können dann die Sätze aus dieser Tabelle ausgewählt werden.

Werden zwei oder mehr Tabellen in der FROM-Klausel angegeben (in diesem Fall muß die WHERE-Klausel angegeben werden), so entsteht eine Ergebnistabelle, die alle Satzelemente von *select-liste* enthält und deren Ergebnissätze aus allen möglichen Kombinationen der Sätze der einzelnen Tabellen bestehen (sog. kartesisches Produkt). Da in der WHERE-Klausel in jedem Fall eine Joinbedingung angegeben werden muß, wenn in der FROM-Klausel mehr als eine Tabelle aufgeführt ist, wird die durch die FROM-Klausel festgelegte Ergebnistabelle in der Regel nicht alle Sätze der Ergebnistabelle enthalten.

synonym — *synonym* vereinbart einen neuen Namen für *tabelle*. Innerhalb der SELECT-Abfrage kann dann der neue Name verwendet werden. Der Name für *synonym* muß den Konventionen für Namen entsprechen.

Regeln

- Eine FROM-Klausel darf sich nur auf Basistabellen oder einen View beziehen.
- Wenn in der FROM-Klausel identische Tabellennamen auftreten, müssen Synonyme für die Unterscheidung dieser Tabellennamen angegeben werden.
- Alle Synonyme innerhalb der FROM-Klausel müssen unterschiedlich sein und dürfen mit keinem Tabellennamen übereinstimmen, für den kein *synonym* vereinbart wurde.

WHERE Sätze auswählen

In der WHERE-Klausel geben Sie die Bedingungen an, um Sätze für die Ergebnistabelle auszuwählen. Die Ergebnistabelle enthält nur Sätze, die die angegebene Bedingung erfüllen.

```
WHERE bedingung
```

WHERE bedingung

wählt aus der durch die FROM-Klausel festgelegten Ergebnistabelle Sätze aus, die die vorgegebene Bedingung erfüllen (siehe Metavariable *bedingung*).

Die WHERE-Klausel muß angegeben werden, wenn Sie in der FROM-Klausel mehr als eine Tabelle angegeben haben. In diesem Fall muß eine Joinbedingung angegeben werden.

Wenn die Angabe fehlt und Sie in der FROM-Klausel nur eine Tabelle angeben, enthält die Ergebnistabelle alle Sätze der angegebenen Tabelle.

Jedes in *bedingung* unmittelbar enthaltene *satzelement* muß in einer *tabelle* der FROM-Klausel vorkommen.

sqlausdruck Ausdrücke angeben

Ausdrücke bestehen aus Satzelementen, Konstanten, Variablen oder Mengenfunktionen. Ausdrücke können auch zu einem Ausdruck zusammengefaßt werden, indem sie durch arithmetische Operatoren (+,-,*,/) verknüpft werden (Rechenausdruck).

```
                    ⌈satzelement                ⌉
                    |wert                       |
                    |mengenfunktion              |
                    |                           |
                    |⌈+⌉                        |
                    |{ }sqlausdruck             |
                    |⌊-⌋                        |
sqlausdruck ::=     {                           }
                    |                           |
                    |             ⌈+⌉           |
                    |             |-|           |
                    |sqlausdruck{   }sqlausdruck|
                    |             |*|           |
                    |             ⌊/⌋           |
                    |                           |
                    |(sqlausdruck)              |
                    ⌊                           ⌋
```

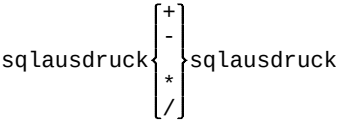
satzelement
: bezeichnet den Namen eines Satzelements (siehe Metavariable *satzelement*). Das Satzelement muß in einer Tabelle enthalten sein, die in der FROM-Klausel der SELECT-Abfrage bzw. innerhalb der Anweisungen DELETE, INSERT oder UPDATE angegeben ist.

wert
: bezeichnet einen Wert (siehe Metavariable *wert*).

mengenfunktion
: bezeichnet eine 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.

