Deutsch

FUJITSU Software BS2000

SESAM/SQL-Server V9.1 Sprachbeschreibung Teil 1

Benutzerhandbuch

Juni 2019

Kritik… Anregungen… Korrekturen…

Die Redaktion ist interessiert an Ihren Kommentaren zu diesem Handbuch. Ihre Rückmeldungen helfen uns, die Dokumentation zu optimieren und auf Ihre Wünsche und Bedürfnisse abzustimmen.

Sie können uns Ihre Kommentare per E-Mail an bs2000services@ts.fujitsu.com senden.

Zertifizierte Dokumentation nach DIN EN ISO 9001:2015

Um eine gleichbleibend hohe Qualität und Anwenderfreundlichkeit zu gewährleisten, wurde diese Dokumentation nach den Vorgaben eines Qualitätsmanagementsystems erstellt, welches die Forderungen der erfüllt.

Copyright und Handelsmarken

Copyright © 2018 Fujitsu Technology Solutions GmbH.

Alle Rechte vorbehalten.
Liefermöglichkeiten und technische Änderungen vorbehalten.

Alle verwendeten Hard- und Softwarenamen sind Handelsnamen und/oder Warenzeichen der jeweiligen Hersteller.

Inhaltsverzeichnis

Sprachbeschreibung Teil 1 . . . . . **14**
1 Einleitung **15**
1.1 Zielsetzung und Zielgruppen des Handbuchs **16**
1.2 Konzept des Handbuchs **17**
1.3 Darstellungsmittel **18**
2 Programmeinbettung von SQL **20**
2.1 Programmstruktur **21**
2.2 Benutzervariable **22**
2.2.1 Benutzervariable definieren 23
2.2.2 Benutzervariable verwenden 24
2.2.3 Indikatorvariable 25
2.2.3.1 Indikatorvariable definieren 26
2.2.3.2 Indikatorvariable verwenden 27
2.3 Erfolgskontrolle und Fehlerbehandlung **28**
2.3.1 Erfolgskontrolle 29
2.3.2 Fehlerbehandlung 30
2.4 Cursor **31**
2.4.1 Cursor zum Lesen 32
2.4.2 Änderbarer Cursor 33
2.4.3 Cursor definieren 34
2.4.4 Cursor öffnen 35
2.4.5 Cursor positionieren und Satz lesen 36
2.4.6 Satz ändern oder löschen 37
2.4.7 Cursor speichern 38
2.4.8 Cursor schließen 39
2.4.9 Cursor wiederherstellen 40
2.4.10 Cursor-Beispiele 41
2.5 Dynamische SQL **43**
2.5.1 Dynamisch formulierte Anweisung 44
2.5.1.1 Dynamisch formulierte Anweisung vorbereiten 45
2.5.1.2 Datentypen der Platzhalter und Ergebniswerte abfragen 46
2.5.1.3 Dynamisch formulierte Anweisung ausführen 47
2.5.2 Dynamisch formulierte Cursorbeschreibung 48
2.5.2.1 Dynamisch formulierte Cursorbeschreibung vorbereiten 49
2.5.2.2 SQL-Datentypen der Platzhalter bestimmen 50
2.5.2.3 SQL-Datentypen der Ergebnisspalten bestimmen 51
2.5.2.4 Dynamisch formulierte Cursorbeschreibung auswerten 52

2.5.2.5 Ergebnisse ablegen 53
2.5.3 Deskriptorbereich 54
2.5.3.1 Deskriptorbereich anlegen 55
2.5.3.2 Aufbau eines Deskriptorbereichs 56
2.5.3.3 Deskriptorbereichsfelder 57
2.5.3.4 Deskriptorbereich belegen 62
2.5.3.5 Deskriptorbereich abfragen 63
2.5.3.6 Werte aus dem Deskriptorbereich verwenden 64
2.5.3.7 Deskriptorbereich freigeben 65
2.6 SQL-Anweisungen in CALL-DML-Transaktionen 66
2.6.1 CALL-DML-Anwendungen schrittweise umstellen 67
2.6.2 User-Close berücksichtigen und Betriebsmittel freigeben 68
2.6.3 Isolationslevel einstellen 69
3 Lexikalische Elemente und Namen 70
3.1 SESAM/SQL-Zeichenvorrat 71
3.2 Lexikalische Einheiten 72
3.2.1 Zeichenfolgen 73
3.2.2 Ziffernfolgen 74
3.2.3 Delimiter-Symbole 75
3.2.4 Trenner 76
3.2.5 Kommentar 77
3.3 Pragmas und Annotationen 78
3.3.1 Pragma AUTONOMOUS TRANSACTION 81
3.3.2 Pragma DATA TYPE 82
3.3.3 Pragma DEBUG ROUTINE 83
3.3.4 Pragma DEBUG VALUE 84
3.3.5 Pragma EXPLAIN 86
3.3.6 Pragma ISOLATION LEVEL 87
3.3.7 Pragma LIMIT ABORT_EXECUTION 88
3.3.8 Pragma LOCK MODE 89
3.3.9 Pragma LOOP LIMIT 90
3.3.10 Pragma PREFETCH 91
3.3.11 Pragma UTILITY MODE 93
3.4 Namen 94
3.4.1 Einfache Namen 95
3.4.2 Qualifizierte Namen 100
3.4.3 Namen definieren 102
4 Datentypen und Werte 103
4.1 Überblick über Datentypen und zugehörige Wertebereiche 104
4.1.1 Einteilung der Datentypen 105
4.1.2 Wertebereich 106

4.1.3 Spalte 107
4.1.4 Parameter von Routinen und lokale Variable 108
4.2 Datentypen 109
4.2.1 Übersicht über SQL-Datentypen 110
4.2.2 Alphanumerische Datentypen und National-Datentypen 111
4.2.2.1 CHARACTER - Zeichenketten fester Länge 112
4.2.2.2 CHARACTER VARYING - Zeichenketten variabler Länge 113
4.2.2.3 NATIONAL CHARACTER - Zeichenketten fester Länge 114
4.2.2.4 NATIONAL CHARACTER VARYING - Zeichenketten variabler Länge . . 115
4.2.3 Numerische Datentypen 116
4.2.3.1 SMALLINT - Kleine Ganzzahlen 117
4.2.3.2 INTEGER - Ganzzahlen 118
4.2.3.3 NUMERIC - Festpunktzahlen 119
4.2.3.4 DECIMAL - Festpunktzahlen 120
4.2.3.5 REAL- Gleitpunktzahlen mit einfacher Genauigkeit 121
4.2.3.6 DOUBLE PRECISION - Doppelte Genauigkeit 122
4.2.3.7 FLOAT - Gleitpunktzahl 123
4.2.4 Zeit-Datentypen 124
4.2.4.1 DATE - Datum 125
4.2.4.2 TIME - Uhrzeit 126
4.2.4.3 TIMESTAMP - Zeitstempel 127
4.2.5 Verträglichkeit von Datentypen 128
4.3 Werte 129
4.3.1 Literale 130
4.3.2 Wert angeben 131
4.3.3 Werte für multiple Spalten 132
4.3.4 NULL-Wert 133
4.3.4.1 Schlüsselwort für den NULL-Wert 134
4.3.4.2 NULL-Wert in Tabellenspalten 135
4.3.4.3 NULL-Wert in Funktionen, Ausdrücken und Prädikaten 136
4.3.4.4 NULL-Wert in GROUP BY 137
4.3.4.5 NULL-Wert in ORDER BY 138
4.3.5 Zeichenketten 139
4.3.5.1 Alphanumerische Literale 140
4.3.5.2 National-Literale 142
4.3.5.3 Spezial-Literale 145
4.3.5.4 Zeichenketten verwenden 147
4.3.6 Numerische Werte 150
4.3.6.1 Numerische Literale 151
4.3.6.2 Numerische Werte verwenden 152
4.3.7 Zeitwerte 153

4.3.7.1 Zeitliterale . . . 154
4.3.7.2 Zeitwerte verwenden . . . 156
4.4 Zuweisungsregeln . . . 158
4.4.1 Werte in Tabellenspalten eintragen . . . 159
4.4.2 Defaultwerte für Tabellenspalten . . . 160
4.4.3 Werte für Platzhalter . . . 161
4.4.4 Werte in Benutzervariable oder Deskriptorbereich lesen . . . 162
4.4.5 Werte zwischen Benutzervariablen und Deskriptorbereich übertragen . . . 164
4.4.6 Zieldatentyp anpassen mit dem CAST-Operator . . . 166
4.4.7 Eingabeparameter für Routinen versorgen . . . 167
4.4.8 Werte in Prozedurparameter (Ausgabe) oder lokale Variable eintragen . . . 168
5 Zusammengesetzte Sprachelemente . . . 169
5.1 Ausdruck . . . 170
5.2 Funktion . . . 175
5.2.1 Zeitfunktionen . . . 177
5.2.2 Zeichenkettenfunktionen . . . 178
5.2.3 Numerische Funktionen . . . 180
5.2.4 Mengenfunktionen . . . 181
5.2.5 Tabellenfunktionen . . . 184
5.2.6 Kryptografische Funktionen . . . 185
5.2.7 User Defined Functions (UDF) . . . 187
5.2.8 Alphabetischer Nachschlageteil Funktionen . . . 188
5.2.8.1 ABS() - Absolutwert . . . 189
5.2.8.2 AVG() - Arithmetisches Mittel . . . 190
5.2.8.3 CEILING() - kleinste ganze Zahl größer als der Wert . . . 192
5.2.8.4 CHAR_LENGTH() - Zeichenkettenlänge bestimmen . . . 193
5.2.8.5 COLLATE() - Collation-Element für National-Zeichenketten ermitteln . . . 195
5.2.8.6 COUNT(*) - Tabellensätze zählen . . . 197
5.2.8.7 COUNT() - Elemente zählen . . . 198
5.2.8.8 CSV() - BS2000-Datei als Tabelle lesen . . . 200
5.2.8.9 CURRENT_DATE - Aktuelles Datum . . . 204
5.2.8.10 CURRENT_TIME(3) - Aktuelle Uhrzeit . . . 205
5.2.8.11 CURRENT_TIMESTAMP(3) - Aktueller Zeitstempel . . . 206
5.2.8.12 DATE_OF_JULIAN_DAY() - Julianische Tagesnummer umwandeln . . . 207
5.2.8.13 DECRYPT() - Daten entschlüsseln . . . 208
5.2.8.14 DEE() - Tabelle ohne Spalten . . . 211
5.2.8.15 ENCRYPT() - Daten verschlüsseln . . . 212
5.2.8.16 EXTRACT() - Bestandteile eines Zeitwertes extrahieren . . . 214
5.2.8.17 FLOOR() - größte ganze Zahl kleiner als der Wert . . . 216
5.2.8.18 HEX_OF_VALUE() - Beliebigen Wert in Hexadezimalform darstellen . . . 217
5.2.8.19 JULIAN_DAY_OF_DATE() - Datum umwandeln . . . 220

5.2.8.20 LOCALTIME(3) - Aktuelle Ortszeit 222
5.2.8.21 LOCALTIMESTAMP(3) - Aktueller Ortszeitstempel 223
5.2.8.22 LOWER() - Großbuchstaben umwandeln 224
5.2.8.23 MAX() - Maximum bestimmen 225
5.2.8.24 MIN() - Minimum bestimmen 227
5.2.8.25 MOD() - Rest einer Ganzzahl-Division (Modulo) 229
5.2.8.26 NORMALIZE() - National-Zeichenkette in Normalform bringen 230
5.2.8.27 OCTET_LENGTH() - Zeichenkettenlänge bestimmen 232
5.2.8.28 POSITION() - Zeichenkettenposition bestimmen 233
5.2.8.29 REP_OF_VALUE() - Beliebigen Wert als Zeichenkette darstellen 234
5.2.8.30 SIGN() - Signum bestimmen 236
5.2.8.31 SUBSTRING() - Teilzeichenkette extrahieren 237
5.2.8.32 SUM() - Summe berechnen 240
5.2.8.33 TRANSLATE() - Zeichenkette transliterieren bzw. transcodieren 242
5.2.8.34 TRIM() - Zeichen entfernen 245
5.2.8.35 TRUNC() - Nachkommastellen entfernen 247
5.2.8.36 UPPER() - Kleinbuchstaben umwandeln 248
5.2.8.37 VALUE_OF_HEX() - Hexadezimalform als Wert darstellen 249
5.2.8.38 VALUE_OF_REP() - Zeichenkette als Wert darstellen 251
5.3 Prädikat 253
5.3.1 Vergleich von zwei Zeilen 255
5.3.1.1 Vergleichsregeln 257
5.3.2 Quantifizierter Vergleich (Vergleich mit den Zeilen einer Tabelle) 260
5.3.3 BETWEEN-Prädikat (Bereichsabfrage) 262
5.3.4 CASTABLE-Prädikat (Konvertierbarkeitsprüfung) 264
5.3.5 IN-Prädikat (Elementabfrage) 265
5.3.6 LIKE-Prädikat (einfacher Mustervergleich) 268
5.3.7 LIKE_REGEX-Prädikat (Mustervergleich mit regulären Ausdrücken) 271
5.3.8 NULL-Prädikat (Vergleich auf NULL-Wert) 278
5.3.9 EXISTS-Prädikat (Existenzabfrage) 280
5.4 Suchbedingung 281
5.5 CASE-Ausdruck 284
5.5.1 CASE-Ausdruck mit Suchbedingung 286
5.5.2 Einfacher CASE-Ausdruck 288
5.5.3 CASE-Ausdruck mit NULLIF 290
5.5.4 CASE-Ausdruck mit COALESCE 291
5.5.5 CASE-Ausdruck mit MIN / MAX 293
5.6 CAST-Ausdruck 295
5.7 Integritätsbedingung 300
5.7.1 Spaltenbedingung 302
5.7.2 Tabellenbedingung 305

5.8 Spaltendefinition ... 308
6 Abfrage-Ausdruck ... 313
6.1 Tabellenangabe ... 315
6.2 SELECT-Ausdruck ... 318
6.2.1 SELECT-Liste - Ergebnisspalten auswählen ... 319
6.2.2 SELECT ... FROM - Tabellen angeben ... 322
6.2.3 SELECT ... WHERE - Ergebnissätze auswählen ... 324
6.2.4 SELECT ... GROUP BY - Ergebnissätze gruppieren ... 326
6.2.5 SELECT ... HAVING - Gruppen auswählen ... 328
6.3 TABLE - Tabellenabfrage ... 329
6.4 Join ... 330
6.4.1 Join-Ausdruck ... 331
6.4.2 Join ohne Join-Ausdruck ... 333
6.4.3 Join-Typen ... 334
6.4.3.1 Cross Join ... 335
6.4.3.2 Inner Join ... 337
6.4.3.3 Outer Join ... 339
6.4.3.4 Union Join ... 340
6.4.3.5 Zusammengesetzter Join ... 341
6.5 Unterabfrage ... 344
6.5.1 Korrelierte Unterabfragen ... 345
6.6 Verbindung von Abfrage-Ausdrücken mit UNION ... 347
6.7 Verbindung von Abfrage-Ausdrücken mit EXCEPT ... 350
6.8 Änderbarkeit von Abfrage-Ausdrücken ... 352
6.8.1 Regeln für änderbare Abfrage-Ausdrücke ... 353
6.8.2 Änderbarer View ... 354
6.8.3 Ändern über Cursor ... 355
7 Routinen ... 356
7.1 Prozeduren (Stored Procedures) ... 358
7.1.1 Erzeugen einer Prozedur ... 359
7.1.2 Ausführen einer Prozedur ... 361
7.1.3 Löschen einer Prozedur ... 362
7.1.4 Beispiele für Prozeduren ... 363
7.2 User Defined Functions (UDFs) ... 367
7.2.1 Erzeugen einer UDF ... 368
7.2.2 Ausführen einer UDF ... 370
7.2.3 Löschen einer UDF ... 371
7.2.4 Unkorrelierte Funktionsaufrufe ... 372
7.2.5 Beispiele für UDFs ... 374
7.3 EXECUTE-Privileg für Routinen ... 375
7.4 Informationen über Routinen ... 376

7.5 Pragmas in Routinen 377
7.6 Kontrollanweisungen in Routinen 379
7.7 COMPOUND-Anweisung in Routinen 380
7.8 Diagnoseinformationen in Routinen 381
8 SQL-Anweisungen 387
8.1 Inhaltliche Zusammenstellung 388
8.1.1 SQL-Anweisungen zur Schemadefinition und -verwaltung 389
8.1.2 SQL-Anweisungen zum Abfragen und Ändern von Daten 391
8.1.3 SQL-Anweisungen zur Transaktionsverwaltung 392
8.1.4 SQL-Anweisungen zur Session-Steuerung 393
8.1.5 SQL-Anweisungen der dynamischen SQL 394
8.1.6 WHENEVER-Anweisung zur ESQL-Fehlerbehandlung 395
8.1.7 SQL-Anweisungen zur Verwaltung der Speicherstruktur 396
8.1.8 SQL-Anweisungen zur Verwaltung von Benutzereinträgen 397
8.1.9 Utility-Anweisungen 398
8.1.10 Kontrollanweisungen 399
8.1.11 Diagnoseanweisungen 400
8.2 Alphabetische Beschreibung 401
8.2.1 Beschreibungsformat 402
8.2.2 SQL-Anweisungen in Routinen 403
8.2.3 Beschreibung der SQL-Anweisungen 407
8.2.3.1 ALLOCATE DESCRIPTOR - SQL-Deskriptorbereich anfordern 409
8.2.3.2 ALTER SPACE - Space-Parameter ändern 411
8.2.3.3 ALTER STOGROUP - Storage Group ändern 413
8.2.3.4 ALTER TABLE - Basistabelle ändern 415
8.2.3.5 CALL - Prozedur ausführen 429
8.2.3.6 CASE - SQL-Anweisungen bedingt ausführen 432
8.2.3.7 CLOSE - Cursor schließen 436
8.2.3.8 COMMIT WORK - Transaktion beenden 437
8.2.3.9 COMPOUND - SQL-Anweisungen in gemeinsamem Kontext ausführen . 439
8.2.3.10 CREATE FUNCTION - User Defined Function (UDF) erzeugen 447
8.2.3.11 CREATE INDEX - Index erzeugen 451
8.2.3.12 CREATE PROCEDURE - Prozedur erzeugen 454
8.2.3.13 CREATE SCHEMA - Schema erzeugen 458
8.2.3.14 CREATE SPACE - Space erzeugen 461
8.2.3.15 CREATE STOGROUP - Storage Group erzeugen 464
8.2.3.16 CREATE SYSTEM_USER - Systemzugang erzeugen 466
8.2.3.17 CREATE TABLE - Basistabelle erzeugen 469
8.2.3.18 CREATE USER - Berechtigungsschlüssel erzeugen 478
8.2.3.19 CREATE VIEW - View erzeugen 479
8.2.3.20 DEALLOCATE DESCRIPTOR - SQL-Deskriptorbereich freigeben 484

8.2.3.21 DECLARE CURSOR - Cursor vereinbaren 485
8.2.3.22 DELETE - Sätze löschen 491
8.2.3.23 DESCRIBE - Datentypen von Ein- und Ausgabewerten abfragen 494
8.2.3.24 DROP FUNCTION - User Defined Function (UDF) löschen 497
8.2.3.25 DROP INDEX - Index löschen 498
8.2.3.26 DROP PROCEDURE - Prozedur löschen 499
8.2.3.27 DROP SCHEMA - Schema löschen 500
8.2.3.28 DROP SPACE - Space löschen 501
8.2.3.29 DROP STOGROUP - Storage Group löschen 503
8.2.3.30 DROP SYSTEM_USER - Systemzugang löschen 504
8.2.3.31 DROP TABLE - Basistabelle löschen 507
8.2.3.32 DROP USER - Berechtigungsschlüssel löschen 509
8.2.3.33 DROP VIEW - View löschen 510
8.2.3.34 EXECUTE - Vorbereitete Anweisung ausführen 511
8.2.3.35 EXECUTE IMMEDIATE - Dynamisch formulierte Anweisung ausführen 515
8.2.3.36 FETCH - Cursor positionieren und Satz lesen 518
8.2.3.37 FOR - SQL-Anweisungen in einer Schleife ausführen 523
8.2.3.38 GET DIAGNOSTICS - Diagnoseinformationen ausgeben 526
8.2.3.39 GET DESCRIPTOR - SQL-Deskriptorbereich lesen 529
8.2.3.40 GRANT - Privilegien vergeben 532
8.2.3.41 IF - SQL-Anweisungen bedingt ausführen 539
8.2.3.42 INCLUDE - ESQL-Programmteile einfügen 541
8.2.3.43 INSERT - Sätze in Tabelle einfügen 542
8.2.3.44 ITERATE - zum nächsten Schleifendurchlauf wechseln 549
8.2.3.45 LEAVE - Schleife oder COMPOUND-Anweisung beenden 550
8.2.3.46 LOOP - SQL-Anweisungen in einer Schleife ausführen 551
8.2.3.47 MERGE - Sätze in Tabelle einfügen oder Spaltenwerte ändern 553
8.2.3.48 OPEN - Cursor öffnen 558
8.2.3.49 PERMIT - Benutzeridentifikation für SESAM/SQL V1.x angeben 561
8.2.3.50 PREPARE - Dynamisch formulierte Anweisungen vorbereiten 562
8.2.3.51 REORG STATISTICS - Globale Statistik neu erzeugen 571
8.2.3.52 REPEAT - SQL-Anweisungen in einer Schleife ausführen 572
8.2.3.53 RESIGNAL - Fehler in lokaler Fehler-Routine melden 574
8.2.3.54 RESTORE - Cursor wiederherstellen 576
8.2.3.55 RETURN - Rückgabewert einer User Defined Function (UDF) liefern . . 577
8.2.3.56 REVOKE - Privilegien entziehen 578
8.2.3.57 ROLLBACK WORK - Transaktion zurücksetzen 585
8.2.3.58 SELECT - Einzelnen Satz lesen 587
8.2.3.59 SET - Wert zuweisen 590
8.2.3.60 SET CATALOG - Datenbanknamen voreinstellen 591
8.2.3.61 SET DESCRIPTOR - SQL-Deskriptorbereich ändern 592

8.2.3.62 SET SCHEMA - Schemanamen voreinstellen 597
8.2.3.63 SET SESSION AUTHORIZATION - Berechtigungsschlüssel festlegen . 599
8.2.3.64 SET TRANSACTION - Transaktionseigenschaften festlegen 601
8.2.3.65 SIGNAL - Fehler in Routine melden 605
8.2.3.66 STORE - Cursorposition speichern 607
8.2.3.67 UPDATE - Spaltenwerte ändern 608
8.2.3.68 WHENEVER - Fehlerbehandlung definieren 613
8.2.3.69 WHILE - SQL-Anweisungen in einer Schleife ausführen 615
9 SESAM-CLI 617
9.1 Konzept des SESAM-CLI 618
9.1.1 Struktur von SESAM-CLI-Aufrufen 619
9.1.2 Transaktionseinleitende Anweisungen in CLI-Aufrufen 623
9.2 Aufrufe des SESAM-CLI 624
9.2.1 Übersicht 625
9.2.2 Alphabetischer Nachschlageteil 628
9.2.2.1 SQL_BLOB_CLS_ISBTAB - SQLbcis 629
9.2.2.2 SQL_BLOB_CLS_REF - SQLbcre 631
9.2.2.3 SQL_BLOB_OBJ_CLONE - SQLbocl 632
9.2.2.4 SQL_BLOB_OBJ_CREATE - SQLbocr 633
9.2.2.5 SQL_BLOB_OBJ_CREAT2 - SQLboc2 634
9.2.2.6 SQL_BLOB_OBJ_DROP - SQLbodr 636
9.2.2.7 SQL_BLOB_TAG_GET - SQLbtge 637
9.2.2.8 SQL_BLOB_TAG_PUT - SQLbtpu 639
9.2.2.9 SQL_BLOB_VAL_CLOSE - SQLbvcl 641
9.2.2.10 SQL_BLOB_VAL_FETCH - SQLbvfe 642
9.2.2.11 SQL_BLOB_VAL_GET - SQLbvge 644
9.2.2.12 SQL_BLOB_VAL_LEN - SQLbvle 646
9.2.2.13 SQL_BLOB_VAL_OPEN - SQLbvop 647
9.2.2.14 SQL_BLOB_VAL_PUT - SQLbvpu 649
9.2.2.15 SQL_BLOB_VAL_STOW - SQLbvst 650
9.2.2.16 SQL_DIAG_SEQ_GET - SQLdsg 652
10 Informationsschemata 654
10.1 Views des INFORMATION_SCHEMA 655
10.1.1 BASE_TABLES 657
10.1.2 BASE_TABLE_COLUMNS 658
10.1.3 CATALOG_PRIVILEGES 662
10.1.4 CHARACTER_SETS 663
10.1.5 CHECK_CONSTRAINTS 664
10.1.6 COLLATIONS 665
10.1.7 COLUMNS 666
10.1.8 COLUMN_PRIVILEGES 669