-sqlausdruck, +sqlausdruck
: "-" bewirkt einen Wechsel des Vorzeichens. "+" läßt den Wert von *sqlausdruck* unverändert. *sqlausdruck* muß numerisch sein. *sqlausdruck* darf nicht mit "+" oder "-" anfangen. Erlaubt sind damit insbesondere folgende Varianten für numerische Werte bzw. Ausdrücke:

```
⌈+⌉ ⌈satzelement   ⌉
{ } |wert         |
⌊-⌋ |mengenfunktion|
    ⌊(sqlausdruck)⌋
```



bezeichnet die Rechenarten Addition, Subtraktion, Multiplikation und Division. Beide Operanden müssen numerisch sein.

(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.

Der Datentyp des ermittelten Ergebnisses eines Rechenausdrucks ist numerisch.
Die Genauigkeit (Anzahl der Vor- und Nachkommastellen) und der Skalenfaktor (Anzahl der Nachkommastellen) des ermittelten Rechenergebnisses sind abhängig

- von der Genauigkeit bzw. vom Skalenfaktor der Rechenoperanden und
- vom verwendeten Rechenoperator.

Wie Sie die jeweilige Genauigkeit und den jeweiligen Skalenfaktor ermitteln können, entnehmen Sie der folgenden Tabelle:

Rechenart	Ergebnis
Addition x+y Subtraktion x-y	Die Genauigkeit P ist gleich der Summe aus dem Maximum der Anzahl von Vorkommastellen und dem um 1 erhöhten Maximum der Anzahl von Nachkommastellen; höchstens aber 15. $P = MIN\ (15,\ MAX(Vor_x,Vor_y) + MAX(S_x,S_y) + 1)$ [1)] Der Skalenfaktor S ist gleich der Anzahl der Nachkommastellen des Operanden, der die maximale Anzahl der Nachkommastellen hat. $S = MAX(S_x,S_y)$ [1)]
Multiplikation x*y	Die Genauigkeit P ist gleich der Summe der Genauigkeiten der Faktoren; höchstens aber 15. $P = MIN\ (15,\ P_x+P_y))$ [1)] Der Skalenfaktor ist gleich der Summe der Skalenfaktoren der Faktoren; höchstens aber 15. $S = MIN(15,\ S_x+S_y)$ [1)]
Division x/y	Die Genauigkeit P ist gleich 15. $P = 15$ Der Skalenfaktor S errechnet sich nach folgender Formel: $S = MAX(15-P_x+S_x-S_y,\ 0)$ [1)]

[1)] Vor_x - Anzahl der Vorkommastellen des ersten Rechenoperanden
Vor_y - Anzahl der Vorkommastellen des zweiten Rechenoperanden

S_x - Skalenfaktor (Anzahl der Nachkommastellen) des ersten Rechenoperanden
S_y - Skalenfaktor (Anzahl der Nachkommastellen) des zweiten Rechenoperanden

P_x - Genauigkeit des ersten Rechenoperanden
P_y - Genauigkeit des zweiten Rechenoperanden

Es gilt: $P_x = Vor_x + S_x$ $P_y = Vor_y + S_y$

Wenn das mathematische Ergebnis des Rechenausdrucks nach evtl. Abschneiden von Nachkommastellen nicht in das in der obigen Tabelle beschriebene Ergebnisfeld passt, tritt ein Wertüberlauf auf (SQLCODE -340). Bei Division durch Null erhält man ebenfalls den SQLCODE -340. Ist der Skalenfaktor des Rechenergebnisses größer als 15, dann gehen Nachkommastellen verloren.

Regeln

- Wenn in einem Rechenausdruck ein NULL-Wert vorkommt, dann hat der gesamte Ausdruck den Wert NULL.
- Für das Setzen von arithmetischen Operatoren gelten die üblichen algebraischen Regeln.
- Wenn ein *sqlausdruck* eine *mengenfunktion* enthält, muß jedes *satzelement* in der *select-liste* innerhalb einer *mengenfunktion* stehen.

sql_literal Literale angeben

Literale sind Zeichenfolgen, die einen konstanten Wert repräsentieren.

Man unterscheidet zwischen alphanumerischen, numerischen und sedezimalen Literalen. Numerische Literale können nur numerischen Satzelementen und alphanumerische Literale nur alphanumerischen Satzelementen zugeordnet werden.
DRIVE wandelt sedezimale Literale intern in alphanumerische Literale um.

In den Handbüchern "SQL für SESAM/SQL V1.0" [12] und "SQL für UDS/SQL V1.0" [13] wird für Literale der Begriff Konstanten verwendet (Syntaxelement *konst*).

```
                ⎧alphanumerisches_literal⎫
sql_literal::= ⎨numerisches_literal      ⎬
                ⎩sedec_literal           ⎭
```

Alphanumerische Literale

```
alphanumerisches_literal ::= 'zeichen...' [(n)]
```

zeichen — Jedes beliebige ASCII-Zeichen ist erlaubt.
alphanumerisches_literal darf maximal 256 Zeichen enthalten.
Wird das Zeichen ' (Hochkomma) in *alphanumerisches_literal* verwendet, muß es doppelt angegeben werden: ' '. Das verdoppelte Hochkomma wird als ein Zeichen gewertet.

n — Wiederholungsfaktor. Die vor *n* stehende Zeichenfolge wird *n*-mal wiederholt.

Numerische Literale

```
                              ⎡+⎤ ⎡ganzzahl      ⎤
numerisches_literal ::=  [⎨  ⎬][⎨                ⎬
                              ⎣-⎦ ⎣festpunktzahl⎦
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
                    ⎡ganzzahl[.ganzzahl]⎤
festpunktzahl ::= ⎨ganzzahl.            ⎬
                    ⎣.ganzzahl          ⎦
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
ganzzahl ::= ziffer...
```

ziffer — Ziffer von 0 bis 9.
Eine Festpunktzahl darf maximal 15 Ziffern enthalten, davon maximal 14 Nachkommastellen.

Sedezimale Literale

```
sedec_literal := X’sedec_ziffer ...’  [(n)]
```

sedec_ziffer — Sedezimale Ziffer, d.h. 0, 1, ...9, A, B, ... F.
Jede sedezimale Ziffer repräsentiert die Bit-Belegung eines Halbbytes. Belegt werden muß jeweils ein ganzes Byte, d.h. ’sedec_ziffer ... ’ muß eine gerade Anzahl von sedezimalen Ziffern enthalten.
Maximal dürfen 512 sedezimale Ziffern (= 256 Byte) angegeben werden.

tabelle Tabellen angeben

Tabellen, die in SQL-Anweisungen angegeben werden, können Basistabellen oder Views sein. Tabellen können mit dem Namen *schema* qualifiziert sein.

```
                    ⎧basistabelle⎫
tabelle::= [schema.] ⎨            ⎬
                    ⎩view        ⎭
```

schema — bezeichnet das relationale Schema, in dem sich die Basistabelle befindet.
Fehlt die Angabe, wird das in der OPTION SCHEMA bzw. in PARAMETER DYNAMIC SCHEMA angegebene Schema verwendet.
Ist weder eine Angabe OPTION SCHEMA noch PARAMETER DYNAMIC SCHEMA vorhanden, so muß der Tabellen- bzw. Viewname eindeutig sein.
Führt eine fehlende *schema*-Angabe dazu, daß eine *basistabelle* bzw. ein *view* nicht eindeutig identifizierbar ist, gibt DRIVE eine Fehlermeldung aus. Wenn innerhalb einer Transaktion auf Basistabellen zugegriffen wird, die sich in verschiedenen Schemata befinden, müssen die Schemata in verschiedenen Datenbanken definiert sein.