10.1.9 CONSTRAINT_COLUMN_USAGE 670
10.1.10 CONSTRAINT_TABLE_USAGE 671
10.1.11 DA_LOGS 672
10.1.12 INDEXES 673
10.1.13 INDEX_COLUMN_USAGE 674
10.1.14 KEY_COLUMN_USAGE 675
10.1.15 MEDIA_DESCRIPTIONS 676
10.1.16 MEDIA_RECORDS 677
10.1.17 PARAMETERS 678
10.1.18 PARTITIONS 681
10.1.19 RECOVERY_UNITS 682
10.1.20 REFERENTIAL_CONSTRAINTS 684
10.1.21 ROUTINES 685
10.1.22 ROUTINE_COLUMN_USAGE 689
10.1.23 ROUTINE_PRIVILEGES 690
10.1.24 ROUTINE_ROUTINE_USAGE 691
10.1.25 ROUTINE_TABLE_USAGE 692
10.1.26 SCHEMATA 693
10.1.27 SPACES 694
10.1.28 SQL_FEATURES 695
10.1.29 SQL_IMPL_INFO 696
10.1.30 SQL_LANGUAGES_S 697
10.1.31 SQL_SIZING 698
10.1.32 STOGROUPS 699
10.1.33 STOGROUP_VOLUME_USAGE 700
10.1.34 SYSTEM_ENTRIES 701
10.1.35 TABLES 702
10.1.36 TABLE_CONSTRAINTS 703
10.1.37 TABLE_PRIVILEGES 704
10.1.38 TRANSLATIONS 705
10.1.39 USAGE_PRIVILEGES 706
10.1.40 USERS 707
10.1.41 VIEWS 708
10.1.42 VIEW_COLUMN_USAGE 709
10.1.43 VIEW_ROUTINE_USAGE 710
10.1.44 VIEW_TABLE_USAGE 711
10.2 Views des SYS_INFO_SCHEMA 712
10.2.1 SYS_CATALOGS 714
10.2.2 SYS_CHECK_CONSTRAINTS 715
10.2.3 SYS_CHECK_USAGE 716
10.2.4 SYS_COLUMNS 717

10.2.5 SYS_DA_LOGS 720
10.2.6 SYS_DBC_ENTRIES 721
10.2.7 SYS_DML_RESOURCES 723
10.2.8 SYS_ENVIRONMENT 724
10.2.9 SYS_INDEXES 725
10.2.10 SYS_LOCK_CONFLICTS 727
10.2.11 SYS_MEDIA_DESCRIPTIONS 730
10.2.12 SYS_PARAMETERS 731
10.2.13 SYS_PARTITIONS 733
10.2.14 SYS_PRIVILEGES 734
10.2.15 SYS_RECOVERY_UNITS 735
10.2.16 SYS_REFERENTIAL_CONSTRAINTS 737
10.2.17 SYS_ROUTINES 738
10.2.18 SYS_ROUTINE_ERRORS 740
10.2.19 SYS_ROUTINE_PRIVILEGES 742
10.2.20 SYS_ROUTINE_ROUTINE_USAGE 743
10.2.21 SYS_ROUTINE_USAGE 744
10.2.22 SYS_SCHEMATA 745
10.2.23 SYS_SPACES 746
10.2.24 SYS_SPACE_PROPERTIES 747
10.2.25 SYS_SPECIAL_PRIVILEGES 749
10.2.26 SYS_STOGROUPS 750
10.2.27 SYS_SYSTEM_ENTRIES 751
10.2.28 SYS_TABLES 752
10.2.29 SYS_TABLE_CONSTRAINTS 754
10.2.30 SYS_UNIQUE_CONSTRAINTS 755
10.2.31 SYS_USAGE_PRIVILEGES 756
10.2.32 SYS_USERS 757
10.2.33 SYS_VIEW_USAGE 758
10.2.34 SYS_VIEW_ROUTINE_USAGE 759
11 Anhang 760
11.1 Syntaxelemente von SESAM/SQL 761
11.2 Syntaxübersicht CSV-Datei 773
11.3 SQL-Schlüsselwörter 775
12 Literatur 785

Sprachbeschreibung Teil 1

1 Einleitung

Das Datenbanksystem SESAM/SQL-Server erfüllt durch seine Funktionen und seine Architekturmerkmale alle Anforderungen, die heute an einen leistungsfähigen Datenbankserver gestellt werden. Diese Eigenschaft drückt sich auch im Produktnamen SESAM/SQL-Server aus.

SESAM/SQL-Server gibt es als Standard Edition mit Singletask-Betrieb und als Enterprise Edition, die den Multitask-Betrieb beinhaltet.

Der Einfachheit halber ist im Folgenden von SESAM/SQL die Rede, wenn das Datenbanksystem SESAM/SQL-Server gemeint ist.

Folgende einleitenden Beschreibungen befinden sich zentral im „ Basishandbuch“:

- Kurzbeschreibung des Produkts
- Konzept der SESAM/SQL-Server-Dokumentation
- Beispieldatenbank
- Readme-Datei
- Änderungen gegenüber den Vorgänger-Handbüchern

1.1 Zielsetzung und Zielgruppen des Handbuchs

Das Handbuch richtet sich an alle SESAM/SQL-Benutzer, die mit SQL arbeiten.

Das „ Basishandbuch“ wird als bekannt vorausgesetzt, insbesondere die SESAM/SQL-Objekte und -Konzepte, auf denen die SQL-Anweisungen basieren. Zusätzlich werden grundlegende Kenntnisse über relationale Datenbanken vorausgesetzt.

Wenn Sie SQL-Anweisungen interaktiv über den Utility-Monitor aufrufen, müssen Sie mit dem Utility-Monitor vertraut sein (siehe Handbuch „ Utility-Monitor“).

Wenn Sie SQL-Anweisungen in ein Programm einbetten, müssen Sie mit der Programmiersprache COBOL und dem ESQL-Precompiler (siehe Handbuch „ ESQL-COBOL für SE-SAM/SQL-Server“) vertraut sein.

1.2 Konzept des Handbuchs

Dieses Handbuch ist eine vollständige Beschreibung der Datenbanksprache SQL des Datenbanksystems SESAM /SQL. Die Abweichungen und Erweiterungen gegenüber der SQL-Norm sind beschrieben.

Das Kapitel „Programmeinbettung von SQL“ beschreibt die SQL-spezifischen Konzepte für die Verwendung von SQL-Anweisungen in einer Wirtssprache (COBOL). Die restlichen Kapitel beschreiben die SQL-Sprachelemente in der Reihenfolge, in der sie aufeinander aufbauen. In jedem Kapitel werden die Sprachelemente aus vorherigen Kapiteln als bekannt vorausgesetzt und nicht mehr erklärt.

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

Das Kapitel „SESAM-CLI“ beschreibt die Struktur der SESAM-CLI-Schnittstelle. Mit dieser Schnittstelle ist es möglich, BLOB-Objekte zu erstellen und zu bearbeiten. Im alphabetischen Nachschlageteil dieses Kapitels sind die einzelnen CLI-Aufrufe detailliert erklärt.

Das Kapitel „Informationsschemata“ beschreibt die Views der Schemata INFORMATION_SCHEMA und SYS_INFO_SCHEMA.

Der Anhang ist ein alphabetischer Nachschlageteil zu verwendeten Syntaxen und reservierten Schlüsselwörtern von SESAM/SQL.

Am Ende des Buches finden Sie ein Literatur- und Stichwortverzeichnis.

Das Handbuch enthält zahlreiche Beispiele. Sie beziehen sich stets auf den jeweils zuvor beschriebenen Sachverhalt. Einige Beispiele für SQL-Sprachelemente, insbesondere für Ausdrücke und Abfrage-Ausdrücke, laufen nur innerhalb einer übergeordneten SQL-Anweisung ab und sind für sich genommen nicht ablauffähig.

1.3 Darstellungsmittel

In diesem Handbuch verwenden wir folgende Darstellungsmittel:

—————— ——————	Syntaxdefinitionen; Fortsetzungszeilen innerhalb von Syntaxdefinitionen sind eingerückt.
GROSS	SQL-Schlüsselwörter
unterstrichen	Voreinstellungen
fett	Hervorhebung im Fließtext
kursiv	Variablen in Syntaxdefinitionen und im Fließtext
`Schreibmaschinenschrift`	Programmtext in Syntaxdefinitionen und in Beispielen

`::=`	Definitionszeichen Die Angabe auf der rechten Seite von `::=` definiert die Syntax für das Element auf der linken Seite.
|	In Syntaxdefinitionen trennt dieses Zeichen die alternativen Angaben.
[ ]	Optionale Angaben Die eckigen Klammern sind Metazeichen, die in einer SQL-Anweisung nicht angegeben werden.
`{ | }`	Alternative Angaben in Syntaxdefinitionen (einzeilig), wobei eine der Alternativen angegeben werden muss.

`{`
`|`
`}`
	Alternative Angaben in Syntaxdefinitionen (mehrzeilig). Jede Zeile enthält eine Alternative, wobei eine der Alternativen angegeben werden muss.
	Die geschweiften Klammern sind Metazeichen, die in einer SQL-Anweisung nicht angegeben werden.

,...	In Syntaxdefinitionen bedeutet diese Schreibweise, dass die vorausgehende Angabe beliebig oft, durch Komma getrennt, wiederholt werden kann. Wird keine Wiederholung angegeben, fällt auch das Komma weg.
...	In Syntaxdefinitionen bedeuten die Punkte, dass die vorausgehende Angabe beliebig oft wiederholt werden kann. In Beispielen bedeuten die Punkte, dass die restlichen Teile für das Beispiel ohne Bedeutung sind. Die Punkte sind Metazeichen, die in einer SQL-Anweisung nicht angegeben werden.

Hinweise auf besonders wichtige Informationen

> **!** Warnhinweise

Die Zeichenfolgen `<date>`, `<time>` und `<ver>` bezeichnen in Beispielen die aktuellen Ausgaben für Datum, Uhrzeit und Version, wenn die Beispiele sonst Datums-, Zeit- und Versions-unabhängig sind.

2 Programmeinbettung von SQL

Um von einem Programm aus auf die Datenbank zugreifen zu können, gibt es programmiersprachenspezifische Schnittstellen, die es ermöglichen, SQL-Anweisungen in ein Programm einzubinden. SESAM/SQL bietet eine Schnittstelle für die Programmiersprache COBOL.

Die Konzepte für die Programmeinbettung sind für alle Programmiersprachen gleich und werden unter dem Begriff ESQL (Embedded SQL) zusammengefasst. Ein Programm mit eingebetteten SQL-Anweisungen heißt ESQL-Programm.

In diesem Kapitel sind die Konzepte für die Einbettung von SQL-Anweisungen erklärt. Es behandelt im Einzelnen folgende Themen:

- Programmstruktur
- Benutzervariable
- Erfolgskontrolle und Fehlerbehandlung
- Cursor
- Dynamische SQL

Die sprachspezifischen Einzelheiten sind im Handbuch „ ESQL-COBOL für SESAM/SQL-Server“ beschrieben.

2.1 Programmstruktur

Ein ESQL-Programm besteht aus Programmtext in der jeweiligen Programmiersprache, auch Wirtssprache (host language) genannt und SQL-Anweisungen. Die SQL-Anweisungen dürfen an allen Stellen stehen, an denen Anweisungen der Wirtssprache erlaubt sind. Anfang und Ende einer SQL-Anweisung werden markiert, um die SQL-Anweisungen von den Anweisungen der Wirtssprache unterscheiden zu können. Die Markierungen sind programmiersprachenspezifisch.

Werden in den SQL-Anweisungen Variablen der Wirtssprache (Benutzervariablen) verwendet, enthält das Programm zusätzlich Abschnitte (DECLARE SECTION), in denen diese Variablen definiert werden. DECLARE SECTIONs dürfen an allen Stellen stehen, an denen Variablendefinitionen der Wirtssprache erlaubt sind. Anfang und Ende einer DECLARE SECTION werden mit EXEC SQL BEGIN DECLARE SECTION bzw. EXEC SQL END DECLARE SECTION markiert (die genaue Syntax ist sprachspezifisch und im Handbuch „ ESQL-COBOL für SESAM/SQL-Server“ beschrieben). Ein ESQL-Programm kann beliebig viele DECLARE SECTIONs enthalten.

ESQL-COBOL-Programme mit Ablaufbeispielen für Datenbank-Anweisungen finden Sie in der Beispieldatenbank von SESAM/SQL (siehe „ Basishandbuch“).

2.2 Benutzervariable

Eine Benutzervariable ist eine Variable der Wirtssprache, die in einer eingebetteten SQL-Anweisung verwendet werden kann. Eine Benutzervariable dient dazu, Werte aus der Datenbank in das Programm der Wirtssprache zu übertragen und dort weiterzuverarbeiten oder umgekehrt Daten in die Datenbank zu übertragen und Werte bereitzustellen, die in Berechnungen benötigt werden.

2.2.1 Benutzervariable definieren

Eine Benutzervariable muss im Programm in einer DECLARE SECTION entsprechend den Konventionen der Programmiersprache definiert werden. Die Stellen der Definition und der Verwendung einer Benutzervariablen müssen folgende Bedingungen erfüllen:

- Die Definition muss im Programmtext vor dem Gebrauch der Variable in einer SQL-Anweisung stehen.
- Die Definition muss für jede Verwendung der Variable in einer SQL-Anweisung oder in einer Anweisung der Wirtssprache gemäß den Konventionen der Programmiersprache gültig sein.
- Die Definition einer Variablen, die in einer DECLARE CURSOR-Anweisung (Cursor vereinbaren) verwendet wird, muss für alle OPEN-Anweisungen des vereinbarten Cursor gültig sein.

Der Datentyp der Benutzervariablen richtet sich nach dem Datentyp der SESAM/SQL-Werte, für die diese Benutzervariable verwendet werden soll. Die ESQL-Sprachschnittstelle stellt vordefinierte Datentypen bereit, die für die Benutzervariablen verwendet werden müssen. Im Handbuch „ ESQL-COBOL für SESAM/SQL-Server“ ist für jeden SESAM/SQL-Datentyp der zugeordnete COBOL-Datentyp angegeben.

2.2.2 Benutzervariable verwenden

In SQL-Anweisungen, die Daten aus der Datenbank abfragen, können die gelesenen Werte in Benutzervariablen abgespeichert werden.

In SQL-Anweisungen, die Werte in die Datenbank einfügen, Werte in der Datenbank ändern oder in denen Berechnungen (Funktionen, Ausdrücke, Prädikate, Suchbedingungen) vorkommen, können die Werte über Benutzervariablen bereitgestellt werden.

Weitere Fälle, in denen Werte in SQL-Anweisungen über Benutzervariablen bereitgestellt werden können oder müssen, sind im Kapitel „SQL-Anweisungen“ bei der Beschreibung der jeweiligen SQL-Anweisung angegeben.

Sie geben eine Benutzervariable in einer SQL-Anweisung mit einem vorangestellten Doppelpunkt an:

: benutzervariable

Benutzervariablen können auch Vektoren sein, die mehrere Werte gleichen Datentyps enthalten. Sie können damit multiplen Spalten Aggregate zuweisen oder Aggregate aus multiplen Spalten an die Benutzervariablen übergeben. Die Syntax von Vektoren ist sprachspezifisch und im Handbuch „ ESQL-COBOL für SESAM/SQL-Server“ beschrieben.

2.2.3 Indikatorvariable

Eine Benutzervariable kann mit einer weiteren Benutzervariablen, genannt Indikatorvariable, kombiniert werden. Eine Indikatorvariable dient dazu, den in Programmiersprachen nicht vorhandenen NULL-Wert auszudrücken und die Übertragung alphanumerischer Werte und National-Werte aus der Datenbank zu kontrollieren.

2.2.3.1 Indikatorvariable definieren

Eine Benutzervariable, die als Indikatorvariable verwendet wird, muss mit dem Datentyp der Wirtssprache, der dem SQL-Datentyp SMALLINT entspricht, definiert sein. Der genaue Datentyp ist im Handbuch „ ESQL-COBOL für SESAM/SQL-Server“ angegeben.

2.2.3.2 Indikatorvariable verwenden

Eine Benutzervariable darf nur mit einer Indikatorvariablen kombiniert werden, wenn sie zum Abfragen von Daten aus der Datenbank, Einfügen von Werten in die Datenbank, Ändern von Werten in der Datenbank oder in Berechnungen (Funktionen, Ausdrücke, Prädikate, Suchbedingungen) verwendet wird.

Die Indikatorvariable wird hinter der Benutzervariablen angegeben. Sie kann durch das Schlüsselwort INDICATOR getrennt werden:
`:`*benutzervariable* `[INDICATOR] :`*indikatorvariable*

Ist die Benutzervariable ein Vektor, so muss die zugehörige Indikatorvariable ebenfalls ein Vektor mit derselben Anzahl von Elementen sein. Jedem Element der Benutzervariablen ist das entsprechende Element der Indikatorvariablen zugeordnet. Die Syntax von Vektoren ist im Handbuch „ ESQL-COBOL für SESAM/SQL-Server“ beschrieben.

Werte abfragen

Bei der Abfrage eines Wertes aus der Datenbank mit Zuweisung an eine Benutzervariable wird die zugehörige Indikatorvariable von SESAM/SQL wie folgt belegt:

0	Die Benutzervariable enthält den gelesenen Wert. Die Zuweisung war fehlerfrei.
-1	Der Wert, der zugewiesen werden sollte, ist der NULL-Wert.
> 0	Bei alphanumerischen Werten oder National-Werten: Der Benutzervariablen wurde eine verkürzte Zeichenkette zugewiesen. Der Wert der Indikatorvariablen gibt die Originallänge in Code Units an.

Werte einfügen oder ändern

Wenn Sie in SQL-Anweisungen Werte über Benutzervariablen angeben, können Sie die Indikatorvariable verwenden, um einen NULL-Wert anzugeben. Dazu müssen Sie der Indikatorvariablen vor Aufruf der SQL-Anweisung einen negativen Wert zuweisen. Bei Ausführung der SQL-Anweisung wird dann nicht der Wert der Benutzervariablen verwendet, sondern der NULL-Wert.

2.3 Erfolgskontrolle und Fehlerbehandlung

Nach Ausführung einer SQL-Anweisung sollte im ESQL-Programm abgefragt werden, ob die Ausführung erfolgreich war, um auf Fehlerfälle gezielt reagieren zu können.

2.3.1 Erfolgskontrolle

Um zu prüfen, ob eine Anweisung erfolgreich war, verwenden Sie die von SESAM/SQL in der ESQL-Schnittstelle unterstützte Benutzervariable SQLSTATE.

Sie müssen SQLSTATE in Ihrem Programm in einer DECLARE SECTION mit dem SQL-Datentyp CHAR(5) definieren. Die Definition muss im Programmtext vor der ersten SQL-Anweisung stehen und gemäß den Konventionen der Programmiersprache für alle sie verwendenden Anweisungen gültig sein.

SQLSTATE enthält nach Ausführung einer SQL-Anweisung einen SQL-Statuscode. Die möglichen Werte für SQLSTATE sind im Handbuch „ Meldungen“ beschrieben.

Aus Kompatibilitätsgründen zu SESAM/SQL V1.x wird auch noch die Benutzervariable SQLCODE für die Erfolgskontrolle unterstützt. Sie sollten diese jedoch in neuen Anwendungen nicht verwenden.

2.3.2 Fehlerbehandlung

Es gibt zwei Möglichkeiten, gezielt zu reagieren, wenn eine SQL-Anweisung nicht erfolgreich war:

- SQLSTATE abfragen und abhängig vom Statuscode verzweigen
- die WHENEVER-Anweisung verwenden

  Mit WHENEVER können Sie festlegen, dass nach Ausführung einer SQL-Anweisung mit einem SQLSTATE '00*xxx*' und '01*xxx*' das Programm entweder fortgesetzt wird oder zu einer bestimmten Stelle im Programm verzweigt wird, an der eine Fehlerbehandlung vorgenommen wird. Die Möglichkeit der Programmverzweigung können Sie für zwei Fehlerklassen festlegen:

  - NOT FOUND: keine Daten verfügbar, zum Beispiel bei Erreichen des Tabellenendess
  - SQLERROR: sonstige Fehler, die zum Abbruch der SQL-Anweisungen führten

  Die WHENEVER-Anweisung kann mehrmals in einem Programm vorkommen. Die Angaben einer WHENEVER-Anweisung gelten für alle im Programmtext folgenden SQL-Anweisungen bis zur nächsten WHENEVER-Anweisung für dieselbe Fehlerklasse.

2.4 Cursor

Da es in vielen Programmiersprachen für den Typ Tabelle keine Entsprechung gibt, wird für die Programmeinbettung das Konzept des Cursors verwendet. Ein Cursor ermöglicht es, die Sätze einer Tabelle nacheinander einzeln abzuarbeiten.

Ein Cursor ist einer Tabelle, genannt Cursortabelle, zugeordnet. Diese Tabelle ist die Ergebnistabelle des Abfrageausdrucks, mit dem der Cursor definiert wird.

Für die Arbeit mit Cursorn gibt es eine Reihe von SQL-Anweisungen:

DECLARE CURSOR	Cursor vereinbaren
OPEN	Cursor öffnen
CLOSE	Cursor schließen
FETCH	Cursor positionieren und Satz lesen
DELETE ... WHERE CURRENT OF ...	aktuellen Satz löschen
UPDATE ... WHERE CURRENT OF ...	aktuellen Satz ändern
STORE	Cursorposition speichern
RESTORE	Cursorposition wiederherstellen

Ein Cursor muss definiert, vor Gebrauch geöffnet und nach Gebrauch geschlossen werden. Die SQL-Anweisungen müssen in einer vorgeschriebenen Reihenfolge verwendet werden.

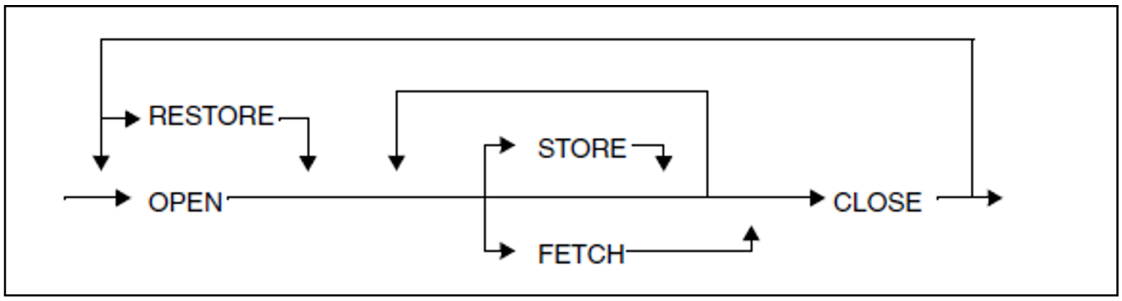
Es gibt nicht-änderbare Cursor und änderbare Cursor.

> ¡ In Routinen werden lokale Cursor mit der DECLARE CURSOR-Anweisung definiert, die **nur** innerhalb der COMPOUND-Anweisung angesprochen werden können, siehe Abschnitt „Lokale Cursor“.
>
> Ein lokaler Cursor unterscheidet sich von einem gewöhnlichen Cursor nur durch den beschränkten Gültigkeitsbereich.

2.4.1 Cursor zum Lesen

Ein nicht-änderbarer Cursor kann nur zum Lesen von Sätzen aus der Cursortabelle verwendet werden und heißt daher auch Cursor zum Lesen.

Die möglichen SQL-Anweisungen bei einem nicht-dynamischen Cursor zum Lesen und ihre Reihenfolge zeigt folgende Zusammenfassung:



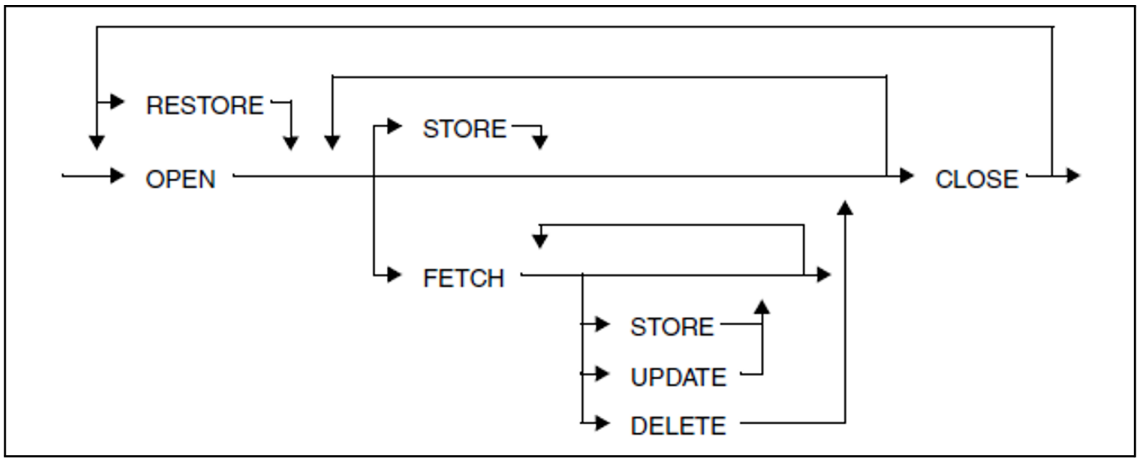
Das Öffnen des Cursors mit RESTORE ist nur nach einem STORE möglich. Wenn eine Cursorposition gespeichert ist, ist FETCH nicht erlaubt.