view — Name eines Views. Der View muß im Programmtext vor allen anderen Anweisungen deklariert sein, die sich auf den View beziehen.

variable Variablen angeben

Geben Sie innerhalb von SQL-Anweisungen Datenelemente als Variablen an, muß folgende Schreibweise beachtet werden.

```
variable::=  &datenname
```

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). Ausgabe-Variablen sind somit alle Variablen, die an folgenden Stellen vorkommen:

- in der RETURN INTO-Klausel der INSERT-Anweisung
- in der INTO-Klausel der SELECT-Anweisung
- in der FETCH-Anweisung nach INTO.

Alle anderen Benutzervariablen sind Eingabe-Benutzervariablen.

Datentypen von Satzelementen und zugehörigen Benutzervariablen müssen miteinander verträglich sein. Miteinander verträglich sind numerische Satzelemente und numerische Benutzervariablen bzw. alphanumerische Satzelemente und alphanumerische Benutzervariablen. Strukturierte Benutzervariablen und Satzelemente sind miteinander verträglich, wenn beide die gleiche Anzahl von Komponenten haben und die entsprechenden Komponenten miteinander verträglich sind.

datenname Name der Variablen.
In DRIVE müssen Variablen in der Form "&datenname" angegeben werden. *datenname* darf maximal 31 Zeichen lang sein.
Die genaue Syntax finden Sie bei der Metavariablen *variable* im Handbuch "DRIVE/WINDOWS: Lexikon der DRIVE-Anweisungen" [3]. DRIVE bietet Ihnen außerdem die Möglichkeit, sich durch DECLARE VARIABLE ... LIKE Variablen anzulegen, deren Struktur und Namen der Tabelle bzw. dem Cursor entspricht.

wert Werte angeben

Ein Wert kann vereinbart werden durch eine Variable, ein Literal oder ein Aggregat. Ein Aggregat kann nur dann für *wert* angegeben werden, wenn einem strukturierten Satzelement ein Wert zugewiesen oder ein strukturiertes Satzelement mit einem Wert verglichen werden soll.

```
          ⎡sql_literal⎤
wert ::= {⎢variable   ⎥}
          ⎣aggregat   ⎦
```

```
                ⎡wert⎤
aggregat ::= <{ ⎢    ⎥},...>
                ⎣NULL⎦
```

sql_literal bezeichnet ein numerisches oder alphanumerisches Literal.

variable Name einer Variablen.

aggregat bezeichnet einen zusammengesetzten Wert für die Darstellung von Werten für ein strukturiertes Satzelement. Für jedes untergeordnete Satzelement des strukturierten Satzelements muß ein Wert angegeben werden.

wert
: bezeichnet wieder ein Literal, eine Variable oder ein *aggregat*.

NULL
: Das Schlüsselwort NULL wird verwendet, wenn der NULL-Wert angegeben werden werden soll.

Literatur

[1] **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.

DRIVE V6.0A (BS2000)
Das Programmiersystem
Benutzerhandbuch

Zielgruppe
Anwendungsprogrammierer
Inhalt
- Einführung in das Programmiersystem DRIVE
- Erläuterung der Funktionen des Dialog-Modus'
- Installationsbeschreibung
- Generierung und Administration von DRIVE im UTM-Betrieb

[2] **DRIVE/WINDOWS** (SINIX)
Programmiersprache
Sprachbeschreibung

Zielgruppe
Anwendungsprogrammierer
Inhalt
Beschreibung der Programmerstellung einschließlich Grafik- und Alpha-Bildschirmformaten sowie Listenformaten mit DRIVE und Report Generator.

DRIVE V6.0A (BS2000)
Beschreibung der Programmiersprache DRIVE

Zielgruppe
Anwendungsprogrammierer
Inhalt
- Erläuterung von SQL-Begriffen und -Konzepten
- Beschreibung der Programmerstellung
- Beispiele

[3] **DRIVE/WINDOWS** (SINIX)
Lexikon der DRIVE-Anweisungen
Referenzhandbuch

Zielgruppe
Anwendungsprogrammierer
Inhalt
Syntax und Funktionsumfang aller DRIVE-Anweisungen. Meldungen und Schlüsselwörter von DRIVE.

DRIVE V6.0A (BS2000)
Lexikon der Anweisungen
Benutzerhandbuch

Zielgruppe
Anwendungsprogrammierer
Inhalt
- Syntax und Funktionsumfang aller DRIVE-Anweisungen
- Meldungen und Schlüsselwörter von DRIVE

[4] **DRIVE/WINDOWS** (SINIX)
Lexikon der DRIVE-SQL-Anweisungen ffr INFORMIX
Referenzhandbuch

Zielgruppe
Anwendungsprogrammierer
Inhalt
Syntax und Funktionsumfang aller DRIVE-SQL-Anweisungen für INFORMIX in Kurzform.

[5] **DRIVE/WINDOWS** (SINIX)
Lexikon der DRIVE-SQL-Anweisungen ffr ISO/SQL
Referenzhandbuch

Zielgruppe
Anwendungsprogrammierer

Inhalt
Syntax und Funktionsumfang aller DRIVE-SQL-Anweisungen für ISO/SQL in Kurzform.

[6] **DRIVE/WINDOWS** (SINIX)
Lexikon der DRIVE-SQL-Anweisungen für SESAM
Referenzhandbuch

Zielgruppe
Anwendungsprogrammierer
Inhalt
Syntax und Funktionsumfang aller DRIVE-SQL-Anweisungen für SESAM in Kurzform.

[7] **DRIVE/WINDOWS** (SINIX)
Lexikon der DRIVE-SQL-Anweisungen ffr UDS
Referenzhandbuch

Zielgruppe
Anwendungsprogrammierer
Inhalt
Syntax und Funktionsumfang aller DRIVE-SQL-Anweisungen für UDS in Kurzform.

[8] **DRIVE/WINDOWS V1.1**
(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.

[9] **DRIVE V6.0A** (BS2000)
Beschreibung der Programmiersprache DRIVE

Zielgruppe
Anwendungsprogrammierer
Inhalt
- Erläuterung von SQL-Begriffen und -Konzepten
- Beschreibung der Programmerstellung
- Beispiele

[10] **DRIVE V6.0A** (BS2000)
Lexikon der Anweisungen
Benutzerhandbuch

Zielgruppe
Anwendungsprogrammierer
Inhalt
- Syntax und Funktionsumfang aller DRIVE-Anweisungen
- Meldungen und Schlüsselwörter von DRIVE

[11] System Interfaces for Applications
SQL für ISO/SQL(BS2000)
Portierbare SQL-Anwendungen für BS2000 und SINIX
Sprachbeschreibung

Zielgruppe
Anwender, die mit SQL oder DRIVE auf SESAM- bzw. UDS-Datenbanken zugreifen wollen.
Inhalt
Das Handbuch beschreibt den Sprachumfang des Produktes ISO/SQL V1.0. Außerdem ermöglicht das Handbuch das Erstellen portierbarer SQL-Anwendungen in BS2000 und SINIX, da der gemeinsame Sprachumfang von ISO/SQL, INFORMIX und SQL-Norm hervorgehoben ist.

[12] **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.

[13] **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.

[14] **INFORMIX** (SINIX)
SQL
Sprachbeschreibung

Zielgruppe
INFORMIX-Anwender
Inhalt
Vollständige Beschreibung der Datenbanksprache INFORMIX-SQL für alle INFORMIX-Produkte, die eine SQL-Benutzerschnittstelle zur Verfügung stellen. Abweichungen und Erweiterungen gegenüber dem ANSI-Standard sind ebenfalls beschrieben.