Die zusätzlichen Anweisungen für dynamische Cursor sind im Abschnitt „Dynamisch formulierte Cursorbeschreibung“ angegeben.

2.4.2 Änderbarer Cursor

Ein änderbarer Cursor kann zusätzlich dazu verwendet werden, Sätze aus einer Tabelle zu löschen oder zu ändern.

Die möglichen SQL-Anweisungen bei einem nicht-dynamischen, änderbaren Cursor und ihre Reihenfolge zeigt folgende Zusammenfassung:



Das Öffnen des Cursors mit RESTORE ist nur nach einem STORE möglich. Wenn eine Cursorposition gespeichert ist, ist FETCH nicht erlaubt.

Die zusätzlichen Anweisungen für dynamische Cursor sind im Abschnitt „Dynamisch formulierte Cursorbeschreibung“ angegeben.

2.4.3 Cursor definieren

Ein Cursor wird mit der DECLARE CURSOR-Anweisung definiert. Bei der Definition wird dem Cursor die Cursorbeschreibung zugeordnet. Die Cursorbeschreibung ist der Abfrageausdruck, der die Cursortabelle definiert.

Für statische Cursor und lokale Cursor (in Routinen) wird der Abfrageausdruck in der DECLARE CURSOR-Anweisung direkt angegeben. Für dynamische Cursor wird er zur Laufzeit des Programms aufgebaut (siehe Abschnitt „Dynamisch formulierte Cursorbeschreibung“).

Bei der Definition können folgende Eigenschaften des Cursors festgelegt werden:

Positionierung

Ein Cursor kann entweder frei positionierbar sein oder nicht.

Ein frei positionierbarer Cursor kann auf jeden Satz der Cursortabelle positioniert werden. Er wird durch Angabe des Schlüsselworts SCROLL definiert.

Ein mit NO SCROLL definierter Cursor kann nur auf den jeweils nächsten Satz positioniert werden.

Lebensdauer

Soll ein Cursor auch nach Beendigung einer Transaktion geöffnet bleiben, kann dies durch Angabe der WITH HOLD-Klausel realisiert werden. Voraussetzung dafür ist aber, dass der Cursor vor dem Ende der Transaktion in geöffnetem Zustand ist. Die WITH HOLD-Klausel ist bei lokalen Cursorn (in Routinen) nicht erlaubt.

Ein mit WITHOUT HOLD definierter Cursor wird bei Transaktionsende implizit geschlossen. WITHOUT HOLD ist der Default-Wert.

Sortierung

In der Cursorbeschreibung kann eine ORDER BY-Klausel angegeben werden, um die Sätze der Cursortabelle zu sortieren.

Treffermenge

In der Cursorbeschreibung kann eine Klausel FETCH FIRST *max* ROWS ONLY zur Begrenzung der gelieferten Treffermenge angegeben werden.

Änderbarkeit

Ein Cursor heißt änderbar, wenn der Abfrageausdruck, mit dem der Cursor definiert wurde, änderbar ist (siehe Abschnitt „Änderbarkeit von Abfrage-Ausdrücken“) und weder SCROLL noch ORDER BY noch die FOR READ ONLY-Klausel bei der Cursorvereinbarung angegeben wurden.

Ein änderbarer Cursor bezieht sich auf genau eine Basistabelle. In dieser Tabelle können dann einzelne Sätze, die mit Hilfe der Cursorposition bestimmt werden, gelöscht oder geändert werden. Mit der FOR UPDATE-Klausel in der Cursorvereinbarung können bei einem änderbaren Cursor diejenigen Spalten angegeben werden, deren Werte geändert werden dürfen.

Ist der Cursor nicht änderbar, kann er nur zum Lesen der Sätze aus der zugeordneten Cursortabelle verwendet werden. Bei `FETCH FIRST` *max* `ROWS ONLY` ist ein Cursor ebenfalls nicht änderbar.

2.4.4 Cursor öffnen

Ein Cursor muss vor Gebrauch geöffnet werden.

Zum Öffnen eines Cursors gibt es die Anweisung OPEN. Die Werte für die in der Cursorbeschreibung enthaltene Benutzervariablen, Spezial-Literale (siehe "Spezial-Literale") und Zeitfunktionen (CURRENT_DATE, CURRENT_TIME(3), CURRENT_TIMESTAMP(3), etc.) werden bestimmt. Nach dem Öffnen steht der Cursor vor dem ersten Satz der zugeordneten Cursortabelle (siehe auch Abschnitt „OPEN - Cursor öffnen“).

2.4.5 Cursor positionieren und Satz lesen

Um einen Satz der Cursortabelle zu lesen, müssen Sie den Cursor mit FETCH auf diesen Satz positionieren. Die Spaltenwerte des aktuellen Satzes werden dabei in Benutzervariablen oder in einem Deskriptorbereich (siehe Abschnitt „Deskriptorbereich“) abgelegt.

Um den nächsten Satz zu lesen, muss der Cursor erneut positioniert werden. Ein mit SCROLL vereinbarter Cursor kann beliebig positioniert werden. Ohne SCROLL bzw. mit NO SCROLL definierte Cursor können nur auf den jeweils nächsten Satz positioniert werden.

2.4.6 Satz ändern oder löschen

Bei einem änderbaren Cursor können Sie nach dem Positionieren des Cursors einen Satz in der Basistabelle, die der Cursorbeschreibung zu Grunde liegt, ändern oder löschen. Dazu verwenden Sie die Anweisung UPDATE... WHERE CURRENT OF bzw. DELE-TE...WHERE CURRENT OF.

Die Änderungs- oder Löschoperation bezieht sich auf den Satz an der aktuellen Position des Cursors in der Cursortabelle. Die Position des Cursors wird durch die Änderungsoperation nicht verändert. Nach einer Löschoperation steht der Cursor vor dem nächsten Satz in der Cursortabelle (oder nach dem letzten Satz, wenn das Tabellenende erreicht ist). Für eine weitere Änderungs- oder Löschoperation muss der Cursor erst mit FETCH positioniert werden.

2.4.7 Cursor speichern

Um die Cursortabelle und -position über das Ende der aktuellen Transaktion hinaus zu erhalten, kann der Cursor mit STORE gespeichert werden. Zwischen STORE und dem nachfolgenden Schließen des Cursors kann die Cursortabelle nicht mehr mit FETCH gelesen werden. STORE ist bei lokalen Cursorn (in Routinen) nicht erlaubt.

Eine einfachere Möglichkeit einen Cursor über mehrere Transaktionen hinweg offen zu halten, bietet die Definition des Cursors mit WITH HOLD-Klausel. Die WITH HOLD-Klausel ist bei lokalen Cursorn (in Routinen) nicht erlaubt.

2.4.8 Cursor schließen

Ein Cursor wird mit CLOSE geschlossen.

Außerdem wird ein Cursor geschlossen, wenn die Transaktion beendet wird, in der der Cursor geöffnet wurde. Dies gilt aber nicht, falls der Cursor mit WITH HOLD spezifiziert wurde und die Transaktion nicht zurückgesetzt wird.

2.4.9 Cursor wiederherstellen

Ein gespeicherter Cursor kann mit RESTORE wiederhergestellt werden. Der Cursor wird geöffnet und der Zugriff auf die Cursortabelle ist wieder möglich. RESTORE ist bei lokalen Cursorn (in Routinen) nicht erlaubt.

Die gespeicherte Information kann unter Umständen verlorengehen. Diese Fälle sind im Abschnitt „RESTORE - Cursor wiederherstellen“ beschrieben.

2.4.10 Cursor-Beispiele

Beispiel für einen Cursor mit ORDER BY

Der Cursor CUR_KONTAKTE definiert einen Ausschnitt aus der Tabelle KONTAKT, in dem Nachname, Vorname und Abteilung für Kunden mit Kundennummern > 103 ausgewählt werden. Die Sätze sollen aufsteigend nach Abteilung und innerhalb der Abteilung absteigend nach dem Nachnamen sortiert werden.

```
DECLARE cur_kontakte CURSOR FOR
SELECT nachname, vorname, abteilung
FROM kontakt WHERE knr > 103
ORDER BY abteilung ASC, nachname DESC
```

Der Cursor wird mit OPEN geöffnet

```
OPEN cur_kontakte
```

Zu diesem Zeitpunkt umfasst die Cursortabelle folgende Sätze:

nachname	vorname	abteilung
Buschmann	Anke	
Bauer	Xaver	
Heinlein	Robert	Einkauf
Davis	Mary	Einkauf

NULL-Werte sind in obiger Tabelle als leere Felder dargestellt. Bei der Sortierung mit ORDER BY werden in SESAM /SQL NULL-Werte als kleiner als alle Nicht-NULL-Werte betrachtet.

In einem ESQL-Programm kann die Cursortabelle satzweise in einer Schleife gelesen werden. Die Spaltenwerte werden in die Benutzervariablen NAME, VORNAME und AB-TEILUNG übertragen.

```
FETCH cur_kontakte INTO :NACHNAME,
:VORNAME INDICATOR :INDVORNAME,
:ABTEILUNG INDICATOR :INDABTEILUNG
```

Beispiel für eine SQL-Datenmanipulation mit Cursor

Der Cursor CUR_MWST wählt alle Leistungen aus, für die keine Mehrwertsteuer berechnet wird. Er wird mit WITH HOLD spezifiziert, so dass er auch nach einem COM-MIT WORK im geöffneten Zustand bleibt, wenn er am Ende der Transaktion im geöffneten Zustand war:

```
DECLARE cur_mwst CURSOR WITH HOLD FOR
SELECT lnr, ltext, mwsatz
FROM leistung WHERE mwsatz=0.00


OPEN cur_mwst
```

Zum Zeitpunkt des OPEN erhält man die folgende Cursortabelle:

lnr	ltext	mwsatz
4	Systemanalyse	0.00
5	Datenbankentwurf	0.00
10	Reisekosten	0.00

Für diese Leistungen soll ein Mehrwertsteuersatz von 15% berücksichtigt werden. Mit einer Folge von FETCH- und UPDATE-Anweisungen können die Sätze der Tabelle LEISTUNG geändert werden. Mit FETCH NEXT wird der Cursor auf den ersten Satz positioniert:

```
FETCH NEXT cur_mwst INTO :LNR,
:LTEXT INDICATOR :INDLTEXT
:MWSATZ INDICATOR :INDMWSATZ

UPDATE leistung SET mwsatz=0.15 WHERE CURRENT OF cur_mwst
```

Anschließend wird der Cursor auf den zweiten Satz der Cursortabelle positioniert:

```
FETCH NEXT cur_mwst INTO :LNR,
:LTEXT INDICATOR :INDLTEXT
:MWSATZ INDICATOR :INDMWSATZ

UPDATE leistung SET mwsatz=0.15 WHERE CURRENT OF cur_mwst
```

Die Transaktion wird mit COMMIT WORK geschlossen. Aufgrund der WITH HOLD-Klausel kann der Cursor auf den dritten Satz der Cursortabelle positioniert werden, indem unmittelbar auf COMMIT WORK eine FETCH-Anweisung eingegeben wird.

2.5 Dynamische SQL

SESAM/SQL bietet die Möglichkeit, SQL-Anweisungen und Cursorbeschreibungen dynamisch bei Ablauf des ESQL-Programms zu erzeugen. Konzepte und Sprachmittel, die damit verbunden sind, werden unter dem Begriff dynamische SQL zusammengefasst und sind in diesem Abschnitt erklärt.

Eine dynamisch formulierte Anweisung (bzw. Cursorbeschreibung) muss zur Übersetzungszeit noch nicht bekannt sein. Sie kann zur Laufzeit dynamisch aufgebaut werden und wird in einer Benutzervariable bereitgestellt.

Eine Routine (siehe Kapitel „Routinen“) darf keine dynamisch formulierten SQL-Anweisungen oder Cursorbeschreibungen enthalten.

Platzhalter

In einer dynamisch formulierten SQL-Anweisung (bzw. Cursorbeschreibung) dürfen Sie keine Benutzervariablen verwenden. Stattdessen können Sie für noch unbekannte Eingabewerte Platzhalter in Form von Fragezeichen angeben. Die Regeln für Platzhalter sind bei „PREPARE - Dynamisch formulierte Anweisungen vorbereiten“ beschrieben.

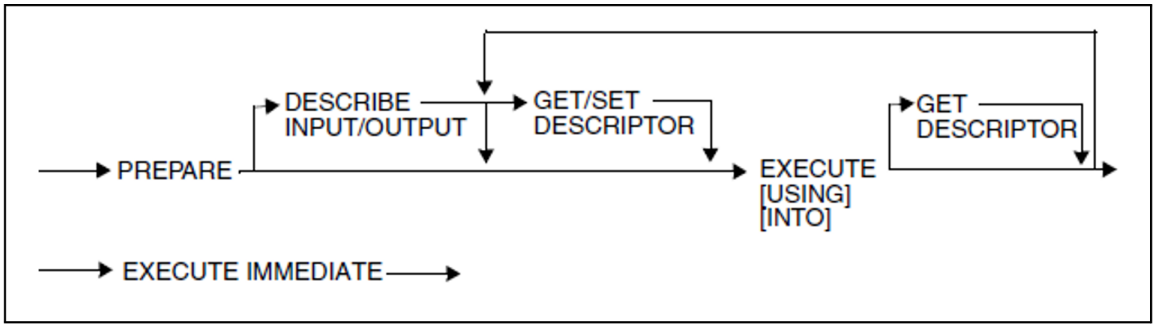
2.5.1 Dynamisch formulierte Anweisung

Eine dynamisch formulierte Anweisung kann entweder einmal direkt ausgeführt werden, oder sie wird vorbereitet. Eine vorbereitete Anweisung kann beliebig oft ausgeführt werden.

Bei einer direkt ausgeführten Anweisung können keine Platzhalter angegeben werden und sie darf keine Ergebniswerte liefern.

Eine vorbereitete Anweisung bleibt mindestens für die Dauer der aktuellen Transaktion für die Ausführung vorbereitet.

Die SQL-Anweisungen für dynamisch formulierte Anweisungen zeigt folgende Zusammenfassung:



Ein Deskriptor muss vor der Verwendung in DESCRIBE und GET bzw. SET DESCRIPTOR mit ALLOCATE DESCRIPTOR angelegt werden (siehe Abschnitt „Deskriptorbereich“).

2.5.1.1 Dynamisch formulierte Anweisung vorbereiten

Mit PREPARE bereiten Sie eine dynamisch formulierte Anweisung vor. Sie definieren einen Namen, genannt Anweisungsbezeichner, mit dem die dynamisch formulierte Anweisung in nachfolgenden Anweisungen, insbesondere beim Ausführen mit EXECUTE, angesprochen werden kann. Bei der Beschreibung „Anweisungen für PREPARE“ sind alle SQL-Anweisungen aufgelistet, die mit PREPARE vorbereitet werden können.

Für die noch nicht bekannten SQL-Anweisungen, für die der Anweisungsbezeichner steht, geben Sie eine alphanumerische Benutzervariable an. Die Länge der Benutzervariable darf bis zu 32000 Zeichen betragen. Die Angabe einer Indikatorvariablen ist nicht erlaubt.

Der Benutzervariable weisen Sie dann im Programm die gewünschte SQL-Anweisung als alphanumerische Zeichenkette zu. Sie können die SQL-Anweisung zum Beispiel über ein Dialogprogramm einlesen und daraus die Zeichenkette zusammenbauen, die in der Benutzervariable übergeben wird.

Wenn die PREPARE-Anweisung ausgeführt wird, muss die dynamisch formulierte Anweisung bis auf die Werte für Platzhalter bekannt sein. Ist die Anweisung nicht korrekt, wird die PREPARE-Anweisung mit Fehler abgebrochen.

2.5.1.2 Datentypen der Platzhalter und Ergebniswerte abfragen

Wenn eine dynamisch formulierte Anweisung Platzhalter enthält, können Sie nach dem Vorbereiten mit PREPARE die Anzahl und die SQL-Datentypen der Platzhalter mit DESCRIBE INPUT abfragen. Sie müssen dazu einen Deskriptorbereich angeben, der die Beschreibung der SQL-Datentypen der Platzhalter aufnimmt.

Die Anzahl und die Datentypen der Ergebniswerte der vorbereiteten Anweisung können Sie mit DESCRIBE OUTPUT abfragen und in einem zuvor angeforderten Deskriptorbereich ablegen. Die Anzahl ist gleich 0, wenn die vorbereitete Anweisung keine SELECT-Anweisung oder Cursorbeschreibung ist.

Die Einträge im Deskriptorbereich können Sie dann mit GET DESCRIPTOR lesen (siehe Abschnitt „Deskriptorbereich“).

2.5.1.3 Dynamisch formulierte Anweisung ausführen

Es gibt die Möglichkeit, eine dynamisch formulierte Anweisung direkt mit EXECUTE IMMEDIATE vorzubereiten und auszuführen. Die Anweisung darf dann jedoch keine Platzhalter enthalten und keine Ergebniswerte liefern. In der Beschreibung der EXECUTE IMMEDIATE-Anweisung, "EXECUTE IMMEDIATE - Dynamisch formulierte Anweisung ausführen", sind alle SQL-Anweisungen aufgelistet, die mit EXECUTE IMMEDIATE ausgeführt werden können.

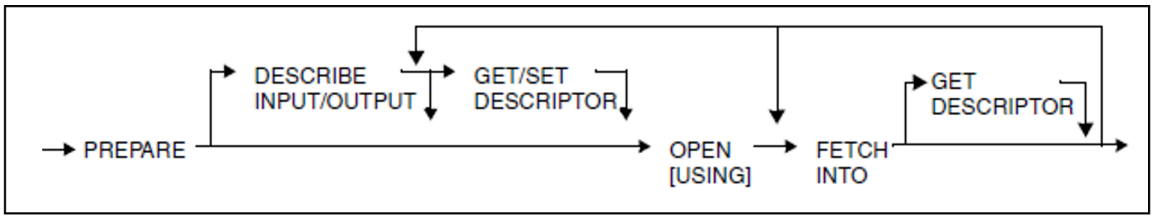
Eine mit PREPARE vorbereitete Anweisung wird mit EXECUTE ausgeführt. Enthält die Anweisung Platzhalter, können die zugehörigen Werte über Benutzervariablen oder einen zuvor belegten Deskriptorbereich in der USING-Klausel der EXECUTE-Anweisung zur Verfügung gestellt werden.

Bei einer dynamisch formulierten SELECT-Anweisung können die Ergebnisse mit der INTO-Klausel in Benutzervariablen oder in einem zuvor angelegten Deskriptorbereich abgelegt werden.

2.5.2 Dynamisch formulierte Cursorbeschreibung

Einem Cursor kann in der DECLARE CURSOR-Anweisung auch eine dynamisch formulierte Cursorbeschreibung zugeordnet werden. Der Cursor heißt dann dynamischer Cursor. Ein nicht-dynamischer Cursor heißt auch statischer Cursor. Eine dynamisch formulierte Cursorbeschreibung wird mit PREPARE vorbereitet.

Die SQL-Anweisungen für eine dynamisch formulierte Cursorbeschreibung zeigt folgende Zusammenfassung:



Die weiteren SQL-Anweisungen für Cursor sind in den Abschnitten „Cursor zum Lesen“ und „Änderbarer Cursor“ zusammengestellt.

2.5.2.1 Dynamisch formulierte Cursorbeschreibung vorbereiten

Mit PREPARE bereiten Sie eine dynamisch formulierte Cursorbeschreibung vor. Sie definieren einen Namen, genannt Anweisungsbezeichner, für die Cursorbeschreibung. Jedem Cursor, der mit diesem Anweisungsbezeichner vereinbart ist, wird die zugehörige Cursorbeschreibung zugeordnet.

Für den noch nicht bekannten Abfrageausdruck geben Sie eine alphanumerische Benutzervariable an. Die Länge der Benutzervariable darf bis zu 32000 Zeichen betragen. Die Angabe einer Indikatorvariablen ist nicht erlaubt.

Dieser Benutzervariable weisen Sie dann beim Programmlauf den gewünschten Abfrageausdruck als alphanumerische Zeichenkette zu.

Wenn die PREPARE-Anweisung ausgeführt wird, muss der Abfrageausdruck bis auf die Werte für Platzhalter bekannt sein. Ist der Abfrageausdruck nicht korrekt, wird die PREPARE-Anweisung mit Fehler abgebrochen.

2.5.2.2 SQL-Datentypen der Platzhalter bestimmen

Wenn eine dynamisch formulierte Cursorbeschreibung Platzhalter enthält, können Sie nach dem Vorbereiten mit PREPARE die Anzahl und die SQL-Datentypen der Platzhalter mit DESCRIBE INPUT abfragen.

Sie müssen dazu einen Deskriptorbereich angeben, der die Beschreibung der Datentypen der Platzhalter aufnimmt. Die Einträge im Deskriptorbereich können Sie dann mit GET DESCRIPTOR lesen (siehe Abschnitt „Deskriptorbereich“).

2.5.2.3 SQL-Datentypen der Ergebnisspalten bestimmen

Bei einer dynamisch formulierten Cursorbeschreibung können Sie die Anzahl und die SQL-Datentypen der Ergebnisspalten mit DESCRIBE OUTPUT abfragen und in einem zuvor angelegten Deskriptorbereich ablegen.

2.5.2.4 Dynamisch formulierte Cursorbeschreibung auswerten

Eine dynamisch formulierte Cursorbeschreibung wird beim Öffnen des Cursors mit OPEN ausgewertet.

Enthält eine dynamisch formulierte Cursorbeschreibung Platzhalter, werden die zugehörigen Werte über Benutzervariablen oder einen zuvor belegten Deskriptorbereich in der USING-Klausel der OPEN-Anweisung zur Verfügung gestellt. Ansonsten gelten für die Auswertung dieselben Regeln wie beim statischen Cursor.

2.5.2.5 Ergebnisse ablegen

Die Sätze der Cursortabelle werden wie bei einem statischen Cursor mit FETCH gelesen. Im Unterschied zum statischen Cursor können die gelesenen Spaltenwerte eines Satzes nicht nur in Benutzervariablen abgelegt werden sondern auch in einem zuvor angeforderten Deskriptorbereich.

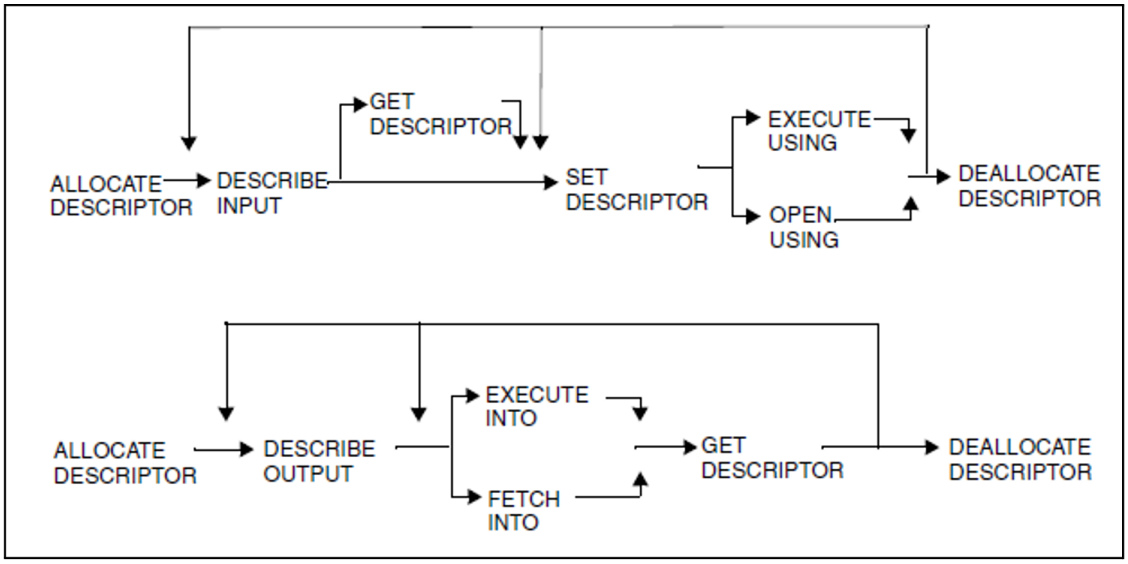
2.5.3 Deskriptorbereich

Ein Deskriptorbereich ist ein Speicherbereich, der für eine dynamisch formulierte Anweisung oder Cursorbeschreibung verwendet werden kann, um Werte und Informationen über SQL-Datentypen abzulegen.