[15] **INFORMIX**
Ergänzungsband

Zielgruppe
INFORMIX-Anwender
Inhalt
Der Ergänzungsband für INFORMIX V4.1 enthält die funktionalen Änderungen für die INFORMIX-Produkte: SQL-Sprachbeschreibung, SQL-Nachschlagen, ESQL/COBOL, ESQL/C, C-ISAM und Interaktiver Debugger. Die Handbücher der Version 4.0 werden benötigt.

[16] **Fehlermeldungen für INFORMIX-Produkte** (SINIX)
Benutzerhandbuch

Zielgruppe
INFORMIX-Anwender
Inhalt
Das Handbuch enthält die Texte der Fehlermeldungen, die bei der Arbeit mit INFORMIX-Produkten auftreten können und die dazugehörigen Maßnahmetexte.

[17] **SESAM/SQL** (BS2000)
Aufbau und Wartung
Benutzerhandbuch

Zielgruppe
Datenbank-Verwalter
Inhalt
- Aufbau und Wartung von SESAM-Datenbanken mit dem Datenbank-Administrationsmonitor SESASB
- Schattendatenbankbetrieb

[18] **Dialog Builder V2.0**

[19] **OSF/Motif**
Programmer's Reference
Release 1.2

Zielgruppe
- Anwendungsentwickler
- Widget-Entwickler

Inhalt
Beschreibung aller Kommandos, Funktionen und Dateiformate des OSF/Motif-Widget-Set.

[20] **SINIX/windows User Environment**
Leitfaden für Experten und Systemverwalter
Benutzerhandbuch

Zielgruppe
Experten in SINIX/windows und Systemverwalter

Inhalt
Das Handbuch erläutert die dem Produkt zugrundeliegenden Konzepte und die Konfiguration der Bedienoberfläche. Die zentralen Clients für Display-Verwaltung, Fensterverwaltung und Verwaltung der Service-Werkzeuge und Dateien werden vorgestellt.

[21] **SINIX/windows User Environment**
Clients zum Nachschlagen
Referenzhandbuch

Zielgruppe
Experten in SINIX/windows und Systemverwalter

Inhalt
Das Handbuch gibt einen vollständigen Überblick über die Clients: über Aufruf, Optionen und Ressourcen, die ihr Aussehen und Verhalten bestimmen. Es erläutert die Reihenfolge und Priorität, in der Ressource-Festlegungen ausgewertet werden.

[22] **X Window System**
Xlib Reference Manual

Zielgruppe
- Anwendungsentwickler
- Widget-Entwickler

Inhalt
Vollständige Beschreibung der C-Programmierschnittstelle der Xlib.

[23] **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

[24] **FHS** (TRANSDATA)
Benutzerhandbuch

Zielgruppe
Programmierer
Inhalt
Programmschnittstellen von FHS für TIAM-, DCAM- und UTM-Anwendungen. Erstellen, Einsatz und Verwalten von Formaten.

[25] **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.

[26] **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.

[27] **UTM** (SINIX)
Anwendung generieren und administrieren
Benutzerhandbuch

Zielgruppe
Systemverwalter und Administratoren

Inhalt
- Aufbau, Generierung und Betrieb von UTM-Anwendungen auf SINIX
- Arbeiten mit UTM-Meldungen und Fehlercodes.

Einsatz
SINIX-Transaktionsbetrieb

[28] **UTM** (SINIX)
Planen und Entwerfen
Benutzerhandbuch

Zielgruppe
- Organisatoren
- Einsatzplaner
- Programmierer

Inhalt
- Einführung in UTM (SINIX), Erläuterung des Programm-Speicher- und Schnittstellenkonzeptes sowie der Behandlung von Daten, Dateien und Datenbanken
- Hinweise zu Design, Optimierung und Performance von UTM-Anwendungen auf SINIX sowie Datenschutz.

Einsatz
SINIX-Transaktionsbetrieb

[29] **UPIC**
Client-Server-Kommunikation mit UTM
Benutzerhandbuch

Zielgruppe
Organisatoren, Einsatzplaner Programmierer von UPIC-Programmen
Inhalt
UPIC ermöglicht Programm-Programm-Kommunikation zwischen einer UPIC-Anwendung und einer UTM-Anwendung. Dies funktioniert sowohl lokal mit einer UTM(SINIX)-Anwendung im gleichen Rechner als auch remote mit UTM-Anwendungen auf anderen SINIX- oder BS2000-Rechnern.
Mit UPIC können Sie moderne Präsentationssysteme wie Motif, X Window System an UTM(SINIX) und UTM(BS2000) anbinden. Das Handbuch beschreibt, wie Sie eine UTM-Anwendung in C programmieren.

[30] **ERMS** (SINIX)
Kommandoschnittstelle

Zielgruppe
- Anwender
- Datenbankverwalter

Inhalt
– Konzepte des ERMS aus Anwendersicht
– Beschreibung und Erläuterung der Kommandos für den Anwender

[31] **RADAR V1.0**

[33] **TOM-REF** (BS2000)
Data-Dictionary-System
Benutzerhandbuch

Zielgruppe
Software-Entwickler
Inhalt
Das Handbuch beschreibt Bedeutung und Nutzen des Data-Dictionary-Systems TOM-REF in den Projektphasen der Software-Entwicklung. Es erklärt die Funktionen und die Bedienung des TOM-REF im BS2000 sowie den Daten-Export in ein ERMS-DD im SINIX.

[34] **C-DS C** (SINIX)
Referenzhandbuch für Programmierer

Zielgruppe
C-Programmierer, die unter SINIX V5.41 mit C-DS V1.0 arbeiten.
Inhalt
Beschreibung der Kommandos für die Programmentwicklung, der Bibliotheksfunktionen und Systemaufrufe und einiger Header-Dateien und c-spezifischer Dateiformate.

[35] **C-DS C** (SINIX)
Leitfaden und Werkzeuge für die Programmierung mit C
Benutzerhandbuch

Zielgruppe
C-Programmierer, die unter SINIX V5.41 mit C-DS V1.0 arbeiten.
Inhalt
Im Handbuch werden das C-Übersetzungssystem (Präprozessor, Binder, Include-Dateien, Bibliotheken) und Dienstprogramme zur Entwicklung, Verwaltung, Wartung und Generierung von C-Programmen beschrieben.

[36] SINIX V5.41
Installationsanleitung MX300

Zielgruppe
Service-Techniker und Systemverwalter *Inhalt* Ausführliche Anleitung zur Installation des Betriebssystems SINIX V5.41. Beschreibung der Inbetriebnahme eines vorinstallierten MX300, der Vorbereitungsarbeiten zur Installation von SINIX V5.41 sowie der Installation und Deinstallation von Software.

[37] MX500 (SINIX V5.40)
MX300 (SINIX V5.41)
Referenzhandbuch für Systemverwalter

Zielgruppe
Systemverwalter
Inhalt
Beschreibt Kommandos und Anwendungsprogramme zur Systempflege sowie Dateiformate, spezielle Dateien zur Systemverwaltung und gibt Diagnosehinweise.

[38] **Kommandos** (SINIX V5.40)
Teil 1, A - K
Beschreibung

Zielgruppe
SINIX-Shell-Anwender
Inhalt
Beschreibung der SINIX-Kommandos in alphabetischer Reihenfolge

[39] **Kommandos** (SINIX V5.40)
Teil 2, L - Z
Beschreibung

Zielgruppe
SINIX-Shell-Anwender
Inhalt
Beschreibung der SINIX-Kommandos in alphabetischer Reihenfolge

[40] SINIX V5.41
Leitfaden für Benutzer Benutzerhandbuch

Zielgruppe
Benutzer
Inhalt
Beschreibung der wesentlichen Elemente des SINIX-Betriebssystems. Dazu gehört: Einführung in die Benutzung von SINIX, das Dateisystem, die Prozeßverarbeitung, die Shell.