Ein Deskriptorbereich kann in folgenden Fällen verwendet werden:

- Die SQL-Datentypen der Platzhalter einer vorbereiteten Anweisung oder Cursorbeschreibung können abgefragt und in einem Deskriptorbereich abgelegt werden (DESCRIBE INPUT).
- Die SQL-Datentypen der Ergebnisspalten einer vorbereiteten SELECT-Anweisung oder Cursorbeschreibung können abgefragt und in einem Deskriptorbereich abgelegt werden (DESCRIBE OUTPUT).
- Die Werte für die Platzhalter einer dynamisch formulierten Anweisung oder Cursorbeschreibung können beim Ausführen aus einem Deskriptorbereich übergeben werden (USING-Klausel bei EXECUTE bzw. OPEN).
- Die Ergebniswerte einer dynamisch formulierten Anweisung oder Cursorbeschreibung können in einem Deskriptorbereich abgelegt werden (INTO-Klausel bei EXECUTE bzw. FETCH).

Für die Verwendung eines Deskriptorbereichs gibt es eine Reihe von SQL-Anweisungen. Sie müssen in einer festgeschriebenen Reihenfolge aufgerufen werden. Die folgende Zusammenfassung zeigt diese Anweisungen sowie mögliche Aufrufreihenfolgen (GET bzw. SET DESCRIPTOR können eine Folge von GET bzw. SET DESCRIPTOR-Anweisungen sein).



2.5.3.1 Deskriptorbereich anlegen

Sie legen einen Deskriptorbereich mit ALLOCATE DESCRIPTOR an. Dabei geben Sie die maximale Anzahl der Einträge an, die dieser Deskriptorbereich aufnehmen kann.

Die Einträge sind nach ALLOCATE DESCRIPTOR noch undefiniert.

2.5.3.2 Aufbau eines Deskriptorbereichs

Ein Deskriptorbereich besteht aus dem Feld COUNT und einer Anzahl von Einträgen.

Jeder Eintrag im Deskriptorbereich besteht aus einer Anzahl von Feldern, die einen SQL-Datentyp beschreiben und einen Wert dieses Typs enthalten können.

Für eine einfache Spalte bzw. einen einfachen Wert wird genau ein Eintrag verwendet. Bei einer multiplen Spalte bzw. einem Aggregat wird pro Spaltenelement bzw. Ausprägung ein Eintrag verwendet.

2.5.3.3 Deskriptorbereichsfelder

Zu den Deskriptorbereichsfeldern gehören das Feld COUNT, das pro Deskriptorbereich einmal vorhanden ist und die Felder der einzelnen Einträge.

Jeder Eintrag besteht aus folgenden Feldern:

- REPETITIONS
- TYPE
- DATETIME_INTERVAL_CODE
- PRECISION
- SCALE
- LENGTH
- INDICATOR
- DATA
- OCTET_LENGTH
- NULLABLE
- NAME
- UNNAMED

Im Folgenden finden Sie eine detaillierte Beschreibung der einzelnen Felder.

Feld COUNT

Das Deskriptorbereichsfeld COUNT enthält einen Wert für die Anzahl der verwendeten bzw. der benötigten Einträge.

Ist bei einer DESCRIBE-Anweisung die Anzahl der benötigten Einträge größer als die festgelegte Maximalanzahl von Einträgen, wird nur das COUNT-Feld auf die benötigte Anzahl gesetzt. Alle anderen Felder werden nicht belegt.

SQL-Datentyp: SMALLINT

Felder eines Eintrags

Nicht alle Felder sind bei jedem Eintrag belegt. Unbelegte Felder haben einen undefinierten Wert.

Die Felder sind im Folgenden in alphabetischer Reihenfolge beschrieben.

DATA

Nur definiert, wenn der Wert im Feld INDICATOR größer oder gleich 0 ist:

Wert des Eintrags.

SQL-Datentyp: bestimmt durch die Felder TYPE, LENGTH, PRECISION, SCALE und DATETIME_INTERVAL_CODE

DATETIME_INTERVAL_CODE

Nur bei Zeit-Datentypen:

Datentyp des Eintrags.

DATETIME_INTERVAL_CODE	SQL-Datentyp
1	DATE
2	TIME
3	TIMESTAMP

Tabelle 1: Deskriptorbereichsfeld DATETIME_INTERVAL_CODE

SQL-Datentyp: SMALLINT

INDICATOR

Angaben zum Wert des Eintrags:

< 0	Wert des Eintrags ist der NULL-Wert
> 0	Originallänge einer alphanumerischen Zeichenkette oder einer National-Zeichenkette, die bei der Übertragung aus der Datenbank verkürzt wurde
0	sonst

SQL-Datentyp: SMALLINT

LENGTH

Nur bei alphanumerischen Datentypen, National-Datentypen und Zeit-Datentypen:

Länge des SQL-Datentyps in Zeichen bzw. bei National-Datentypen in Code Units.

LENGTH	bei SQL-Datentyp
länge	CHAR(*länge*)
max	VARCHAR(*max*)
cu_länge	NCHAR(*cu_länge*)
cu_max	NVARCHAR(*cu_max*)
10	DATE
12	TIME(3)
23	TIMESTAMP(3)

Tabelle 2: Deskriptorbereichsfeld LENGTH

SQL-Datentyp: SMALLINT

NAME

Spaltenname, wenn sich der Eintrag auf eine Spalte bezieht, sonst intern verwendeter Spaltenname.

SQL-Datentyp: CHAR(*n*) oder VARCHAR(*n*) mit *n* >= 128

NULLABLE

Angabe, ob der Wert des Eintrags der NULL-Wert sein kann.

1 Wert kann der NULL-Wert sein

0 sonst

SQL-Datentyp: SMALLINT

OCTET_LENGTH

Maximaler Speicherplatzbedarf des durch die Felder TYPE, LENGTH, PRECISION, SCALE und DATETIME_INTERVAL_CODE bestimmten Datentyps in Bytes. Wenn diese Felder keinen korrekten SQL-Datentyp angeben, ist der Wert von OCTET_LENGTH undefiniert.

Der Wert von OCTET_LENGTH ist für numerische Datentypen und Zeit-Datentypen implementationsabhängig und kann sich in Folgeversionen von SESAM/SQL ändern.

OCTET_LENGTH	**bei SQL-Datentyp**
länge	CHAR(*länge*)
max+2	VARCHAR(*max*)
*2*cu_länge*	NCHAR(*cu_länge*)
*2*cu_max*+2	NVARCHAR(*cu_max*)
stellen+1	NUMERIC(*stellen*,*bruchteil*)
stellen/2+1, wenn *stellen* gerade (*stellen*-1)/2+1, sonst	DECIMAL(*stellen*,*bruchteil*)
4	INTEGER
2	SMALLINT
4, wenn *stellen*<22 8, sonst	FLOAT(*stellen*)
4	REAL
8	DOUBLE PRECISION
6	DATE
8	TIME(3)
14	TIMESTAMP(3)

Tabelle 3: Deskriptorbereichsfeld OCTET_LENGTH

SQL-Datentyp: SMALLINT

PRECISION

Nur bei numerischen Datentypen sowie TIME und TIMESTAMP:

Anzahl der Dezimal- oder Binärstellen des SQL-Datentyps.

PRECISION	**bei SQL-Datentyp**
stellen	NUMERIC(*stellen*,*bruchteil*)
stellen	DECIMAL(*stellen*,*bruchteil*)
31	INTEGER
15	SMALLINT
stellen	FLOAT(*stellen*)
21	REAL
53	DOUBLE PRECISION
3	TIME(3)
3	TIMESTAMP(3)

Tabelle 4: Deskriptorbereichsfeld PRECISION

SQL-Datentyp: SMALLINT

REPETITIONS

Dimension einer multiplen Spalte bzw. eines Aggregats.

Für jede Ausprägung einer multiplen Spalte bzw. eines Aggregats wird ein eigener Eintrag im Deskriptorbereich verwendet. Der erste Eintrag enthält im Feld REPETITIONS die Anzahl der Ausprägungen bzw. Spaltenelemente. Bei allen folgenden Einträgen ist REPETITIONS auf 1 gesetzt.

Bei einfachen Werten ist REPETITIONS auf 1 gesetzt.

SQL-Datentyp: SMALLINT

SCALE

Nur bei Ganzzahl- und Festpunktzahldatentypen:

Anzahl der Nachkommastellen des SQL-Datentyps.

SCALE	**bei SQL-Datentyp**
bruchteil	NUMERIC(*stellen*,*bruchteil*)
bruchteil	DECIMAL(*stellen*,*bruchteil*)

0	INTEGER
0	SMALLINT

Tabelle 5: Deskriptorbereichsfeld SCALE

SQL-Datentyp: SMALLINT

TYPE

SQL-Datentyp des Eintrags:

TYPE	SQL-Datentyp
-42	NVARCHAR
-31	NCHAR
1	CHAR
2	NUMERIC
3	DECIMAL
4	INTEGER
5	SMALLINT
6	FLOAT
7	REAL
8	DOUBLE PRECISION
9	DATE, TIME oder TIMESTAMP
12	VARCHAR

Tabelle 6: Deskriptorbereichsfeld TYPE

SQL-Datentyp: SMALLINT

UNNAMED

Angabe, ob das Feld NAME einen gültigen Spaltennamen enthält.

0 NAME enthält einen Spaltennamen

1 sonst

SQL-Datentyp: SMALLINT

2.5.3.4 Deskriptorbereich belegen

Nachdem Sie einen Deskriptorbereich angelegt haben, können Sie diesen Bereich auf verschiedene Arten belegen, mit:

- Datentypbeschreibungen:
  mit DESCRIBE belegen Sie den Deskriptorbereich mit der Beschreibung der SQL-Datentypen der Platzhalter bzw. Ergebniswerten einer vorbereiteten Anweisung oder Cursorbeschreibung.
- Werten:
  mit EXECUTE ... INTO oder FETCH ... INTO belegen Sie den Deskriptorbereich mit den Ergebniswerten einer Datenabfrage.
- Datentypbeschreibungen und Werten:
  Mit SET DESCRIPTOR setzen Sie Einträge im Deskriptorbereich. Wie die Felder eines Eintrags belegt werden können, ist im Abschnitt „SET DESCRIPTOR - SQL-Deskriptorbereich ändern“ beschrieben.

Die Felder NAME, UNNAMED und NULLABLE werden nur bei DESCRIBE gesetzt.

Die Felder TYPE, DATETIME_INTERVAL_CODE, LENGTH, PRECISION, SCALE, REPE-TITIONS können mit SET DESCRIPTOR und DESCRIBE gesetzt werden.

Die Felder INDICATOR und DATA können mit SET DESCRIPTOR gesetzt werden sowie mit EXECUTE INTO und FETCH INTO, wenn ein SQL-Deskriptorbereich verwendet wird.

Bei der Übertragung eines Werts von einer Benutzervariablen in ein Deskriptorbereichsfeld muss der SQL-Datentyp der Benutzervariable die Bedingungen erfüllen, die bei SET DESCRIPTOR, "SET DESCRIPTOR - SQL-Deskriptorbereich ändern", und im Abschnitt „Werte zwischen Benutzervariablen und Deskriptorbereich übertragen“ beschrieben sind.

2.5.3.5 Deskriptorbereich abfragen

Den Wert des COUNT-Felds sowie die Felder einzelner Deskriptorbereichseinträge können Sie mit GET DESCRIPTOR abfragen.

Zur Abfrage eines Eintrags geben Sie die Nummer des Eintrags an und die Felder, deren Werte Sie abfragen möchten. Die Felder eines Eintrags sind im Abschnitt „Deskriptorbereichsfelder“ beschrieben.

Bei der Übertragung eines Werts von einem Deskriptorbereichsfeld in eine Benutzervariable muss der SQL-Datentyp der Benutzervariable die Bedingungen erfüllen, die bei GET DESCRIPTOR, "GET DESCRIPTOR - SQL-Deskriptorbereich lesen", und im Abschnitt „Werte zwischen Benutzervariablen und Deskriptorbereich übertragen“ beschrieben sind.

2.5.3.6 Werte aus dem Deskriptorbereich verwenden

Die Felder TYPE, DATETIME_INTERVAL_CODE, LENGTH, PRECISION, SCALE, REPETITIONS werden bei EXECUTE, OPEN und FETCH gelesen, wenn ein SQL-Deskriptorbereich für die Eingabe- bzw. Ausgabewerte verwendet wird.

Die Felder INDICATOR und DATA werden bei EXECUTE USING und OPEN USING gelesen, wenn ein SQL-Deskriptorbereich für die Eingabewerte verwendet wird.

2.5.3.7 Deskriptorbereich freigeben

Wenn Sie einen Deskriptorbereich nicht mehr benötigen, geben Sie den Speicherplatz, den ein Deskriptorbereich belegt, mit DEALLOCATE DESCRIPTOR wieder frei.

2.6 SQL-Anweisungen in CALL-DML-Transaktionen

SESAM/SQL unterstützt die SQL- und die CALL-DML-Schnittstelle.
In einer ESQL-COBOL-Anwendung können im Mischbetrieb beide Schnittstellen zusammen verwendet werden (siehe Handbuch „ CALL-DML Anwendungen“).
SQL- und CALL-DML-Schnittstelle können auch innerhalb einer Transaktion gemischt werden: Um bei bestehenden CALL-DML-Anwendungen den schrittweisen Einstieg in die SQL-Welt zu erleichtern, ist es möglich, SQL-Anweisungen innerhalb von CALL-DML-Transaktionen abzusetzen.

CALL-DML-Transaktion

Eine CALL-DML-Transaktion beginnt mit der CALL-DML-Anweisung BTA und endet mit dem Vor- oder Rücksetzen der Transaktion.
Vorgesetzt wird eine CALL-DML-Transaktion durch die CALL-DML-Anweisung ETA. Zurückgesetzt wird sie durch die Anweisung RTA oder intern durch den SESAM/SQL-DBH, etwa bei Deadlock-Auflösung.
Unter openUTM wird eine Transaktion durch eine transaktionsbeendende PEND-Variante vorgesetzt, zurückgesetzt wird sie durch Rücksetzen der UTM-Transaktion.

Erlaubte SQL-Anweisungen in einer CALL-DML-Transaktion

Innerhalb einer CALL-DML-Transaktion können Sie alle SQL-Anweisungen zum Abfragen und Ändern von Daten, SQL-Anweisungen der dynamischen SQL, einige SQL-Anweisungen zur Sessionsteuerung, die CALL-Anweisung und die WHENEVER-Anweisung ausführen (zur Einteilung der SQL-Anweisungen siehe Abschnitt „Inhaltliche Zusammenstellung“).

Innerhalb einer CALL-DML-Transaktion sind folgende SQL-Anweisungen nicht erlaubt:

- COMMIT WORK
- ROLLBACK WORK

Nicht erlaubt sind außerdem alle Anweisungen, die auch in einer SQL-DML-Transaktion nicht verwendet werden dürfen:

- SET TRANSACTION
- SET SESSION AUTHORIZATION
- SQL-Anweisungen zur Schemadefinition und -verwaltung
- SQL-Anweisungen zur Verwaltung der Speicherstruktur
- SQL-Anweisungen zur Verwaltung von Benutzereinträgen
- Utility-Anweisungen

> ¡ Wird die Anweisung SET TRANSACTION vor Beginn einer CALL-DML-Transaktion eingegeben, so gelten die Einstellungen nur für gegebenenfalls vorhandene SQL-Anweisungen innerhalb der folgenden (CALL-DML-)Transaktion. Nach Ende der Transaktion gelten wieder die Voreinstellungen.

2.6.1 CALL-DML-Anwendungen schrittweise umstellen

Um bestehende CALL-DML-Anwendungen auf die SQL-Schnittstelle umzustellen, ist es ratsam, die einzelnen Schritte in bestimmter Reihenfolge auszuführen. Abhängig von der Art der Anwendung bzw. der Anweisung, werden die wichtigsten Schritte im Folgenden kurz zusammengefasst.

TIAM-Anwendung

Wollen Sie eine CALL-DML-Transaktion in einer TIAM-Anwendung auf die SQL-Schnittstelle umstellen, gehen Sie folgendermaßen vor:

1. Alle CALL-DML-Anweisungen ungleich BTA, ETA und RTA schrittweise durch SQL-Anweisungen ersetzen.
2. Erst dann die Anweisungen BTA, ETA und RTA ersetzen:
   - BTA ersatzlos streichen
   - ETA durch COMMIT WORK ersetzen
   - RTA durch ROLLBACK WORK ersetzen.

openUTM-Anwendung

Wollen Sie eine CALL-DML-Transaktion in einer openUTM-Anwendung auf die SQL-Schnittstelle umstellen, gehen Sie folgendermaßen vor:

1. Alle CALL-DML-Anweisungen ungleich BTA, ETA und RTA schrittweise durch SQL-Anweisungen ersetzen.
2. Erst dann die Anweisungen BTA, ETA und RTA ersetzen:
   - BTA ersatzlos streichen
   - ETA ersatzlos streichen
   - RTA durch RSET ersetzen
     (RSET ist eine Funktion an der KDCS-Schnittstelle von openUTM).

CALL-DML-Anweisungen außerhalb einer CALL-DML-Transaktion

Um CALL-DML-Anweisungen, die außerhalb von CALL-DML-Transaktionen eingegeben werden, auf die SQL-Schnittstelle umzustellen, müssen Sie diese durch entsprechende SQL-Anweisungen ersetzen. Einschränkungen bezüglich erlaubter SQL-Anweisungen gibt es in diesem Fall nicht. Beachten Sie, dass die meisten SQL-Anweisungen implizit eine Transaktion öffnen. Diese muss geschlossen werden, bevor die nächste CALL-DML-Anweisung folgt.

2.6.2 User-Close berücksichtigen und Betriebsmittel freigeben

Der User-Close einer CALL-DML-Anwendung schließt alle logischen Dateien eines Auftraggebers. Bei erfolgreicher Ausführung des User-Close werden alle Betriebsmittel der logischen Dateien dieses Auftraggebers freigegeben. Innerhalb von SQL-Anwendungen gibt es keine Möglichkeit, einen SQL-Vorgang explizit zu beenden. Die Betriebsmittel für einen SQL-Vorgang werden erst freigegeben, wenn die zugehörige TIAM-Anwendung beendet wird. Unter openUTM werden die Betriebsmittel für einen SQL-Vorgang beim Ende des entsprechenden UTM-Vorgangs freigegeben.

Für den User-Close einer CALL-DML-Anwendung gibt es keine entsprechende SQL-Anweisung. Enthält eine CALL-DML-Anwendung zahlreiche User-Close-Anweisungen, sollten Sie daher die DBH-Option USERS erhöhen, bevor Sie auf die SQL-Schnittstelle umstellen. So können Sie Betriebsmittel-Engpässe vermeiden.

2.6.3 Isolationslevel einstellen

Das Sperrkonzept, das die Konsistenz der bearbeiteten Daten gewährleistet, wird bei CALL-DML-Anwendungen folgendermaßen realisiert: Greift eine Wiedergewinnungsanweisung auf die Anwenderdaten einer CALL-DML-Tabelle zu, sperrt der SESAM/SQL-DBH den jeweiligen Satz solange gegen den Zugriff anderer Transaktionen, bis die ausführende Transaktion beendet oder zurückgesetzt ist. Abhängig vom Open-Modus wird eine shared bzw. exclusive Sperre gesetzt. Daneben erlaubt SESAM/SQL für einzelne CALL-DML-Anweisungen folgende Modifikationen des Sperrkonzepts:

- Lesen ohne zu sperren (Read No Lock)
- Ignorieren der Sperre (Read No Wait)
- Lesen ohne zu sperren und Ignorieren der Sperre

Bei der Umstellung einer CALL-DML-Transaktion ist es ratsam, das Sperrverhalten möglichst nicht zu verändern. Liegt für die CALL-DML-Transaktion eine shared oder exclusive Sperre vor, sollten Sie vor Beginn der Transaktion mit der SQL-Anweisung SET TRANSACTION den Isolationslevel REPEATABLE READ einstellen.

Wurde für einzelne CALL-DML-Anweisungen das Sperrverhalten modifiziert, empfiehlt es sich, das Pragma ISOLATION LEVEL zu verwenden. Damit können Sie für die korrespondierende SQL-Anweisung gezielt einen Isolationslevel festlegen, der dem Sperrverhalten der jeweiligen CALL-DML-Anweisung entspricht:

- „Lesen ohne zu sperren“ ersetzen Sie durch READ COMMITED
- „Lesen ohne zu sperren und Ignorieren der Sperre“ ersetzen Sie durch READ UNCOMMITED

Nur für das Sperrverhalten „Ignorieren der Sperre“ kennt SESAM/SQL keinen entsprechenden Isolationslevel. Hier ist fallweise abzuwägen, ob der Isolationslevel READ COMMITED oder READ UNCOMMITED geeigneter ist.

3 Lexikalische Elemente und Namen

Dieses Kapitel behandelt folgende Themen:

- SESAM/SQL-Zeichenvorrat
- Lexikalische Einheiten
- Pragmas und Annotationen
- Namen

3.1 SESAM/SQL-Zeichenvorrat

Der SESAM/SQL-Zeichenvorrat setzt sich zusammen aus Buchstaben, Ziffern und Sonderzeichen.

Buchstaben sind Großbuchstaben von A-Z und Kleinbuchstaben a-z (ohne Umlaute und ß).

Ziffern sind die Zeichen 0-9.

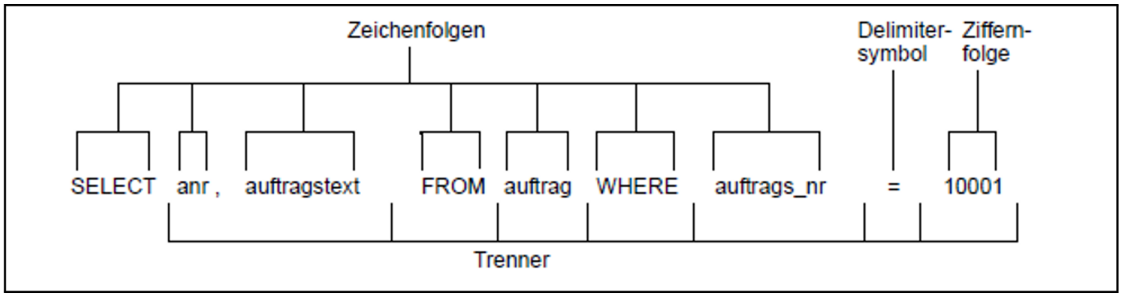
Zu den Sonderzeichen gehören:

" ' : ; , . - & | ( ) = + * / < > ? % _ [ ] (Leerzeichen)

3.2 Lexikalische Einheiten

Die aus dem SQL-Zeichenvorrat gebildeten Wortfolgen werden in lexikalische Einheiten eingeteilt. Eine SQL-Anweisung besteht aus den lexikalischen Einheiten:

- Zeichenfolge
- Ziffernfolge
- Delimiter-Symbol
- Trenner
- Kommentar



3.2.1 Zeichenfolgen

Zu den Zeichenfolgen gehören die SQL-Schlüsselwörter und Namen sowie alphanumerische Literale, National-Literale und Zeitliterale.

Zeichenfolgen für SQL-Schlüsselwörter

Ein SQL-Schlüsselwort ist eine Folge aus Buchstaben, die groß- oder kleingeschrieben werden können. Ein SQL-Schlüsselwort wird nicht in Anführungszeichen oder Hochkomma eingeschlossen. Eine Auflistung aller Schlüsselwörter finden Sie im Abschnitt „SQL-Schlüsselwörter“.

Beispiel: SELECT

In diesem Handbuch sind alle SQL-Schlüsselwörter zur Unterscheidung großgeschrieben.

Zeichenfolgen für Namen

Die Syntax für Namen ist im Abschnitt „Namen“ beschrieben.

Zeichenfolgen für Literale

Zeichenfolgen für alphanumerische Literale, National-Literale und Zeitliterale werden in Hochkommata gesetzt (siehe Abschnitt „Alphanumerische Literale“, Abschnitt „National-Literale“ und Abschnitt „Zeitliterale“).

Beispiel: 'Maier'

3.2.2 Ziffernfolgen

Eine Ziffernfolge ist eine Folge aus den Ziffern 0-9. Numerische Literale werden aus Ziffernfolgen und den Zeichen + - . E gebildet.

Beispiel: 314

Die Syntax für numerische Literale ist im Abschnitt „Numerische Literale“ beschrieben.

3.2.3 Delimiter-Symbole

Zu den Delimiter-Symbolen gehören die Operatoren und folgende Sonderzeichen:

: ; , . ( ) [ ] ?

Operatoren

Operatoren werden zum Bilden von Ausdrücken und Prädikaten verwendet. In der folgenden Tabelle sind die bei SESAM/SQL definierten Operatoren zusammengestellt:

Operator	Bedeutung
*	Multiplikation
/	Division
+	Addition
-	Vorzeichen, Negation
=	Gleich
<>	Ungleich
>	Größer
<	Kleiner
>=	Größer oder gleich
<=	Kleiner oder gleich
||	Konkatenation

Tabelle 7: Operatoren

Die Bedeutung der Operatoren ist im Kapitel „Zusammengesetzte Sprachelemente“ ausführlich beschrieben.

3.2.4 Trenner

Trenner dienen zur Trennung von lexikalischen Einheiten. Trenner sind Leerzeichen, Übergang zur nächsten Zeile und Kommentare.

3.2.5 Kommentar

SQL bietet die Möglichkeit, zur Dokumentation innerhalb von SQL-Anweisungen Kommentare aufzunehmen. Kommentare beginnen mit der Zeichenfolge -- und enden mit dem Ende der Zeile. Zusätzlich gibt es geklammerte Kommentare, die mit `/*` beginnen und mit `*/` enden, und die auch geschachtelt werden können.

Zu den Kommentaren gehören auch Pragmas und Annotationen, siehe Abschnitt „Pragmas und Annotationen“).

3.3 Pragmas und Annotationen

Pragmas und Annotationen sind spezielle SQL-Kommentare, die von SESAM/SQL interpretiert werden. Mit ihnen können Sie Hinweise für die Ausführung von SQL- oder UtilityAnweisungen geben. Pragmas oder Annotationen mit syntaktischen Fehlern werden als Kommentar behandelt und von SESAM/SQL ignoriert.

Ein **Pragma** kann nur am Anfang einer SQL- oder Utility-Anweisung stehen. Davor dürfen nur Kommentare (einschliesslich weiterer Pragmas) und Trenner stehen. Pragmas wirken sich auf die gesamte Anweisung aus, einschliesslich der verwendeten Views. Das Pragma PREFETCH wirkt sich sogar auf alle Operationen mit einem Cursor aus.

Eine **Annotation** kann nur an gewissen Positionen im Text einer Anweisung stehen. Sie wirkt sich, abhängig von ihrer Position, nur auf eine bestimmte Operation in der Anweisung aus. An jeder dieser Positionen kann immer nur eine Annotation stehen. Eine Anweisung kann aber mehrere Annotationen enthalten, ebenso wie die verwendeten Views.

Pragmas und Annotationen wirken nur bei bestimmten Anweisungen, sonst werden sie ignoriert. Zur Verwendung von Pragmas in Routinen siehe Abschnitt „Pragmas in Routinen“ .

Pragmas und Annotationen dienen unterschiedlichen Zwecken. Sie sind in unterschiedlichen Handbüchern von SESAM/SQL beschrieben, siehe die Tabellen auf den folgenden Seiten.

Format

pragma `::= --%PRAGMA` *pragma_text* `,...` *zeilenende*

annotation `::= /*%` *annotation_text* `%*/`

pragma_text

Eine Folge von Schlüsselwörtern, Literalen und Namen.
Die Folge darf Leerzeichen enthalten, aber keine anderen Trenner.

Die Formate für *pragma_text* und ihre Wirkung sind an folgenden Stellen beschrieben:

***pragma_text* beginnt mit**	**Bedeutung**	**Beschreibung siehe**
AUTONOMOUS TRANSACTION	Daten unabhängig von der umgebenden Transaktion schreiben	"Pragma AUTONOMOUS TRANSACTION"
CHECK	Integritätsbedingungen beachten	Handbuch „ SQL-Sprachbeschreibung Teil 2: Utilities“
DATA TYPE	alte CALL-DML-Typen verwenden	"Pragma DATA TYPE"
DEBUG ROUTINE	Fehlerinformationen für Routinen erhalten	"Pragma DEBUG ROUTINE"

DEBUG VALUE	Informationen für Zuweisungen in Routinen erhalten	"Pragma DEBUG VALUE"
EXPLAIN	Zugriffsplan ausgeben	"Pragma EXPLAIN"
IGNORE	Index ignorieren	Handbuch „ Performance“
ISOLATION LEVEL	Isolationslevel festlegen	"Pragma ISOLATION LEVEL"
JOIN	Join-Methode wählen	Handbuch „ Performance“
KEEP JOIN ORDER	Join-Reihenfolge beibehalten	Handbuch „ Performance“
LIMIT ABORT_EXECUTION	Betriebsmittelverbrauch limitieren	"Pragma LIMIT ABORT_EXECUTION"
LOCK MODE	Sperrmodus einstellen	"Pragma LOCK MODE"
LOOP LIMIT	Anzahl Schleifendurchläufe begrenzen	"Pragma LOOP LIMIT"
OPTIMIZATION	Zugriffsplanung einschränken	Handbuch „ Performance“
PREFETCH	Schubmodus steuern	"Pragma PREFETCH"
SIMPLIFICATION	Optimierungstechniken steuern	Handbuch „ Performance“
USE	Index nutzen	Handbuch „ Performance“
UTILITY MODE	Transaktionssicherung steuern	"Pragma UTILITY MODE"

Tabelle 8: Pragmas

Wenn Sie in einer Anweisung mehrere Pragmas angeben, die mit dem gleichen Schlüsselwort beginnen, dann wird nur die letzte Angabe verwendet. Die Pragmas IGNORE und USE werden aber alle, unabhängig von ihrer Reihenfolge, nach speziellen Regeln interpretiert.

zeilenende

Zeilenwechsel im SQL-Quelltext.
Wenn der SQL-Text in einer Anweisung PREPARE oder EXECUTE IMMEDIATE als Zeichenkette angegeben wird, dann stellt darin das alphanumerische Zeichen X'15' einen Zeilenwechsel dar.

annotation_text

Eine Folge von Schlüsselwörtern.
Die Folge darf Leerzeichen und Zeilenwechsel enthalten, aber keine Kommentare.

Eine Annotation muss auf ein Schlüsselwort folgen. Dazwischen dürfen nur Leerzeichen und Zeilenwechsel stehen, aber keine Kommentare. Das vorangehende Schlüsselwort bestimmt das erlaubte Format von *annotation_text* und die Wirkung der Annotation. Eine Annotation, die nicht diesen Regeln folgt, wird als Kommentar betrachtet und ignoriert.

Die Formate für *annotation_text* und ihre Wirkung sind an folgenden Stellen beschrieben::

Annotation hinter Schlüsselwort	**Bedeutung**	**Beschreibung siehe**
JOIN	Join-Algorithmus wählen	Handbuch „ Performance“
CACHE	CSV-Datei in temporärer Datei zwischenspeichern	Handbuch „ Performance“
VOLATILE	Funktionswert stets neu berechnen	"Unkorrelierte Funktionsaufrufe"
IMMUTABLE	Funktionswert in unkorrelierten Funktionsaufrufen nicht neu berechnen	"Unkorrelierte Funktionsaufrufe"

Tabelle 9: Annotationen

Wenn sich ein Pragma und eine Annotation auf eine Operation in einer Anweisung unterschiedlich auswirken würden (z.B. eine unterschiedliche Wahl des Join-Algorithmus), dann hat normalerweise die Annotation Vorrang. Die Einzelheiten sind in der Beschreibung der Annotation enthalten.

3.3.1 Pragma AUTONOMOUS TRANSACTION

Das Pragma AUTONOMOUS TRANSACTION ermöglicht es, Daten unabhängig vom Ausgang der umgebenden Transaktion in eine Datenbank zu schreiben.
Insbesondere werden die Daten persistent in die Datenbank geschrieben, bevor möglicherweise die SQL-Anweisung ROLLBACK WORK der Transaktion ausgeführt wird.

Das Pragma darf nur bei SQL-Anweisungen zum Ändern von Daten, also bei INSERT, UPDATE (Suchbedingung erfüllt), DELETE (Suchbedingung erfüllt), MERGE und CALL angegeben werden. Wird das Pragma bei Anweisungen zum Abfragen von Daten angegeben, dann wird die Anweisung mit SQLSTATE abgewiesen.

Das Pragma darf in Routinen nicht verwendet werden.

```
AUTONOMOUS TRANSACTION
```

Hinweise

- Die SQL-Anweisung hinter dem Pragma AUTONOMOUS TRANSACTION wird in der aktuellen Transaktion des Anwenders, aber in einer eigenen Ablaufumgebung (eigenen Thread, eigener Transaktionskontext) ausgeführt. Transaktionssteuernde Anweisungen des Anwenders haben keine Wirkung.

  Es wird die interne Benutzer-Identifikation (`APPLICATION-NAME=AUTTRAN`) verwendet, siehe Handbuch „Datenbankbetrieb“. Sie ist während des Ablaufs der autonomen Transaktion in Informationsausgaben sichtbar. Eine autonome Transaktion kann aber nicht administriert werden.
- Lock-Konflikte
  Der Transaktionskontext der autonomen Transaktion ist unabhängig von der umgebenden Transaktion der Anwendung und von anderen Transaktionen.
  Dies kann einerseits zu einem Deadlock zwischen der autonomen Transaktion und der umgebenden Transaktion führen. Diese Deadlock-Situation wird durch Rücksetzen der autonomen Transaktion aufgelöst. Der autonomen Transaktion wird der SQLSTATE 81SAT gemeldet.
  Dies kann andererseits zu einem Deadlock zwischen der autonomen Transaktion und anderen Transaktionen führen. Solche Deadlock-Situationen werden durch Rücksetzen der „billigsten“ Transaktion aufgelöst. Wenn die autonome Transaktion davon betroffen ist, dann wird ihr der SQLSTATE 81SAT gemeldet.
- Abbruch der Anwendung
  Wenn die Anwendung abbricht, die die autonome Transaktion ausgelöst hat, dann wird zuerst die autonome Transaktion abgebrochen und danach die aktuelle Transaktion bzw. die Anwendung.

3.3.2 Pragma DATA TYPE

Das Pragma DATA TYPE legt fest, dass eine Spalte im Attributformat für Nur-CALL-DML-Tabellen angelegt wird.

Das Pragma hat nur eine Wirkung, wenn es bei der Anweisung ALTER TABLE ... ADD COLUMN ... angegeben ist und die Tabelle eine Nur-CALL-DML-Tabelle ist.

```
DATA TYPE OLDEST
```

3.3.3 Pragma DEBUG ROUTINE

Das Pragma DEBUG ROUTINE stellt zusätzliche Informationen zu einem möglicherweise fehlerhaften Ablauf einer Routine zur Verfügung. Diese Informationen können über den View SYS_ROUTINE_ERRORS des SYS_INFO_SCHEMA gelesen werden, siehe "SYS_ROUTINE_ERRORS".

Das Pragma DEBUG ROUTINE wirkt nur außerhalb von Routinen. Es wirkt vor der SQL-Anweisung CALL und vor den DML-Anweisungen DECLARE CURSOR, DELETE, INSERT, MERGE, SELECT und UPDATE. Bei Angabe **vor** DML-Anweisungen wirkt das Pragma auf alle User Defined Functions (UDF) und die darin enthaltenen Routinen der DML-Anweisung.

> ¡ Das Pragma wurde in SESAM/SQL V9.0 umbenannt. Auch die Angabe von DEBUG PROCEDURE ist aus Kompatibilitätsgründen noch möglich.

`DEBUG ROUTINE [ALL | USER] [LEVEL` *vorzeichenlose_ganzzahl* `]`

---

vorzeichenlose_ganzzahl

Bei *vorzeichenlose_ganzzahl* > 0 werden für die durchgeführten SQL-Anweisungen der aktuellen Routine zusätzliche Informationen gesammelt.

vorzeichenlose_ganzzahl = 1 ist der Standardwert, wenn die Klausel LEVEL nicht angegeben wird.

Bei *vorzeichenlose_ganzzahl* = 0 wird das Pragma ignoriert.

Folgende Vorgehensweise bietet sich an:

Das Pragma ist zunächst zu Testzwecken mit einem Wert > 0 in einer Anwendung aktiv und wird dann später (ohne Änderung der Textlänge) durch den Wert 0 deaktiviert.

USER

Abhängig vom eingestellten LEVEL werden Informationen für die SQL-Anweisungen gesammelt, denen das Pragma DEBUG VALUE (siehe "Pragma DEBUG VALUE") vorangestellt ist.

ALL

Neben den bei USER erwähnten DEBUG-Informationen werden (unabhängig vom eingestellten LEVEL) auch allgemeine DEBUG-Informationen erstellt.
Beispielsweise wird jeder von einer fehlerhaften SQL-Anweisung gemeldete SQLSTATE oder SQLrowcount aufgezeichnet. Interne Aufrufe von Routinen werden ebenfalls aufgezeichnet. Auch die Position einer SQL-Anweisung innerhalb des Textes einer Routine wird normalerweise aufgezeichnet.

3.3.4 Pragma DEBUG VALUE

Das Pragma DEBUG VALUE stellt zusätzliche Informationen für folgende SQL-Anweisungen zur Verfügung:

- SET in Routinen (Prozeduren und User Defined Functions (UDF))
- RETURN in User Defined Functions (UDF)

Diese Informationen können über den View SYS_ROUTINE_ERRORS des SYS_INFO_SCHEMA gelesen werden, siehe "SYS_ROUTINE_ERRORS".

Das Pragma DEBUG VALUE ist derzeit nur vor diesen SQL-Anweisungen wirksam.

```
DEBUG VALUE [LEVEL vorzeichenlose_ganzzahl ]
```

vorzeichenlose_ganzzahl

Bei *vorzeichenlose_ganzzahl* > 0 werden für die o.g. Anweisungen zusätzliche Informationen dann gesammelt, wenn das Pragma DEBUG ROUTINE vor der SQL-Anweisung CALL bzw. vor einer DML-Anweisung (für darin enthaltene Routinen) steht. Zusätzlich muss *vorzeichenlose_ganzzahl* bei DEBUG ROUTINE größer oder gleich *vorzeichenlose_ganzzahl* bei DEBUG VALUE sein.

Folgende Informationen werden dann gesammelt:

- bei SET der zugewiesene Wert und der Name des Zielfeldes (Parameter oder lokale Variable)
- bei RETURN der zurückgegebene Wert

Bei Zeichenketten werden lange Werte ggf. am Ende abgeschnitten.

vorzeichenlose_ganzzahl = 1 ist der Standardwert, wenn die Klausel LEVEL nicht angegeben wird.

Bei *vorzeichenlose_ganzzahl* = 0 hat das Pragma keine Wirkung.

Folgende Vorgehensweise bietet sich an:

Das Pragma ist zunächst zu Testzwecken mit einem Wert > 0 in einer Anwendung aktiv und wird dann später (ohne Änderung der Textlänge) durch den Wert 0 deaktiviert.

> ¡ Das Pragma DEBUG VALUE kann nach Ende einer Test- bzw. Debugging-Phase im Text einer Routine verbleiben, solange die aufrufenden SQL-Anweisungen nicht das entsprechende Pragma DEBUG ROUTINE verwenden.

Beispiel

Den SET-Anweisungen einer Prozedur kann das Pragma DEBUG VALUE mit unterschiedlichen Werten für *vorzeichenlose_ganzzahl* vorangestellt werden. Durch Aufruf der Routine mit dem Pragma DEBUG ROUTINE und unterschiedlichen Werten für *vorzeichenlose_ganzzahl* werden Informationen in unterschiedlichem Umfang gesammelt.

```
CREATE PROCEDURE P (OUT par1 INTEGER,OUT par2 INTEGER)
       MODIFIES SQL DATA
       BEGIN
       --%PRAGMA DEBUG VALUE LEVEL 3
          SET par1 = 42;
       --%PRAGMA DEBUG VALUE LEVEL 10
          SET par2 = 43;
       END
```

Bei folgendem Prozeduraufruf wird nur die erste Zuweisung (`par1=42`) aufgezeichnet:

```
-- %PRAGMA DEBUG ROUTINE LEVEL 5
CALL P(mypar1, mypar2)
```

Bei folgendem Prozeduraufruf werden beide Zuweisungen aufgezeichnet:

```
-- %PRAGMA DEBUG ROUTINE LEVEL 20
CALL P(mypar1, mypar2)
```

Die Pragmas DEBUG VALUE können unverändert im Text der Routine verbleiben. Sie kommen erst bei entsprechender *vorzeichenlose_ganzzahl* im Pragma DEBUG ROUTINE zur Wirkung.

3.3.5 Pragma EXPLAIN

Das Pragma EXPLAIN dient dazu, den vom Optimizer gewählten Zugriffsplan auszugeben. Sie können dieses Pragma nur verwenden, wenn der aktuelle Berechtigungsschlüssel das Sonder-Privileg UTILITY besitzt.

Das Pragma hat nur in folgenden SQL-Anweisungen eine Wirkung:

- CALL
- Cursorbeschreibung (für dynamischen Cursor)
- DECLARE CURSOR (für statischen Cursor)
- DELETE
- INSERT
- MERGE
- SELECT
- UPDATE

In Routinen wird das Pragma ignoriert, siehe Abschnitt „Pragmas in Routinen“.

Bei einer nicht dynamisch formulierten Anweisung hat das Pragma nur dann eine Wirkung, wenn Sie das Programm mit Datenbankkontakt vorübersetzen.

`EXPLAIN INTO` *datei*

---

datei

Name der SAM-Datei, in die die Erklärung ausgegeben wird. Wenn die Datei bereits existiert, wird die Erklärung angehängt.

Wenn *datei* eine BS2000-Kennung angibt, dann wird diese Kennung verwendet. Sonst wird die Kennung des Data Base Handlers für die mit der SQL-Anweisung angesprochene Datenbank verwendet. In beiden Fällen muss der Data Base Handler Schreibrechte für die Datei besitzen. Für *datei* geben Sie ein alphanumerisches Literal an. Darin sollten keine Kleinbuchstaben vorkommen.

Bei dynamisch formulierten Anweisungen wird die Erklärung zum Ausführungzeitpunkt der PREPARE-Anweisung bzw. der EXECUTE IMMEDIATE-Anweisung ausgegeben. Bei nicht dynamisch formulierten Anweisungen wird die Erklärung zum Zeitpunkt der Vorübersetzung ausgegeben.

Die Erklärung besteht aus der SQL-Anweisung und einer aufbereiteten Darstellung des Zugriffsplans. Die Darstellung von Zugriffsplänen finden Sie im Handbuch „ Performance“.

Die Datei können Sie mit SHOW-FILE anzeigen. Um die Datei mit EDT lesen zu können, müssen Sie folgendes Kommando eingeben:
`ADD-FILE-LINK LINK-NAME=EDTSAM,FILE-NAME=`*datei*`,...,BUFFER-LENGTH=(STD,2),...`

Im EDT können Sie auch eingeben: `@OPEN F=` *datei* `,TYPE=CATALOG`

3.3.6 Pragma ISOLATION LEVEL

Das Pragma ISOLATION LEVEL legt den Isolationslevel für die Datenbankzugriffe einer SQL- bzw. Utility-Anweisung fest.

Das Pragma hat nur in folgenden SQL-Anweisungen eine Wirkung:

- CALL und in Routinen (siehe Abschnitt „Pragmas in Routinen“)
- Cursorbeschreibung (für dynamischen Cursor)
- DECLARE CURSOR (für statischen Cursor)
- DELETE
- INSERT
- MERGE
- SELECT
- UPDATE

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

> **ACHTUNG!** Das Isolationslevel READ NOWAIT kann nur im Pragma spezifiziert, nicht aber in der SQL-Anweisung SET TRANSACTION angegeben werden. Das Verhalten beim Isolationslevel READ NOWAIT entspricht dem Verhalten beim Konsistenzlevel 1, siehe "SET TRANSACTION - Transaktionseigenschaften festlegen".
>
> Wenn Sie einen geringeren Isolationslevel angeben als für die Transaktion festgelegt ist, dann ist der festgelegte Isolationslevel der Transaktion nicht mehr garantiert!

Die Isolationslevel sind im Abschnitt „SET TRANSACTION - Transaktionseigenschaften festlegen“ beschrieben.

Wenn Sie das Pragma ISOLATION LEVEL angegeben haben, erfolgt jeder Datenbankzugriff, der mit dieser Anweisung zusammenhängt, unter diesem Isolationslevel.

3.3.7 Pragma LIMIT ABORT_EXECUTION

Das Pragma LIMIT ABORT_EXECUTION steuert den Ressourcenverbrauch in der Verarbeitung einer SQL-Anweisung. Mit diesem Pragma können Sie Anweisungen gezielt mit einem lokalen Abbruchskriterium versehen. Dieses lokale Abbruchskriterium ist restriktiver als das für komplexe Batch-Programme nötige globale Abbruchskriterium ABORT-EXECUTION. ABORT-EXECUTION wird mit RETRIEVAL-CONTROL bzw. MODIFY-RETRIEVAL-CONTROL eingestellt.

Das per LIMIT ABORT_EXECUTION eingestellte lokale Abbruchskriterium

- gilt nur für den aktuellen Auftrag.
- kann nicht per MODIFY-RETRIEVAL-CONTROL übersteuert werden.
- ist wirkungslos, wenn das Pragma nicht in einer „suchenden“ Anweisung steht.
- ist wirkungslos, wenn als Wert 0 angegeben ist oder der angegebene Wert größer ist als der des globalen Abbruchkriteriums. In diesem Fall gilt der Wert des globalen Abbruchkriteriums.

Werden in einem Auftrag mehrere Pragmas LIMIT ABORT_EXECUTION angegeben, so gilt der letzte gültige Wert des Pragmas. Ist kein Pragma LIMIT ABORT_EXECUTION angegeben, so gilt das globale Abbruchkriterium.

In einer Folge von DECLARE CURSOR-, OPEN- und FETCH-Anweisungen muss das Pragma in der DECLARE CURSOR-Anweisung angegeben werden. Seine Wirkung zeigt sich je nach eingeschlagenem Suchpfad aber erst bei Ausführung der OPEN- bzw. FETCH-Anweisung.

Das Pragma kann auch bei CALL und in Routinen eingesetzt werden, siehe Abschnitt „Pragmas in Routinen“.

`LIMIT ABORT_EXECUTION` *blockzugriffe*

---

blockzugriffe

Mit diesem Argument legen Sie die Anzahl der logischen Blockzugriffe fest. Bei deren Erreichen wird die Trefferermittlung abgebrochen und die Anweisung beendet. Die Anzahl der Blockzugriffe wird als vorzeichenlose Ganzzahl im Wertebereich von 0 bis 2147483647 angegeben.

3.3.8 Pragma LOCK MODE

Das Pragma LOCK MODE stellt den Sperrmodus ein. Es wirkt nur in SQL-DML-Anweisungen..

Das Pragma kann bei CALL und in Routinen eingesetzt werden, siehe Abschnitt „Pragmas in Routinen“.

```
LOCK MODE EXCLUSIVE
```

Ist LOCK MODE EXCLUSIVE angegeben, so erfolgt jeder Zugriff auf die Datenbank, der direkt oder indirekt mit dieser SQL-Anweisung zusammenhängt, mit Exklusiv-Sperren. Ansonsten wird der Sperrmodus vom System festgelegt.

3.3.9 Pragma LOOP LIMIT

Mit dem Pragma LOOP LIMIT können Sie die Anzahl der Schleifendurchläufe in einer Routine begrenzen.

Das Pragma LOOP LIMIT ist vor der SQL-Anweisung CALL und vor anderen DML-Anweisungen wirksam. Bei Angabe vor DML-Anweisungen wirkt das Pragma auf alle User Defined Functions (UDF) und die darin enthaltenen Routinen der DML-Anweisung. Das Pragma hat vor SQL-Anweisungen in einer Routine keine Wirkung.

`LOOP LIMIT` *vorzeichenlose_ganzzahl*

---

vorzeichenlose_ganzzahl

Gibt die maximale Anzahl der Durchläufe für eine Laufschleife an.

Bei *vorzeichenlose_ganzzahl*=0 ist die Anzahl der Schleifendurchläufe unbegrenzt. *vorzeichenlose_ganzzahl* =0 ist auch der Standardwert, wenn das Pragma nicht angegeben wird.

Wenn dieses Pragma angegeben wird, dann wird für jede aufgerufene Laufschleife der betreffenden Routine der Schleifenrumpf nach Durchführung der angegebenen Anzahl von Durchläufen abgebrochen und es wird ein SQLSTATE gemeldet. Damit können Endlosschleifen verhindert werden.

3.3.10 Pragma PREFETCH

Das Pragma PREFETCH steuert den Schubmodus der SQL-Anweisung FETCH (Cursor positionieren). Der Schubmodus beschleunigt die Ausführung der Anweisung FETCH. Er ist nur wirksam, wenn FETCH den Cursor auf den nächsten Satz der Cursortabelle positioniert (FETCH NEXT...).