[41] case/4/0 Methodenhandbuch
* microTOOL GmbH Berlin

[42] case/4/0 Referenzhandbuch
* microTOOL GmbH Berlin

[43] case/4/0 Bedienerhandbuch
* microTOOL GmbH Berlin

[44] **Widget Set**
Programmer's Reference

Zielgruppe
– Anwendungsentwickler
– Widget-Entwickler
Inhalt
Beschreibung aller Kommandos, Funktionen und Dateiformate zu den Widgets XmSniBrowser, XmSniFormat, XmSniHelp, XmSniTable und XmSniTree in der Siemens-Nixdorf-Erweiterung vom OSF/Motif-Widget-Set.

[45] **SINIX-SPOOL**
Anwenden, Verwalten, Programmieren
Benutzerhandbuch

Zielgruppe
Benutzer, Verwalter und Programmierer des SINIX-SPOOL's
Inhalt
Beschreibung der Kommandos, Verwaltungsfunktionen und der C-Schnittstelle zum SINIX-SPOOL

[46] **Styleguide**
Richtlinien zur Gestaltung von Benutzeroberflächen
Benutzerhandbuch

Zielgruppe
Entwickler von Anwendungsprogrammen
Inhalt
Der Styleguide gibt Regeln und Empfehlungen für die Entwicklung einheitlicher Benutzeroberflächen. Es werden jeweils Aufbau, Inhalt und Bedienablauf dargestellt.

[47] SINIX V5.40 (MX500)
SINIX V5.41 (MX300)
Leitfaden für Systemverwalter
Beschreibung

Zielgruppe
Systemverwalter
Inhalt
– Einführung in die Systemverwaltung von SINIX-Systemen
– Anleitung zur Konfigurierung und Wartung des SINIX-Systems

[48] **Kommandos** (SINIX V5.40)
Teil 3, Tabellen und Verzeichnisse
Beschreibung

Zielgruppe
SINIX-Shell-Anwender
Inhalt
Tabellen und Verzeichnisse zu den in Teil 1 und 2 beschriebenen Kommandos – Inhaltsverzeichnis
- Kommando-Übersicht
- Reguläre Ausdrücke
- Sonderzeichen der BOURNE-Shell
- Gerätedateien für Datenträger
- Dateien des SPOOL-Systems in SINIX V5.23 und V5.40
- Zeichensatz ISO 646
- Fachwort, Literatur- und Stichwortverzeichnis

[49] **File Transfer mit SINIX**
FT-SINIX (SINIX) V5.0A
FTOS-SINIX (SINIX) V2.0A
Benutzerhandbuch

Zielgruppe
Das Handbuch wendet sich an SINIX-Benutzer, die FT/FTOS-SINIX nutzen wollen, sowie an den SINIX-Systemverwalter.
Inhalt
Das Handbuch beschreibt die Funktionen von FT-SINIX/FTOS-SINIX. FT-SINIX dient zur Übertragung von Dateien und zum Dateimanagement auf Basis der FTNEA-Protokolle. Das Zusatzprodukt FTSOS-SINIX ermöglicht die Funktionen auf Basis des FTAM-Protokolls.

Mit * markierte Titel sind nicht von der Siemens Nixdorf Informationssysteme AG oder der Siemens AG herausgegeben.

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

Inhalt

DRIVE/WINDOWS V1.1 (BS2000/SINIX)

Lexikon der DRIVE-SQL-Anweisungen für UDS

Zielgruppe

Anwendungsprogrammierer

Inhalt

Syntax und Funktionsumfang aller DRIVE-SQL-Anweisungen für UDS in Kurzform.

Ausgabe: Dezember 1993

Datei: DRV_UDS.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.