Über das Pragma PREFETCH können Sie den Schubmodus einschalten und einen Blockungsfaktor (n) festlegen. Bei Ausführung der ersten Anweisung FETCH NEXT... werden dann die Spaltenwerte des aktuellen Satzes gelesen, die folgenden n -1 Sätze der zugehörigen Cursortabelle werden in einem Zwischenpuffer gespeichert. Bei Ausführung der nachfolgenden n-1 Anweisungen FETCH NEXT..., die denselben Cursor bezeichnen, kann dann direkt, ohne DBH-Kontakt, auf den nächsten Satz zugegriffen werden.

Das Pragma PREFETCH hat nur in folgenden SQL-Anweisungen eine Wirkung:

- DECLARE CURSOR (für statischen Cursor)
- Cursorbeschreibung (für dynamische Cursor)

Enthält die Cursorbeschreibung der DECLARE CURSOR-Anweisung bzw. die Cursorbeschreibung für dynamische Cursor eine FOR UPDATE-Klausel, wird das Pragma PREFETCH ignoriert, der Schubmodus wird nicht eingeschaltet.

Der eingeschaltete Schubmodus macht den in der DECLARE CURSOR-Anweisung bzw. der Cursorbeschreibung vereinbarten Cursor zum Prefetch-Cursor.

Im linked-in-Betrieb wird ein Cursor mit Schubmodus nicht unterstützt.

PREFETCH *blockungsfaktor*

---

blockungsfaktor

Den Blockungsfaktor müssen Sie als vorzeichenlose Ganzzahl angeben (Datentyp SMALLINT).

Hat der Blockungsfaktor (n) einen Wert > 0, werden maximal n-1 Sätze der spezifizierten Cursortabelle in einem Zwischenpuffer gespeichert.

Hat der Blockungsfaktor den Wert 0, bleibt das Pragma PREFETCH ohne Wirkung.

Sie können die Wirksamkeit des Pragmas und damit den Schubmodus ein- bzw. ausschalten, indem Sie für n einen Wert > 0 bzw. den Wert 0 angeben.

Bei eingeschaltetem Schubmodus gelten folgende Einschränkungen:

- In derselben Übersetzungseinheit ist für den Prefetch-Cursor *cursor* nur noch die Anweisung FETCH NEXT erlaubt. Folgende SQL-Anweisungen sind nicht mehr ausführbar::
  - UPDATE ... WHERE CURRENT of *cursor*
  - DELETE ... WHERE CURRENT of *cursor*
  - STORE *cursor*
  - FETCH *cursor* mit einer anderen Cursorpositionierung als NEXT bzw. mit einer von der ersten Anweisung FETCH NEXT verschiedenen INTO-Klausel.

- Nach Ausführung einer FETCH NEXT-Anweisung, deren INTO-Klausel den Namen eines SQL-Deskriptorbereichs enthält, darf dieser SQL-Deskriptorbereich nicht durch eine SET DESCRIPTOR-, DESCRIBE- oder DEALLOCATE DESCRIPTOR-Anweisung verändert werden.
- Der Prefetch-Cursor darf stets nur mit derselben Anweisung FETCH NEXT angesprochen werden, d.h. mit derselben Anweisung in einer Schleife oder einem Unterprogramm..

3.3.11 Pragma UTILITY MODE

Das Pragma UTILITY MODE legt fest, ob für die SQL-Anweisung, in der dieses Pragma angegeben ist, die Transaktionssicherung wirksam sein soll. Die Transaktionssicherung ermöglicht, dass im Fehlerfall eine Transaktion auf einen konsistenten Zustand zurückgesetzt wird.

Das Pragma UTILITY MODE ist nur in der SQL-Anweisung ALTER TABLE wirksam.

Es wirkt nur, wenn die ALTER TABLE-Anweisung Spalten einer Basistabelle hinzufügt, ändert oder löscht. Bei einer ALTER TABLE-Anweisung, die Integritätsbedingungen zufügt oder löscht, bleibt das Pragma UTILITY MODE ohne Wirkung.

```
UTILITY MODE {ON | OFF}
```

ON	Bei der Ausführung der SQL-Anweisung wird die Transaktionssicherung ausgeschaltet. Die zugehörige ALTER TABLE-Anweisung eröffnet keine Transaktion. Sicherungsdaten für die ALTER TABLE-Anweisung werden nicht gespeichert. Führt ein Fehler zum Abbruch der Anweisung, ist das Rücksetzen auf einen konsistenten Zustand nicht möglich. Der Space, auf dem die zu ändernde Basistabelle liegt, ist im Fehlerfall defekt und muss mit Hilfe der Utility-Anweisung RECOVER repariert werden (siehe Handbuch „ SQL-Sprachbeschreibung Teil 2: Utilities“).
OFF	Das Pragma hat keine Auswirkungen. Die Transaktionssicherung bleibt eingeschaltet.

Eine ALTER TABLE-Anweisung, für die das Pragma UTILITY MODE ON eingeschaltet und wirksam ist, wird in folgenden Fällen mit Fehlermeldung abgebrochen:

- wenn eine Transaktion aktiv ist
- wenn es sich um eine ALTER TABLE-Anweisung zum kaskadierenden Löschen einer Spalte handelt, nämlich mit DROP COLUMN *spalte* CASCADE
- wenn es sich um eine ALTER TABLE-Anweisung zum Löschen einer Spalte handelt und noch ein Index für diese Spalte definiert ist
- wenn es sich um eine ALTER TABLE Anweisung zum Hinzufügen einer Spalte mit Indexdefinition für diese Spalte handelt

Wenn Sie für eine ALTER TABLE-Anweisung das Pragma UTILITY MODE nicht angeben, gilt als Standardeinstellung UTILITY MODE OFF.

> **ACHTUNG!** Das Pragma UTILITY MODE ON hat zur Folge, dass nach einem Fehler oder Consistency Check der Space, auf dem die zu ändernde Basistabelle liegt, defekt ist. Um Datenverlust zu vermeiden, sollten Sie vor Eingabe der ALTER TABLE- Anweisung den Space sichern. Sie brauchen die Sicherung, wenn Sie einen defekten Space mit Hilfe der Utility-Anweisung RECOVER reparieren wollen.

3.4 Namen

Namen sind Zeichenfolgen, die verwendet werden, um Objekte zu identifizieren.

In SESAM/SQL gibt es Namen für folgende SQL-Objekte:

- Datenbank (Catalog)
- Schema
- Space
- Storage Group
- Tabelle (Basistabelle, View, Korrelation)
- Spalte
- Index
- Integritätsbedingung
- Berechtigungsschlüssel
- Cursor
- Routine (Routinen-Parameter, lokale Variable, Fehler)
- Marke
- Dynamisch formulierte Anweisung:
  Der Name für eine dynamisch formulierte Anweisung wird in diesem Handbuch **Anweisungsbezeichner** genannt, um ihn vom eigentlichen Namen der Anweisung, wie z.B. SELECT, zu unterscheiden.
- Symbolischer Attributname einer CALL-DML-Spalte:
  Die Syntax für den symbolischen Attributnamen einer Spalte entspricht der Syntax für symbolische Attributnamen in SESAM/SQL-Version 1.x.
- Benutzervariable:
  Der Name für eine Benutzervariable muss den Konventionen der jeweiligen Programmiersprache entsprechen. Diese Konventionen sind in den Handbüchern der jeweiligen Programmiersprache beschrieben und hier nicht erklärt.

3.4.1 Einfache Namen

Einfache Namen sind entweder reguläre Namen aus Buchstaben, Ziffern und Unterstrich, die nicht in Anführungszeichen eingeschlossen werden, oder Spezialnamen, die in Anführungszeichen eingeschlossen werden müssen.

einf_name ::= { *regulärer_name* | *spezialname* }

regulärer_name ::= *buchstabe* [{ *buchstabe* | *ziffer* | _ }] ...

spezialname ::= " *zeichen...* "

buchstabe ::= a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p|q|r|s|t|u|v|w|x|y|z|

A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|T|U|V|W|X|Y|Z

ziffer ::= 0|1|2|3|4|5|6|7|8|9

regulärer_name

Regulärer Name, der nicht in Anführungszeichen eingeschlossen wird. Ein regulärer Name darf kein reserviertes SQL-Schlüsselwort sein (siehe Abschnitt „SQL-Schlüsselwörter“).

buchstabe

Kleinbuchstabe zwischen a und z oder Großbuchstabe zwischen A und Z des SESAM/SQL-Zeichenvorrats. Kleinbuchstaben werden automatisch in Großbuchstaben umgewandelt. Umlaute dürfen nicht verwendet werden.

ziffer

Ziffer zwischen 0 und 9.

_ Unterstrich

spezialname

Spezialname, der in Anführungszeichen eingeschlossen wird. Ein Spezialname darf ein reserviertes SQL-Schlüsselwort sein und kann Sonderzeichen enthalten.

zeichen

Das erste Zeichen darf kein Unterstrich sein. Ansonsten können Sie für *zeichen* ein beliebiges abdruckbares Zeichen (d.h. >= X'40') des SESAM/SQL-Zeichenvorrats angeben. Groß- und Kleinbuchstaben werden unterschieden. Ist *zeichen* ein Anführungszeichen (X'7F'), muss es verdoppelt werden. Ein doppeltes Anführungszeichen gilt als **ein** Zeichen.

Gleichheit von einfachen Namen

Zwei reguläre Namen gelten als gleich, wenn nach Umwandlung der Buchstaben in Großbuchstaben die Zeichen an den sich entsprechenden Zeichenpositionen jeweils gleich sind.

Ein regulärer Name und ein Spezialname gelten als gleich, wenn nach Umwandlung der Buchstaben des regulären Namens in Großbuchstaben und Entfernen der Anführungszeichen beim Spezialnamen die Zeichen an den sich entsprechenden Zeichenpositionen jeweils gleich sind. Bei unterschiedlich langen Zeichenfolgen wird die kürzere mit Leerzeichen aufgefüllt.

Zwei Spezialnamen gelten als gleich, wenn nach Entfernen der Anführungszeichen die Zeichen an den sich entsprechenden Zeichenpositionen jeweils gleich sind. Bei unterschiedlich langen Zeichenfolgen wird die kürzere mit Leerzeichen aufgefüllt.

Beispiel

Folgende einfache Namen gelten als gleich:

```
ABc
abc
"ABC"
"ABC     "
```

Folgende einfache Namen sind verschieden:

`Abc` und `"Abc"`
`"ABC"` und `"abc"`

Gleiche Namen können in jeder Referenz gegeneinander ausgetauscht werden.

Folgende Namen von Datenbankobjekten sind einfache Namen:

{ *anweisungsbezeichner*
| *berechtigungsschlüssel*
| *catalog*
| *cursor*
| *einf_basistabellenname*
| *einf_bedingungsname*
| *einf_indexname*
| *einf_routinenname*
| *einf_schemaname*
| *einf_spacename*
| *einf_stogroupname*
| *einf_viewname*
| *fehlername*
| *korrelationsname*
| *lokale_variable*
| *marke*
| *routinenparameter*
| *spalte* }
::= *einf_name*

anweisungsbezeichner

Name für eine dynamisch formulierte Anweisung. Der Anweisungsbezeichner muss innerhalb der Übersetzungseinheit eindeutig sein.
Der Anweisungsbezeichner darf max. 18 Zeichen lang sein.

berechtigungsschlüssel

Name für einen Berechtigungsschlüssel. Die ersten 10 Zeichen des Berechtigungsschlüssels müssen innerhalb der Datenbank eindeutig sein.
Wird der Name des Berechtigungsschlüssels ohne Anführungszeichen angegeben, darf er nur Buchstaben und Ziffern enthalten. Wird der Berechtigungsschlüssel in Anführungszeichen angegeben, muss er mit einem Großbuchstaben beginnen und darf nur Großbuchstaben, Ziffern und die Sonderzeichen - und . enthalten. Die Sonderzeichen dürfen nicht am Ende des signifikanten Teils (erste 10 Zeichen) des Berechtigungsschlüssels vorkommen.
Nicht erlaubt sind die Zeichenfolgen "..", ".-" und "-.".
Die Zeichenfolge "--" ist erlaubt.
Der Berechtigungsschlüssel darf max. 18 Zeichen lang sein.

catalog

Name für eine Datenbank. Wird der Datenbankname ohne Anführungszeichen angegeben, darf er nur Buchstaben und Ziffern enthalten. Wird der Datenbankname in Anführungszeichen angegeben, muss er mit einem Großbuchstaben beginnen und darf nur Großbuchstaben, Ziffern und die Sonderzeichen - und . enthalten. Die Sonderzeichen dürfen nicht am Ende des Datenbanknamens vorkommen. Nicht erlaubt sind die Zeichenfolgen "..", ".-" und "-.". Die Zeichenfolge "--" ist erlaubt. Der Datenbankname darf max. 18 Zeichen lang sein.

cursor

Name für einen Cursor. Pro Übersetzungseinheit darf derselbe Cursorname nur einmal in einer DECLARE CURSOR-Anweisung vorkommen.
Der Cursorname darf max. 18 Zeichen lang sein.

einf_basistabellenname

Name für eine Basistabelle. Der einfache Name der Basistabelle muss innerhalb der Basistabellen- und Viewnamen des Schemas eindeutig sein.
Der einfache Basistabellenname darf max. 31 Zeichen lang sein.

einf_bedingungsname

Name für eine Integritätsbedingung. Der einfache Name muss innerhalb der Integritätsbedingungsnamen eines Schemas eindeutig sein.
Der einfache Name einer Integritätsbedingung darf max. 31 Zeichen lang sein.

einf_indexname

Name für einen Index. Der einfache Indexname muss innerhalb der Indexnamen des Schemas eindeutig sein.
Der einfache Indexname darf max. 18 Zeichen lang sein.

einf_routinenname

Name einer Routine. Der einfache Routinen-Name muss innerhalb der Routinen-Namen des Schemas eindeutig sein.
Der einfache Routinen-Name darf max. 31 Zeichen lang sein.

einf_schemaname

Name für ein Schema. Der einfache Schemaname muss innerhalb der Schemanamen der Datenbank eindeutig sein.
Der einfache Schemaname darf max. 31 Zeichen lang sein.

einf_spacename

Name für einen Space. Die ersten 12 Zeichen des einfachen Spacenamens müssen innerhalb der Spacenamen der Datenbank eindeutig sein. Wird der Spacename ohne Anführungszeichen angegeben, darf er nur Buchstaben und Ziffern enthalten. Wird der Spacename in Anführungszeichen angegeben, muss er mit einem Großbuchstaben beginnen und darf nur Großbuchstaben, Ziffern und die Sonderzeichen - und . enthalten. Die Sonderzeichen dürfen nicht am Ende des signifikanten Teils (erste 12 Zeichen) des Spacenamens vorkommen.
Nicht erlaubt sind die Zeichenfolgen "..", ".-" und "-.".
Die Zeichenfolge "--" ist erlaubt. Der einfache Spacename darf max. 18 Zeichen lang sein.

einf_stogroupname

Name für eine Storage Group. Der einfache Name der Storage Group muss innerhalb der Storage Group-Namen einer Datenbank eindeutig sein.
Der einfache Name einer Storage Group darf max. 18 Zeichen lang sein.

einf_viewname

Name für einen View. Der einfache Name des View muss innerhalb der Basistabellen- und Viewnamen des Schemas eindeutig sein.
Der einfache Viewname darf max. 31 Zeichen lang sein.

fehlername

Name für einen Fehler oder einen SQLSTATE in einer COMPOUND-Anweisung. Alle Fehlernamen der COMPOUND-Anweisung müssen sich voneinander unterscheiden..
Der Fehlername darf max. 31 Zeichen lang sein.

korrelationsname

Umbenennung einer Tabelle.
Der Korrelationsname darf max. 31 Zeichen lang sein.

lokale_variable

Name für eine lokale Variable in einer COMPOUND-Anweisung. Der Variablenname muss innerhalb der COMPOUND-Anweisung eindeutig sein und sich von allen Parameternamen der Routine unterscheiden.
Der Variablenname darf max. 31 Zeichen lang sein.

marke

Name für eine Marke in einer Routine. Die Marke darf nicht identisch sein mit einer anderen Marke innerhalb des Anweisungsrumpfes.
Reservierte Schluesselwörter sowie folgende Namen sind nicht als Markennamen erlaubt: ATOMIC, DO, ELSEIF, ITERATE, IF, LEAVE, LOOP, REPEAT, RESIGNAL, SIGNAL, UNTIL, WHILE.
Der Markenname darf max. 31 Zeichen lang sein.

routinenparameter

Name für einen Routinen-Parameter. Der Parametername muss innerhalb der Routine eindeutig sein. Der Parametername darf max. 31 Zeichen lang sein.

spalte

Name für eine Spalte. Der Spaltenname muss innerhalb der Tabelle eindeutig sein. Der einfache Spaltenname darf max. 31 Zeichen lang sein.

3.4.2 Qualifizierte Namen

Um in einer SQL-Anweisung verschiedene Objekte mit dem gleichen Namen eindeutig identifizieren zu können, gibt es die Möglichkeit, Namen zu qualifizieren. Folgende Qualifikationen sind möglich:

- Qualifikation mit dem Datenbanknamen für:
  Schema, Space, Storage Group, Tabelle, Index, Integritätsbedingung und Routine
- Qualifikation mit dem Schemanamen für:
  Tabelle, Index, Integritätsbedingung und Routine
- Qualifikation mit dem Tabellen- bzw. Korrelationsnamen für:
  Spalte (siehe "Tabellenangabe")

Folgende Syntaxübersicht fasst diese Möglichkeiten zusammen:

---

qualifizierter_name ::=
{ *index*
| *integritätsbedingungsname*
| *routine*
| *schema*
| *space*
| *stogroup*
| *tabelle* }

index ::= [[*catalog* .] *einf_schemaname* .] *einf_indexname*

integritätsbedingungsname ::= [[*catalog* .] *einf_schemaname* .] *einf_bedingungsname*

routine ::= [[*catalog* .] *einf_schemaname* .] *einf_routinenname*

schema ::= [*catalog* .] *einf_schemaname*

space ::= [*catalog* .] *einf_spacename*

stogroup ::= [*catalog* .] *einf_stogroupname*

tabelle ::=

```
{       [[ catalog . ] einf_schemaname . ]  einf_basistabellenname |
        [[ catalog . ] einf_schemaname . ]  einf_viewname |
        korrelationsname }
```

Implizite Qualifikation

Es gilt folgende implizite Qualifikation:

- Ist keine Schema-Qualifikation angegeben, bezieht sich der Name auf das voreingestellte Schema.
- Ist keine Catalog-Qualifikation angegeben, bezieht sich der Name auf die voreingestellte Datenbank.

Schema und Datenbank werden mit der Precompiler-Option SOURCE-PROPERTIES voreingestellt (siehe Handbuch „ ESQL-COBOL für SESAM/SQL-Server“). Die voreingestellten Datenbank- und Schemanamen

können mit SET CATALOG bzw. SET SCHEMA umdefiniert werden. Die umdefinierte Voreinstellung gilt für alle Anweisungen, die nach der Umdefinition mit PREPARE vorbereitet oder mit EXECUTE IMMEDIATE ausgeführt werden, bis zur nächsten Umdefinition oder bis zum Ende der SQL-Session.

> ¡ Bei CREATE- und GRANT-Anweisungen innerhalb einer CREATE SCHEMA-Anweisung gelten andere Regeln für die implizite Qualifikation (siehe Abschnitt „CREATE SCHEMA - Schema erzeugen“).

Beispiel

Die Qualifikation im Tabellennamen gibt an, zu welchem Schema und zu welcher Datenbank die Tabelle gehört:

`auftragkunden.auftragsver.kunde:`

Tabelle KUNDE im Schema AUFTRAGSVER der Datenbank AUFTRAGKUNDEN.

`auftragsver.kunde:`

Tabelle KUNDE im Schema AUFTRAGSVER der voreingestellten Datenbank.

`kunde:`

Tabelle KUNDE im voreingestellten Schema.

Übersicht

Art des Namens	Beispiele	Bemerkung
regulärer Name	`Kunde` `kunde`	„Kunde“ und „kunde“ sind gleichbedeutend
	`auftrag_2`	Ziffern und Unterstrich sind zulässig
Spezialname	`"TAB-ELLE"` `";$&%!"`	Sonderzeichen sind zulässig
	`"mit_2_Anfuehrungszeichen:"""""`	Anführungszeichen müssen doppelt angegeben werden
einfacher Name	`auftragsver`	Schema AUFTRAGSVER
qualifizierter Name	`auftragkunden.auftragsver.View1`	Tabelle VIEW1 im Schema AUFTRAGSVER der Datenbank AUFTRAGKUNDEN
	`"View"."SELECT(5)"`	Einfache Spalte SELECT(5) der Tabelle View
	`"VIEW"."SELECT"(5)`	Ausprägung der multiplen Spalte SELECT der Tabelle VIEW
	`A.anr`	Mit dem Korrelationsnamen A qualifizierter Spaltenname ANR

Tabelle 10: Namen in SESAM/SQL

3.4.3 Namen definieren

Der Name eines Objekts wird in der Regel bei der Definition des Objekts mit der entsprechenden SQL-Anweisung festgelegt. Damit ist der Name eingeführt und das Objekt kann in nachfolgenden Anweisungen mit diesem Namen angesprochen werden.

Folgende Tabelle zeigt, wie die unterschiedlichen Namen definiert bzw. deklariert werden:

SQL-Objekt	**SQL-Anweisung oder -Anweisungsteil**
Datenbank (Catalog)	CREATE CATALOG (Utility-Anweisung)
Schema	CREATE SCHEMA
Tabelle: Basistabelle View Korrelation	 CREATE TABLE CREATE VIEW Tabellenangabe in Abfrage-Ausdruck
Spalte	CREATE TABLE, ALTER TABLE CREATE VIEW, Abfrage-Ausdruck
Integritätsbedingung	CONSTRAINT-Klausel in CREATE TABLE, ALTER TABLE
Index	CREATE INDEX
Routine: Prozedur User Defined Function (UDF)	 CREATE PROCEDURE CREATE FUNCTION
Storage Group	CREATE STOGROUP
Space	CREATE SPACE
Berechtigungsschlüssel	CREATE USER
Cursor	DECLARE CURSOR
Anweisungsbezeichner	PREPARE

Tabelle 11: Namen definieren

4 Datentypen und Werte

Dieses Kapitel gliedert sich in folgende Abschnitte:

- Überblick
- Datentypen
- Werte
- Zuweisungsregeln

Das Kapitel hat zwei Hauptteile. Nach einem Überblick über die SESAM/SQL-Datentypen und die zugehörigen Wertebereiche werden im ersten Teil alle Einzelheiten erklärt, die Sie bezüglich Datentypen bei der Definition der Tabellenspalten wissen müssen:

- Syntax
- Wertebereich, den der Datentyp festlegt
- Verträglichkeit von Datentypen

> ¡ Bei Routinen haben auch die Routinen-Parameter und die lokalen Variablen einen Datentyp.

Im zweiten Teil werden dann alle Einzelheiten erklärt, die Sie bei der Verwendung der Werte eines Datentyps kennen müssen:

- Syntax der Literale
- Regeln für das Eintragen der Werte in Tabellenspalten, Routinen-Parameter und lokale Variablen
- Regeln für die Verwendung der Werte in Ausdrücken und Suchbedingungen
- Regeln für die Datentypverträglichkeit und Konvertierung bei Zuweisungen

4.1 Überblick über Datentypen und zugehörige Wertebereiche

Die Werte bzw. Daten, die als Inhalte der Tabellen vorkommen können, sind in verschiedene Wertebereiche unterteilt. Die Wertebereiche werden durch Datentypen festgelegt.

4.1.1 Einteilung der Datentypen

Die Datentypen, die SESAM/SQL zur Verfügung stellt, sind in folgende Gruppen eingeteilt:

- Zeichenketten:
  - Alphanumerische Datentypen:
    - CHARACTER
    - CHARACTER VARYING

    > ℹ Das Wort „alphanumerisch“ drückt in der Handbuchreihe SESAM/SQL die Zugehörigkeit zu einem EBCDIC-Zeichensatz aus, z.B. alphanumerischer Datentyp, alphanumerischer Wert, alphanumerisches Literal. Für die alphanumerischen Datentypen werden in diesem Handbuch die Kurzformen CHAR und VARCHAR verwendet.

  - National-Datentypen:
    - NATIONAL CHARACTER
    - NATIONAL CHARACTER VARYING

    > ℹ Das Wort „National“ drückt in der Handbuchreihe SESAM/SQL die Zugehörigkeit zum Unicode-Zeichensatz aus, z.B. National-Datentyp, National-Wert, National-Literal. Für die National-Datentypen werden in diesem Handbuch die Kurzformen NCHAR und NVARCHAR verwendet.

- Numerische Datentypen
  - Ganzzahlige Datentypen:
    - SMALLINT
    - INTEGER
  - Festpunktzahl-Datentypen:
    - NUMERIC
    - DECIMAL
  - Gleitpunktzahl-Datentypen:
    - REAL
    - DOUBLE PRECISION
    - FLOAT
- Zeit-Datentypen:
  - DATE
  - TIME
  - TIMESTAMP

4.1.2 Wertebereich

Jeder Datentyp definiert einen zugehörigen Wertebereich. Entsprechend den Datentypgruppen gibt es alphanumerische Werte, National-Werte, numerische Werte und Zeitwerte. Zusätzlich gibt es NULL-Werte (siehe Abschnitt „NULL-Wert“).

Für die Werte gibt es entsprechende Literale und Regeln, wie die Werte verwendet werden dürfen. Diese sind im Abschnitt „Werte“ beschrieben.

4.1.3 Spalte

Die Sätze einer Tabelle sind in Spalten aufgeteilt. Eine Spalte besitzt einen Namen und einen Datentyp.

SESAM/SQL unterscheidet zwischen einfachen und multiplen Spalten.

Bei einer einfachen Spalte kann pro Satz genau ein Wert gespeichert werden.

Bei einer multiplen Spalte können pro Satz mehrere Werte desselben Datentyps gespeichert werden. Eine multiple Spalte besteht aus mehreren Spaltenelementen. In jedem Spaltenelement kann pro Satz genau ein Wert gespeichert werden. Der Wert eines Spaltenelements heißt **Ausprägung**. Der Wert einer multiplen Spalte heißt **Aggregat**. Ein Aggregat setzt sich aus den Ausprägungen der einzelnen Spaltenelemente zusammen.

Ein Spaltenelement wird mit seiner Positionsnummer innerhalb der multiplen Spalte angesprochen. Zusammenhängende Teilbereiche einer multiplen Spalte werden durch die Positionsnummern des ersten und des letzten Spaltenelements dieses Teilbereichs angegeben.

Beispiel

X[2] oder X(2)

zweites Spaltenelement der multiplen Spalte X

X[4..7] oder X(4..7)

Teilbereich aus dem 4., 5., 6. und 7. Spaltenelement der multiplen Spalte X

4.1.4 Parameter von Routinen und lokale Variable

In Routinen können Parameter und lokale Variable benutzt werden. Parameter und lokale Variable besitzen einen Namen und einen Datentyp. Im Gegensatz zu Spalten können sie nicht multipel sein.

4.2 Datentypen

Für jede Spalte einer Tabelle muss bei der Spaltendefinition mit CREATE TABLE bzw. ALTER TABLE ein Datentyp angegeben werden. Er legt fest, welche Werte in die Spalte eingetragen werden können. Zum Teil können mit ALTER TABLE nach der Definition noch Änderungen an der Datentypfestlegung vorgenommen werden.

BLOBs (Binary Large Objects) basieren in SESAM/SQL auf vorhandenen Datentypen und sind daher kein neuer Datentyp. Die Struktur und Bearbeitung solcher Objekte wird im Kapitel „SESAM-CLI“ und im „ Basishandbuch“ beschrieben.

NULL-Wert ausschließen

Soll für eine Spalte der NULL-Wert ausgeschlossen werden, muss dies bei der Definition mit CREATE TABLE bzw. ALTER TABLE durch eine Nicht-NULL-Bedingung angegeben werden (siehe Abschnitt „Spaltenbedingung“).

Multiple Spalte

Alle Elemente einer multiplen Spalte haben denselben Datentyp. Für eine multiple Spalte kann jeder Datentyp außer VARCHAR und NVARCHAR verwendet werden. Die Dimension einer multiplen Spalte gibt die Anzahl der Elemente an; sie wird bei der Datentypfestlegung angegeben und muss zwischen 1 und 255 liegen.

4.2.1 Übersicht über SQL-Datentypen

In der folgenden Übersicht ist die Syntax für alle SQL-Datentypen zur Spaltendefinition zusammengestellt:

```
datentyp ::=
{
    [{[ dimension ] | ( dimension )}] CHAR[ACTER][( länge )] |
    { CHAR[ACTER] VARYING | VARCHAR } ( max ) |
    [{[ dimension ] | ( dimension ) }] { NATIONAL CHAR[ACTER] | NCHAR }[( cu_länge
    [CODE_UNITS])] |
    { NATIONAL CHAR[ACTER] VARYING | NCHAR VARYING | NVARCHAR } ( cu_max [CODE_UNITS]
    ) |
    [{[ dimension ] | ( dimension )}]
    {
        SMALLINT |
        INT[EGER] |
        NUMERIC [( stellen [, bruchteil ])] |
        DEC[IMAL][( stellen [, bruchteil ])] |
        REAL |
        DOUBLE PRECISION |
        FLOAT [( stellen )] |
        DATE |
        TIME(3) |
        TIMESTAMP(3)
    }
}
```

> ¡ In der Syntax kursiv gedruckte eckige Klammern sind Sonderzeichen und müssen in der Anweisung angegeben werden.

Die Datentypen sind in der Reihenfolge beschrieben, in der sie in der Übersicht aufgelistet sind.

4.2.2 Alphanumerische Datentypen und National-Datentypen

In den folgenden Abschnitten sind die alphanumerischen Datentypen und National-Datentypen beschrieben.

4.2.2.1 CHARACTER - Zeichenketten fester Länge

Der Datentyp CHARACTER oder CHAR wird für Spalten verwendet, die alphanumerische Werte (siehe Abschnitt „Alphanumerische Literale“) fester Länge enthalten können.

```
[ { [ dimension ] | ( dimension ) } ] CHAR[ACTER][( länge )]
```

dimension

Vorzeichenlose Ganzzahl zwischen 1 und 255. Die Spalte ist eine multiple Spalte; *dimension* gibt die Anzahl der Spaltenelemente an. *dimension* kann in eckigen oder runden Klammern angegeben werden.

> ¡ In der Syntax kursiv gedruckte eckige Klammern sind Sonderzeichen und müssen in der Anweisung angegeben werden.

dimension nicht angegeben:
Die Spalte ist eine einfache Spalte.

länge

Vorzeichenlose Ganzzahl zwischen 1 und 256, die die Länge der CHAR-Spalte in Zeichen angibt.

länge nicht angegeben:
Es gilt *länge*=1.

Wertebereich für CHAR-Spalten

Eine CHAR-Spalte kann alphanumerische Werte mit der für die Spalte festgelegten Länge enthalten.

Beispiel

Die Tabelle KUNDE enthält 6 CHAR-Spalten unterschiedlicher Länge.
Die Werte, die die Spalten enthalten können, sind alphanumerische Zeichenketten der Länge 3, 25, 40 bzw. 50:

```
firma      CHAR(40) NOT NULL
strasse    CHAR(40)
ort        CHAR(40)
land       CHAR(3)
ktelefon   CHAR(25)
kinfo      CHAR(50)
```

4.2.2.2 CHARACTER VARYING - Zeichenketten variabler Länge

Der Datentyp CHARACTER VARYING oder VARCHAR wird für Spalten verwendet, die alphanumerische Werte (siehe Abschnitt „Alphanumerische Literale“) variabler Länge enthalten können.

```
CHAR[ACTER] VARYING( max ) | VARCHAR( max )
```

max

Vorzeichenlose Ganzzahl zwischen 1 und 32000, die die maximale Länge der VAR-CHAR-Spalten in Zeichen festlegt.

Wertebereich für VARCHAR-Spalten

Eine VARCHAR-Spalte kann alphanumerische Werte beliebiger Länge kleiner oder gleich der angegebenen maximalen Länge enthalten.

Beispiel

Sie definieren eine VARCHAR-Spalte BESCHREIBUNG, die alphanumerische Werte mit einer Länge von maximal 1000 Zeichen aufnehmen kann, wie folgt:

```
beschreibung VARCHAR(1000)
```

4.2.2.3 NATIONAL CHARACTER - Zeichenketten fester Länge

Der Datentyp NATIONAL CHARACTER oder NCHAR wird für Spalten verwendet, die National-Werte (siehe Abschnitt „National-Literale“) fester Länge enthalten können.

```
[ { [ dimension ] | ( dimension ) } ]
{ NATIONAL CHAR[ACTER] | NCHAR } [ ( cu_länge [CODE_UNITS] ) ]
```

dimension

Vorzeichenlose Ganzzahl zwischen 1 und 255. Die Spalte ist eine multiple Spalte; *dimension* gibt die Anzahl der Spaltenelemente an. *dimension* kann in eckigen oder runden Klammern angegeben werden.

> ¡ In der Syntax kursiv gedruckte eckige Klammern sind Sonderzeichen und müssen in der Anweisung angegeben werden.

dimension nicht angegeben:
Die Spalte ist eine einfache Spalte.

cu_länge

Vorzeichenlose Ganzzahl zwischen 1 und 128, die die Länge der NCHAR-Spalte in Code Units angibt.

cu_länge nicht angegeben:
Es gilt *cu_länge*=1.

> ¡ In SESAM/SQL wird für Unicode-Zeichenketten die Codierungsform UTF-16 verwendet, bei der jede Code Unit aus 2 Byte besteht.

Wertebereich für NCHAR-Spalten

Eine NCHAR-Spalte kann National-Werte mit der für die Spalte festgelegten Länge enthalten..

Beispiel

Die Tabelle HANDBUECHER enthält eine INTEGER- und zwei NCHAR-Spalten fester Länge. Die Werte, die die NCHAR-Spalten enthalten können, sind National Zeichenketten der Länge 20 bzw. 30:

```
bestellnummer       INTEGER
sprache             NCHAR(20)
titel               NCHAR(30)
```

4.2.2.4 NATIONAL CHARACTER VARYING - Zeichenketten variabler Länge

Der Datentyp NATIONAL CHARACTER VARYING oder NVARCHAR wird für Spalten verwendet, die National-Werte (siehe Abschnitt „National-Literale“) variabler Länge enthalten können.

```
{ NATIONAL CHAR[ACTER] VARYING | NCHAR VARYING | NVARCHAR }( cu_max [CODE_UNITS])
```

cu_max

Vorzeichenlose Ganzzahl zwischen 1 und 16000, die die maximale Länge der NVAR-CHAR-Spalten in Code Units festlegt.

> **i** In SESAM/SQL wird für Unicode-Zeichenketten die Codierungsform UTF-16 verwendet, bei der jede Code Unit aus 2 Byte besteht.

Wertebereich für NVARCHAR-Spalten

Eine NVARCHAR-Spalte kann National-Werte beliebiger Länge kleiner oder gleich der angegebenen maximalen Länge enthalten.

Beispiel

Sie definieren eine NVARCHAR-Spalte BESCHREIBUNG_IN_GRIECHISCH, die National-Werte mit einer Länge von maximal 1000 Zeichen aufnehmen kann, wie folgt:

```
beschreibung_in_Griechisch NVARCHAR(1000)
```

4.2.3 Numerische Datentypen

In den folgenden Abschnitten sind die numerischen Datentypen beschrieben.

4.2.3.1 SMALLINT - Kleine Ganzzahlen

Der Datentyp SMALLINT wird für Spalten verwendet, die kleine Ganzzahlen (siehe Abschnitt „Numerische Werte“) aufnehmen können.

```
[{[ dimension ] | ( dimension )}] SMALLINT
```

dimension

Vorzeichenlose Ganzzahl zwischen 1 und 255. Die Spalte ist eine multiple Spalte;*dimension* gibt die Anzahl der Spaltenelemente an.

> In der Syntax kursiv gedruckte eckige Klammern sind Sonderzeichen und müssen in der Anweisung angegeben werden.

dimension nicht angegeben:
Die Spalte ist eine einfache Spalte.

Wertebereich für SMALLINT-Spalten

Der Wertebereich einer SMALLINT-Spalte ist -2^{15} bis $2^{15}-1$.

Beispiel

Sie definieren eine SMALLINT-Spalte MENGE wie folgt:

```
menge SMALLINT
```

4.2.3.2 INTEGER - Ganzzahlen

Der Datentyp INTEGER wird für Spalten verwendet, die große Ganzzahlen (siehe Abschnitt „Numerische Werte“) aufnehmen können.

```
[{[ dimension ] | ( dimension )}] INT[EGER]
```

dimension

Vorzeichenlose Ganzzahl zwischen 1 und 255. Die Spalte ist eine multiple Spalte;*dimension* gibt die Anzahl der Spaltenelemente an.

> In der Syntax kursiv gedruckte eckige Klammern sind Sonderzeichen und müssen in der Anweisung angegeben werden.

dimension nicht angegeben:
Die Spalte ist eine einfache Spalte.

Wertebereich für INTEGER-Spalten

Der Wertebereich einer INTEGER-Spalte ist -2^{31} bis $2^{31}-1$.

Beispiel

Die Tabelle LEISTUNG hat drei INTEGER-Spalten:

```
lnr             INTEGER
anr             INTEGER NOT NULL
lanz            INTEGER CHECK (lanz > 0)
```

4.2.3.3 NUMERIC - Festpunktzahlen

Der Datentyp NUMERIC wird für Spalten verwendet, die Festpunktzahlen (siehe Abschnitt „Numerische Werte“) aufnehmen können. Im Unterschied zu DECIMAL ist die interne Darstellung bei NUMERIC für Bildschirmausgaben effizienter.

```
[{[ dimension ] | ( dimension )}] NUMERIC[( stellen [, bruchteil ])]
```

dimension

Vorzeichenlose Ganzzahl zwischen 1 und 255. Die Spalte ist eine multiple Spalte; *dimension* gibt die Anzahl der Spaltenelemente an.

> In der Syntax kursiv gedruckte eckige Klammern sind Sonderzeichen und müssen in der Anweisung angegeben werden.

dimension nicht angegeben:
Die Spalte ist eine einfache Spalte.

stellen

Vorzeichenlose Ganzzahl zwischen 1 und 31, die die Gesamtanzahl der Dezimalstellen angibt.

stellen nicht angegeben:
Es gilt *stellen*=1.

bruchteil

Vorzeichenlose Ganzzahl zwischen 0 und *stellen*, die die Anzahl der Nachkommastellen angibt.

bruchteil nicht angegeben:
Es gilt *bruchteil*=0.

Wertebereich für NUMERIC-Festpunkt-Spalten

Eine NUMERIC-Festpunkt-Spalte kann Festpunktzahlen enthalten, deren Betrag 0 ist oder im Bereich von $10^{-\text{bruchteil}}$ bis $10^{\text{stellen-bruchteil}}-10^{-\text{bruchteil}}$ liegt.

Beispiel

Die Tabelle LEISTUNG hat drei NUMERIC-Festpunkt-Spalten:

```
lsatz           NUMERIC(5,0)
mwsatz          NUMERIC(2,2)
rnr             NUMERIC(4,0)
```

Die Spalte MWSATZ enthält Festpunktzahlen, die zwei Nachkommastellen und keine (von Null verschiedene) Vorkommastellen haben.

4.2.3.4 DECIMAL - Festpunktzahlen

Der Datentyp DECIMAL wird für Spalten verwendet, die Festpunktzahlen (siehe Abschnitt „Numerische Werte“) aufnehmen können.

Im Unterschied zu NUMERIC ist die interne Darstellung bei DECIMAL kürzer und für Berechnungen effizienter.

```
[{[ dimension ] | ( dimension )}] DEC[IMAL][( stellen [, bruchteil ])]
```

dimension

Vorzeichenlose Ganzzahl zwischen 1 und 255. Die Spalte ist eine multiple Spalte; *dimension* gibt die Anzahl der Spaltenelemente an.

> In der Syntax kursiv gedruckte eckige Klammern sind Sonderzeichen und müssen in der Anweisung angegeben werden.

dimension nicht angegeben:
Die Spalte ist eine einfache Spalte.

stellen

Vorzeichenlose Ganzzahl zwischen 1 und 31, die die Gesamtanzahl der Dezimalstellen angibt.

stellen nicht angegeben:
Es gilt *stellen*=1.

bruchteil

Vorzeichenlose Ganzzahl zwischen 0 und *stellen*, die die Anzahl der Nachkommastellen angibt.

bruchteil nicht angegeben:
Es gilt *bruchteil*=0.

Wertebereich für DECIMAL-Festpunkt-Spalten

Eine DECIMAL-Festpunkt-Spalte kann Festpunktzahlen enthalten, deren Betrag 0 ist oder im Bereich von $10^{-\text{bruchteil}}$ bis $10^{\text{stellen-bruchteil}}-10^{-\text{bruchteil}}$ liegt.

Beispiel

Sie definieren eine DECIMAL-Spalte GEWICHT mit sechs Vorkomma- und zwei Nachkommastellen:

```
gewicht DECIMAL(8,2)
```

4.2.3.5 REAL- Gleitpunktzahlen mit einfacher Genauigkeit

Der Datentyp REAL wird für Spalten verwendet, die Gleitpunktzahlen (siehe Abschnitt „Numerische Werte“) mit einfacher Genauigkeit aufnehmen können.

```
[{[ dimension ] | ( dimension )}] REAL
```

dimension

Vorzeichenlose Ganzzahl zwischen 1 und 255. Die Spalte ist eine multiple Spalte; *dimension* gibt die Anzahl der Spaltenelemente an.

> In der Syntax kursiv gedruckte eckige Klammern sind Sonderzeichen und müssen in der Anweisung angegeben werden.

dimension nicht angegeben:
Die Spalte ist eine einfache Spalte.

Wertebereich für REAL-Spalten

Eine REAL-Spalte kann Gleitpunktzahlen enthalten, deren Betrag 0 ist oder im Bereich von $5.4E^{-79}$ bis $7.2E^{+75}$ liegt.

Die Genauigkeit der REAL-Gleitpunktzahlen beträgt 21 Binärstellen, das sind etwa 6 Dezimalstellen.

Beispiel

Sie definieren eine REAL-Spalte GEWICHT:

```
gewicht REAL
```

4.2.3.6 DOUBLE PRECISION - Doppelte Genauigkeit

Der Datentyp DOUBLE PRECISION wird für Spalten verwendet, die Gleitpunktzahlen (siehee Abschnitt „Numerische Werte“) mit doppelter Genauigkeit aufnehmen können.

```
[{[ dimension ] | ( dimension )}] DOUBLE PRECISION
```

dimension

Vorzeichenlose Ganzzahl zwischen 1 und 255. Die Spalte ist eine multiple Spalte; *dimension* gibt die Anzahl der Spaltenelemente an.

> In der Syntax kursiv gedruckte eckige Klammern sind Sonderzeichen und müssen in der Anweisung angegeben werden.

dimension nicht angegeben:
Die Spalte ist eine einfache Spalte.

Wertebereich für DOUBLE PRECISION-Spalten

Eine DOUBLE PRECISION-Spalte kann Gleitpunktzahlen enthalten, deren Betrag 0 ist oder im Bereich von 5.4E $^{-79}$ bis 7.2E $^{+75}$ liegt.

Die Genauigkeit der DOUBLE PRECISION-Gleitpunktzahlen beträgt 53 Binärstellen, das sind etwa 16 Dezimalstellen.

Beispiel

Sie definieren eine DOUBLE PRECISION-Spalte GEWICHT:

```
gewicht DOUBLE PRECISION
```

4.2.3.7 FLOAT - Gleitpunktzahl

Der Datentyp FLOAT wird für Spalten verwendet, die Gleitpunktzahlen (siehe Abschnitt „Numerische Werte“) aufnehmen können. Die Genauigkeit kann angegeben werden.

```
[{[ dimension ] | ( dimension )}] FLOAT[( stellen )]
```

dimension

Vorzeichenlose Ganzzahl zwischen 1 und 255. Die Spalte ist eine multiple Spalte; *dimension* gibt die Anzahl der Spaltenelemente an.

> In der Syntax kursiv gedruckte eckige Klammern sind Sonderzeichen und müssen in der Anweisung angegeben werden.

dimension nicht angegeben:
Die Spalte ist eine einfache Spalte.

stellen

Vorzeichenlose Ganzzahl zwischen 1 und 53, die die Mindestanzahl der Binärstellen für die Mantisse angibt.

stellen nicht angegeben:
Es gilt *stellen*=1.

Wertebereich für FLOAT-Spalten

Eine FLOAT-Spalte kann Gleitpunktzahlen enthalten, deren Betrag 0 ist oder im Bereich von $5.4E^{-79}$ bis $7.2E^{+75}$ liegt.

Die Genauigkeit der FLOAT-Gleitpunktzahlen beträgt bei SESAM/SQL 53 Binärstellen, falls *stellen* größer als 21 ist, sonst 21 Binärstellen.

Beispiel

Sie definieren eine FLOAT-Spalte MESSWERT mit einer Genauigkeit von mindestens 30 Binärstellen:

```
messwert      FLOAT(30)
```

4.2.4 Zeit-Datentypen

In den folgenden Abschnitten sind die Zeit-Datentypen beschrieben.

4.2.4.1 DATE - Datum

Der Datentyp DATE wird für Spalten verwendet, die ein Datum (siehe Abschnitt „Zeitwerte“) aufnehmen können.

```
[{[ dimension ] | ( dimension )}] DATE
```

dimension

Vorzeichenlose Ganzzahl zwischen 1 und 255. Die Spalte ist eine multiple Spalte; *dimension* gibt die Anzahl der Spaltenelemente an.

> In der Syntax kursiv gedruckte eckige Klammern sind Sonderzeichen und müssen in der Anweisung angegeben werden.

dimension nicht angegeben:
Die Spalte ist eine einfache Spalte.

Wertebereich für DATE-Spalten

Eine DATE-Spalte kann Datumsangaben aus dem Bereich von 0001-01-01 bis 9999-12-31 enthalten. Die Datumsangabe muss die Regeln des Gregorianischen Kalenders erfüllen, auch wenn sie ein Datum vor Einführung des Gregorianischen Kalenders bezeichnet.

Beispiel

Die Tabelle AUFTRAG enthält drei DATE-Spalten:

```
adatum           DATE DEFAULT CURRENT_DATE
fertigist        DATE
fertigsoll       DATE
```

4.2.4.2 TIME - Uhrzeit

Der Datentyp TIME wird für Spalten verwendet, die eine Uhrzeit (siehe Abschnitt „Zeitwerte“) aufnehmen können.

```
[{[ dimension ] | ( dimension )}] TIME(3)
```

dimension

Vorzeichenlose Ganzzahl zwischen 1 und 255. Die Spalte ist eine multiple Spalte;*dimension* gibt die Anzahl der Spaltenelemente an.

> In der Syntax kursiv gedruckte eckige Klammern sind Sonderzeichen und müssen in der Anweisung angegeben werden.

dimension nicht angegeben:
Die Spalte ist eine einfache Spalte.

Wertebereich für TIME-Spalten

Eine TIME-Spalte kann Uhrzeiten aus dem Bereich von 00:00:00.000 bis 23:59:61.999 enthalten. Der Bereich für die Sekunde von 00.000 bis 61.999 erlaubt die Angabe von bis zu zwei Schaltsekunden.

Beispiel

Sie definieren eine TIME-Spalte WECKZEIT mit:

```
weckzeit    TIME(3)
```

4.2.4.3 TIMESTAMP - Zeitstempel

Der Datentyp TIMESTAMP wird für Spalten verwendet, die einen Zeitstempel (siehe Abschnitt „Zeitwerte“) aufnehmen können.

```
[{[ dimension ] | ( dimension )}] TIMESTAMP(3)
```

dimension

Vorzeichenlose Ganzzahl zwischen 1 und 255. Die Spalte ist eine multiple Spalte; *dimension* gibt die Anzahl der Spaltenelemente an.

> In der Syntax kursiv gedruckte eckige Klammern sind Sonderzeichen und müssen in der Anweisung angegeben werden.

dimension nicht angegeben:
Die Spalte ist eine einfache Spalte.

Wertebereich für TIMESTAMP-Spalten

Eine TIMESTAMP-Spalte kann Datumsangaben aus dem Bereich von 0001-01-01 bis 9999-12-31 und Uhrzeiten aus dem Bereich von 00:00:00.000 bis 23:59:61.999 enthalten.

Der Bereich für die Sekunde von 00.000 bis 61.999 erlaubt die Angabe von bis zu zwei Schaltsekunden. Die Datumsangabe muss die Regeln des Gregorianischen Kalenders erfüllen, auch wenn sie ein Datum vor Einführung des Gregorianischen Kalenders bezeichnet..

Beispiel

Sie definieren eine TIMESTAMP-Spalte TERMIN wie folgt:

```
termin        TIMESTAMP(3)
```

4.2.5 Verträglichkeit von Datentypen

Bei der Verwendung von Werten in Berechnungen, Prädikaten und Zuweisungen, müssen die Datentypen der beteiligten Operanden verträglich sein.

Zwei Datentypen sind verträglich, wenn sie eine der folgenden Bedingungen erfüllen:

- Beide Datentypen sind CHAR oder VARCHAR.
- Beide Datentypen sind NCHAR oder NVARCHAR.
- Beide Datentypen sind numerisch (SMALLINT, INTEGER, NUMERIC, DECIMAL, RE-AL, DOUBLE PRECISION oder FLOAT).
- Beide Datentypen sind DATE.
- Beide Datentypen sind TIME.
- Beide Datentypen sind TIMESTAMP.

Werte aus verschiedenen Zeichensätzen werden in SESAM/SQL nicht implizit umgewandelt, um sie verträglich zu machen.
Eine Transliteration von Zeichenketten ist mit der Funktion TRANSLATE möglich, siehe Abschnitt „TRANSLATE() - Zeichenkette transliterieren bzw. transcodieren“.

4.3 Werte

Werte werden in SESAM/SQL-Anweisungen zu folgendem Zweck angegeben:

- Spaltenwerte eintragen und ändern (INSERT, MERGE, UPDATE)
- Berechnungen und Vergleiche durchführen (z.B. SELECT-Spaltenauswahl, HAVING-, ON- und WHERE-Suchbedingungen)

SESAM/SQL unterscheidet zwischen NULL-Werten und Nicht-NULL-Werten. Die NichtNULL-Werte sind entsprechend den Datentypen unterteilt.

Es gibt daher folgende Gruppen von Werten:

- NULL-Werte (siehe Abschnitt „NULL-Wert“)
- Alphanumerische Werte (siehe Abschnitt „Zeichenketten“)
- National-Werte (siehe Abschnitt „Zeichenketten“)
- Numerische Werte (siehe Abschnitt „Numerische Werte“)
- Zeitwerte (siehe Abschnitt „Zeitwerte“)

REF-Werte, die im Zusammenhang mit BLOB (Binary Large Object) auftauchen, sind spezielle alphanumerische Werte und dienen der Referenzierung von sogenannten BLOB-Objekten in Basistabellen. Die Definition von REF-Werten in Basistabellen wird im Abschnitt „Spaltendefinition“ beschrieben. Die Struktur und Bearbeitung der BLOB-Objekte wird im Kapitel „SESAM-CLI“ und im „ Basishandbuch“ erklärt.

4.3.1 Literale

Außer für NULL-Werte gibt es für jede Gruppe von Werten entsprechende Literale:

literal : : = { *alphanumerisches_literal* | *national_literal* | *spezial_literal* | *numerisches_literal* | *zeit_literal* }

alphanumerisches_literal

Alphanumerisches Literal (siehe Abschnitt „Alphanumerische Literale“).

national_literal

National-Literal (siehe Abschnitt „National-Literale“).

spezial_literal

Spezial-Literal (siehe Abschnitt „Spezial-Literale“).

numerisches_literal

Numerisches Literal (siehe Abschnitt „Numerische Literale“).

zeit_literal

Zeitliteral (siehe Abschnitt „Zeitliterale“).

4.3.2 Wert angeben

Ein Wert kann auf folgende Arten angegeben werden:

- als Literal
- mit einer Benutzervariablen, wenn die Anweisung **nicht** Bestandteil einer Routine ist (siehe Abschnitt „Benutzervariable“)
- mit einem Parameter (siehe "CREATE PROCEDURE - Prozedur erzeugen") oder einer lokalen Variablen (siehe "COMPOUND - SQL-Anweisungen in gemeinsamem Kontext ausführen"), wenn die Anweisung Bestandteil einer Routine ist
- mit einem Platzhalter „?“ für noch nicht bekannte Werte
  (in einer dynamisch formulierten Anweisung oder Cursorbeschreibung, siehe Abschnitt „Dynamische SQL“)

---

wert `::=`

`{`

literal |

`:` *benutzervariable* `[[INDICATOR] :` *indikatorvariable* `]` |

routinenparameter |

lokale_variable |

?

`}`

---

literal

Alphanumerisches Literal, National-Literal, Spezial-Literal, numerisches Literal oder Zeitliteral.

benutzervariable

Name einer Benutzervariable, die den Wert enthält.

Wenn Sie eine Indikatorvariable angegeben haben und der Wert der Indikatorvariable negativ ist, wird nicht der Wert der Benutzervariable verwendet, sondern der NULL-Wert.

indikatorvariable

Name einer Indikatorvariable zur vorangehenden Benutzervariable. Der Datentyp von *indikatorvariable* ist SMALLINT.

routinenparameter

Name des Parameters einer Routine, der den Wert enthält.

lokale_variable

Name einer lokalen Variablen einer Routine, die den Wert enthält.

? Platzhalter in einer dynamisch formulierten SQL-Anweisung.

4.3.3 Werte für multiple Spalten

Ein Aggregat ist ein Wert für eine multiple Spalte. Ein Aggregat besteht aus einem oder mehreren Elementen, genannt Ausprägungen. Die Anzahl der Ausprägungen muss zwischen 1 und 255 liegen und der Dimension der multiplen Spalte entsprechen. Werte in multiplen Spalten werden multiple Werte genannt; Werte in einfachen Spalten werden einfache Werte (oder kurz: Werte) genannt.

aggregat `::= <{` *wert* `| NULL },...>`

wert

Wert der Ausprägung.

NULL

NULL-Wert für die Ausprägung.

Wenn Sie Elemente der multiplen Spalte mit INSERT oder UPDATE auf den NULL-Wert setzen und nachfolgende Elemente nicht NULL sind, werden die Nicht-NULL-Werte in der multiplen Spalte zu kleineren Positionsnummern verschoben und die NULL-Werte nach allen Nicht-NULL-Werten eingetragen.

Beispiel

Die numerische multiple Spalte FARBTAB mit 3 Elementen kann wie folgt mit INSERT belegt werden:

```
INSERT INTO farbtab (rgb(1..3)) VALUES (<0.88,NULL,0.77>)
```

Die multiple Spalte enthält dann den multiplen Wert:

```
<0.88,0.77,NULL>
```

4.3.4 NULL-Wert

NULL-Werte sind eine Besonderheit von relationalen Datenbanken. Der NULL-Wert bedeutet undefiniert oder unbekannt.

Der NULL-Wert ist von allen anderen Werten verschieden. Er darf nicht mit einer Zeichenkette der Länge 0, dem Leerzeichen oder der numerischen 0 verwechselt werden.

4.3.4.1 Schlüsselwort für den NULL-Wert

Das Schlüsselwort für den NULL-Wert ist das Wort NULL. NULL darf nur beim Einfügen (INSERT), Einfügen /Ändern (MERGE), Ändern (UPDATE), in einem CAST-Ausdruck, in einem CASE-Ausdruck und als Voreinstellung (DEFAULT) bei der Spaltendefinition angeben werden, um einen Spaltenwert auf den NULL-Wert zu setzen.

Beispiel

Sie tragen in die Tabelle ARTIKEL einen Artikel ein, dessen Farbe unbekannt ist:

```
INSERT INTO artikel VALUES (5, 'Ventil', NULL, 1.00, 350, 100)
```

NULL kann auch in Prädikaten (Suchfragen, IF-Anweisung), als Standardwert von lokalen Variablen (in Routinen) sowie in SET- und RETURN-Anweisungen angegeben werden.

4.3.4.2 NULL-Wert in Tabellenspalten

Sie können den NULL-Wert für eine Spalte einer Basistabelle verbieten, indem Sie eine der folgenden Spaltenbedingungen bei der Spaltendefinition angeben:

- Nicht-NULL-Bedingung
- Primärschlüsselbedingung
- Check-Bedingung, die den NULL-Wert verbietet

Wird der NULL-Wert bei der Definition nicht ausgeschlossen, kann die Spalte den NULL-Wert enthalten.

4.3.4.3 NULL-Wert in Funktionen, Ausdrücken und Prädikaten

Das Schlüsselwort NULL darf nicht für Werte in Ausdrücken (ausgenommen CASE- und CAST-Ausdrücke), Funktionen und Prädikaten angegeben werden. Allerdings können Sie Teilausdrücke angeben (zum Beispiel einen Spaltennamen), die den NULL-Wert ergeben.

Kommt in einem Ausdruck der NULL-Wert vor, ist das Ergebnis auch der NULL-Wert.

Kommt in einem Prädikat der NULL-Wert vor, ist das Ergebnis in den meisten Fällen der Wahrheitswert unbestimmt. Allerdings gibt es Ausnahmen wie zum Beispiel das Prädikat IS [NOT] NULL. In Kapitel „Zusammengesetzte Sprachelemente“ ist für jede Funktion, jeden Operator und jedes Prädikat das Ergebnis angegeben, wenn ein Operand der NULL-Wert ist.

4.3.4.4 NULL-Wert in GROUP BY

Bei der Gruppenbildung mit der GROUP BY-Klausel in einer SELECT-Anweisung werden alle Sätze zu einer Gruppe zusammengefasst, die in den gleichen Gruppierungsspalten den NULL-Wert enthalten und in den restlichen Gruppierungsspalten jeweils gleiche Werte haben.

4.3.4.5 NULL-Wert in ORDER BY

Bei der Sortierung von Cursortabellen mit der ORDER BY-Klausel in einer Cursorbeschreibung sind die NULL-Werte kleiner als alle Nicht-NULL-Werte.

4.3.5 Zeichenketten

Zeichenketten sind eine Folge von beliebigen Zeichen in EBCDIC oder Unicode. EBCDIC-Zeichenketten werden als „alphanumerische Werte“, Unicode-Zeichenketten als „National-Werte“ bezeichnet.

In SESAM/SQL werden alphanumerische Literale, National-Literale und Spezial-Literale zur Darstellung von Zeichenketten verwendet.

4.3.5.1 Alphanumerische Literale

Die Syntax für ein alphanumerisches Literal ist wie folgt definiert:

alphanumerisches_literal `::=`

```
{
      '[ zeichen ...]'[ trenner ...'[ zeichen ...]']... |
    X'[ hex hex ]...'[ trenner ...'[ hex hex ]...']...
}
hex ::= 0|1|2|3|4|5|6|7|8|9|a|b|c|d|e|f|A|B|C|D|E|F
```

zeichen

Beliebiges EBCDIC-Zeichen. Soll eine Zeichenkette ein Hochkomma (') enthalten, so müssen Sie das Hochkomma verdoppeln. Das verdoppelte Hochkomma gilt als ein Zeichen (z.B. `'Zeichenketten variabler Länge sind vom Typ ''CHARACTER VARYING'''`).

hex

Eines der Sedezimalzeichen 0-9, A-F oder a-f

Der Datentyp eines alphanumerischen Literals ist CHAR(*länge*). *länge* ist die Anzahl der Zeichen bzw. der Paare von Hexa-Ziffern. Alphanumerische Literale dürfen maximal 256 Zeichen lang sein. Zeichenketten der Länge 0 sind als Literale erlaubt, obwohl es nicht möglich ist, einen Datentyp CHAR(0) zu definieren (siehe Abschnitt „Alphanumerische Datentypen und National-Datentypen“). Der Datentyp ist dann VARCHAR(0).

Die beiden Formen *zeichen* und *hex* vom alphanumerischen Literal können konkateniert werden wie z.B. bei „fünfzig “ (`'f'||X'FD'||'nfzig'`) oder bei Konkatenation mit einem Spezial-Literal (`'Benutzer:'||CURRENT_USER`).

„||“ muss als Operator für die Konkatenation verwendet werden.

> **i** Bei der Konkatenation von Zeichenketten müssen entweder beide Operanden alphanumerisch (CHAR oder VARCHAR) oder beide vom National-Typ (NCHAR oder NVARCHAR) sein, siehe Abschnitt „Verträglichkeit von Datentypen“.

trenner

Trenner, der zwei Teilketten voneinander trennt (siehe Abschnitt „Trenner“). Setzt sich ein alphanumerische Literal aus zwei oder mehreren Teilketten zusammen, müssen benachbarte Teilketten durch einen oder mehrere Trenner getrennt sein. Mindestens einer der Trenner muss ein Übergang zur nächsten Zeile sein.

Das Ergebnis eines aus Teilketten zusammengesetzten alphanumerischen Literals ist die Konkatenation der beteiligten Teilketten, ohne dass dafür der Operator für Konkatenation geschrieben werden muss.

Beispiel

Das folgende alphanumerische Literal ist aus drei Teilketten zusammengesetzt:

```
'Getrennt '     -- erste Teilkette
'von Tisch '    -- zweite Teilkette
'und Bett'      -- dritte Teilkette
```

Das Ergebnis ist die Zeichenkette 'Getrennt von Tisch und Bett'.

4.3.5.2 National-Literale

Die Syntax für ein National-Literal ist wie folgt definiert:

national_literal ::=

{

N'[*zeichen* ...]'[*trenner* ...'[*zeichen* ...]'] ... |
NX'[*4hex* ...]'[*trenner* ...'[*4hex* ...]'] ...|
U&'[*uc-zeichen* ...]'[*trenner...* '[*uc-zeichen* '...] ... [UESCAPE' *esc* ']

}

uc-zeichen ::= { *zeichen* | *esc 4hex* | *esc+ 6hex* | *esc esc* }

zeichen

Ein Unicode-Zeichen, das auch im Zeichensatz EDF03IRV enthalten ist. Soll eine Zeichenkette ein Hochkomma (') enthalten, so müssen Sie das Hochkomma verdoppeln. Das verdoppelte Hochkomma gilt als **ein** Zeichen.

4hex

4hex ist eine Gruppe von 4 aufeinanderfolgenden Sedezimalzeichen und stellt eine UTF-16 Code Unit dar, die im Bereich 0000 bis FFFF liegen muss. (Allerdings sind die UTF-16 Code Units FFFE und FFFF sowie die Code Units im Bereich FDD0 - FDEF sogenannte Noncharacters und dürfen in SESAM/SQL nicht in Literalen verwendet werden, siehe Unicode-Konzept in SESAM/SQL im „Basishandbuch“). Bei der Angabe von *4hex* können die Sedezimalziffern A bis F auch mit Kleinbuchstaben geschreiben werden.

Beispiel

`NX'004100420043'` für die Zeichenkette 'ABC'.

esc 4hex

Sedezimale Darstellung eines Code Points durch das Entwertungszeichen *esc* und (ohne Leerzeichen dazwischen) einem 4-stelligen sedezimalen Wert *4hex* , der im Bereich 0000 bis FFFD liegen muss. Die Angabe *esc* muss genau so geschrieben werden, wie in der UESCAPE-Klausel angegeben. Bei der Angabe von *4hex* können die Sedezimalziffern A bis F auch mit Kleinbuchstaben geschrieben werden..

Beispiel

`U&'\00DF'` für das Zeichen 'ß'

`U&'\0395\03BB\03BB\03B7\03BD\03B9\03BA\03AC hei\00DFt Griechisch'`

für die Zeichenkette „ heißt Griechisch“

esc+ 6hex

Sedezimale Darstellung eines Code Points durch das Entwertungszeichen *esc*, gefolgt von *„+“* und (ohne Leerzeichen dazwischen) einem 6-stelligen sedezimalen Wert *6hex*, der im Bereich 000000 bis 10FFFD liegen muss. (Die Code Points 10FFFE und 10FFFF sowie die Code Points aus den Bereichen 0xFFFE und 0xFFFF (x ist eine Sedezimalzahl) sind sogenannte Noncharacters und dürfen in SESAM/SQL nicht in Literalen verwendet werden, siehe Unicode-Konzept in SESAM/SQL im „ Basishandbuch“). Die Angabe *esc* muss genau so geschrieben werden, wie in der UESCAPE-Klausel angegeben. Bei der Angabe von *6hex* können die Sedezimalziffern A bis F auch mit Kleinbuchstaben geschrieben werden..

Beispiel

`U&'\+0000DF'` für das Zeichen 'ß'.

esc esc

Mit *esc esc* (ohne Leerzeichen dazwischen) können Sie das Entwertungszeichen *esc* entwerten, sodass diese Zeichenfolge ein Zeichen *esc* darstellt.

Beispiel

`U&'\\'` für das Zeichen '\'

UESCAPE `'` *esc* `'`

Angabe eines Entwertungszeichens. *esc* kann ein beliebiges alphanumerisches Zeichen mit Ausnahme von Pluszeichen, Anführungszeichen ("), Hochkomma (') und Leerzeichen sein.

Wenn UESCAPE `'` *esc* `'` nicht angegeben wird, dann wird der Gegenschrägstrich (\) als Standardwert verwendet.

Der Datentyp eines National-Literals ist NCHAR(*cu_länge*). *cu_länge* ist die Anzahl der Code Units (1 Code Unit in UTF-16 = 2 Byte). Die Zeichenketten dürfen maximal 128 Code Units lang sein. Zeichenketten der Länge 0 sind als Literale erlaubt, obwohl es nicht möglich ist, einen Datentyp NCHAR(0) zu definieren (siehe Abschnitt „Zeichenketten“). Der Datentyp ist dann NVARCHAR(0).

Für die Darstellung eines Code Points in UTF-16 wird 1 Code Unit benötigt, außer bei Code Points, die im Bereich 010000 bis 10FFFD enthalten sind. Diese Code Points benötigen zwei Code Units.

Die verschiedenen Formen vom National-Datentyp können konkateniert werden wie z.B. bei „Preis in €“:

```
N'Preis in '||NX'20AC'
```

```
N'Preis in '||U&'\20AC'
```

„||“ muss als Operator für die Konkatenation verwendet werden.

> **i** Bei der Konkatenation von Zeichenketten müssen entweder beide Operanden alphanumerisch (CHAR oder VARCHAR) oder beide vom National-Typ (NCHAR oder NVARCHAR) sein, siehe Abschnitt „Verträglichkeit von Datentypen“.

trenner

Trenner, der zwei Teilketten voneinander trennt (siehe Abschnitt „Trenner“). Setzt sich ein National-Literal aus zwei oder mehreren Teilketten zusammen, müssen benachbarte Teilketten durch einen oder mehrere Trenner getrennt sein. Mindestens einer der Trenner muss ein Übergang zur nächsten Zeile sein.

Das Ergebnis eines aus Teilketten zusammengesetzten Zeichenketten-Literals ist die Konkatenation der beteiligten Teilketten, ohne dass dafür der Operator für Konkatenation geschrieben werden muss.

4.3.5.3 Spezial-Literale

Die Syntax für Spezial-Literale ist wie folgt definiert:

spezial_literal ::=

```
{
   CURRENT_CATALOG |
   CURRENT_ISOLATION_LEVEL |
   CURRENT_REFERENCED_CATALOG |
   CURRENT_SCHEMA |
   [CURRENT_]USER |
   SYSTEM_USER
}
```

CURRENT_CATALOG

Name der mit den SQL-Anweisungen SET CATALOG oder SET SCHEMA voreingestellten Datenbank oder die Zeichenfolge `*IMPLICIT`, wenn keine Datenbank voreingestellt ist.
Das Ergebnis ist eine alphanumerische Zeichenkette vom Typ CHAR(18).

CURRENT_ISOLATION_LEVEL

Isolationslevel der aktuellen Transaktion (wird implizit durch die Anwenderkonfiguration oder explizit durch die SQL-Anweisung SET TRANSACTION */evel* am Beginn einer Transaktion festgelegt). Es bezeichnet nicht das Isolationslevel, das durch das Pragma ISOLATION LEVEL anweisungsspezifisch festgelegt werden kann.
Das Ergebnis ist ein Wert vom Typ INTEGER gemäß folgender Tabelle:.

Ergebnis	Isolationslevel	Konsistenzlevel
8	SERIALIZABLE	4
4	REPEATABLE READ	3
5	READ NO WAIT	1
2	READ COMMITTED	2
1	READ UNCOMMITTED	0

CURRENT_REFERENCED_CATALOG

Name der Datenbank, auf die sich die aktuelle Anweisung bezieht.
Das Ergebnis ist eine alphanumerische Zeichenkette vom Typ CHAR(18).

CURRENT_SCHEMA

Name des mit der SQL-Anweisung SET SCHEMA voreingestellten Schemas oder die Zeichenfolge `*IMPLICIT`, wenn kein Schema voreingestellt ist.
Das Ergebnis ist eine alphanumerische Zeichenkette vom Typ VARCHAR(31).

[CURRENT_]USER

Name des aktuellen Berechtigungsschlüssels.
Das Ergebnis ist eine alphanumerische Zeichenkette vom Typ CHAR(18).

SYSTEM_USER

Name des aktuellen Systembenutzers. Der Name wird zusammengesetzt aus dem Rechnernamen, dem UTM-Anwendungsnamen (bzw. Leerzeichen) und der UTM- bzw. BS2000-Benutzerkennung.
Das Ergebnis ist eine alphanumerische Zeichenkette vom Typ CHAR(24).

4.3.5.4 Zeichenketten verwenden

Ein alphanumerischer Wert oder ein National-Wert kann verwendet werden in:

- Zuweisungen:
  (siehe Abschnitt „Zuweisungsregeln“)
- Funktionen:
  ein alphanumerischer Wert oder ein National-Wert kann in den Mengenfunktionen COUNT(), MIN() und MAX(), in den numerischen Funktionen und in Zeichenkettenfunktionen verwendet werden.
- Konkatenation:
  zwei alphanumerische Werte können zu einem alphanumerischen Wert verbunden werden; zwei National-Werte können zu einem National-Wert verbunden werden. Siehe Abschnitt „Verträglichkeit von Datentypen“.
- Prädikaten:
  ein alphanumerischer Wert oder ein National-Wert kann in Vergleichen mit einem anderen Wert oder mit einer Ergebnisspalte, in Bereichsabfragen, in Elementabfragen und in Mustervergleichen verwendet werden. Alle beteiligten Werte müssen entweder alphanumerische Werte oder National-Werte sein, siehe Abschnitt „Verträglichkeit von Datentypen“. Die Vergleichsregeln sind im Abschnitt „Vergleich von zwei Zeilen“ beschrieben.

Funktionen, Ausdrücke und Prädikate sind ausführlich im Kapitel „Zusammengesetzte Sprachelemente“ beschrieben.

Alphanumerische Literale in der Form X'...' dürfen nicht verwendet werden in SET CATALOG-, SET SCHEMA-, SET SESSION AUTHORIZATION-Anweisungen und auch nicht bei GLOBAL *deskriptor*.

Beispiele

Vor- und Nachname in die Tabelle KUNDE eintragen:

```
INSERT INTO kunde (knr, firma, strasse, plz)
VALUES (100,'Siemens AG','Otto-Hahn-Ring 6',81739)
```

```
INSERT INTO kunde (knr, firma, strasse, plz)
VALUES (100,Siemens AG,"Otto-Hahn-Ring 6",81739)
```

Das ist ein Fehler: Zeichenketten müssen in Hochkommata eingeschlossen werden.

Namen der Tabellen, der Berechtigungsschlüssel und die Privilegien suchen, für die der aktuelle Berechtigungsschlüssel ein Tabellen-Privileg vergeben hat:

```
CREATE VIEW berechtigt AS SELECT TABLE_NAME, GRANTEE, PRIVILEGE_TYPE
FROM INFORMATION_SCHEMA.TABLE_PRIVILEGES WHERE GRANTOR = UTIUNIV
```

Tabelle BUECHER mit der VARCHAR-Spalte TITEL definieren und Werte eintragen:

```
CREATE TABLE buecher (bestellnummer INTEGER, titel VARCHAR(50))
COMMIT WORK
INSERT INTO buecher VALUES (3456, 'Morgen geht die Sonne auf')
INSERT INTO buecher VALUES (5777, 'Kinderlieder')
```

```
INSERT INTO buecher VALUES (7888,
'Das ist ein ueberlanger Titel mit mehr als fuenfzig Zeichen')
```

Der letzte Titel wird nicht eingetragen. Es wird eine Fehlermeldung ausgegeben.

Zusatzinformation zur Kontaktperson Mary Davis in die Tabelle KONTAKT eintragen:

```
UPDATE kontakt set koinfo=('Ms. Davis ist '
'vom 1.8. bis 31.10. '
'beurlaubt') where konr=40
```

Das ist ein Fehler:

```
UPDATE kontakt set koinfo=
('Ms. Davis ist ' 'vom 1.8. bis 31.10. ' 'beurlaubt')
where konr=40
```

Mindestens einer der Trenner zwischen den Teilketten muss ein Übergang zur nächsten Zeile sein.

Vergleiche von Zeichenketten

`' Mai' < ' Maier'` ist wahr

`' Majer' < ' Maier'` ist falsch

Tabelle HANDBUECHER mit den NCHAR-Spalten SPRACHE und TITEL definieren und Werte eintragen:

```
CREATE TABLE handbuecher
(bestellnummer INTEGER, sprache NCHAR(20), titel NCHAR(30))

COMMIT WORK

INSERT INTO handbuecher
VALUES (1001, N'Deutsch', N'Betriebsanleitung'),
       (1002, N'English', N'Operating Manual'),
       (1003, U&'Fran\00E7ais', N'Manuel d''utilisation'),
       (1004, U&'Espa\00F1ol', N'Manual de instrucciones'),
       (1005, N'Italiano', N'Istruzioni per l''uso'),
       (1006, NX'039503BB03BB03B703BD03B903BA03AC',
              NX'039F03B403B703B303AF03B503C2002003BB'
                '03B503B903C403BF03C503C103B303AF03B103C2')
```

Die Spalten SPRACHE und TITEL enthalten dann die folgenden National-Werte:

```
SPRACHE         TITEL
Deutsch         Betriebsanleitung
English         Operating Manual
Français        Manuel d'utilisation
Español         Manual de instrucciones
Italiano        Istruzioni per l'uso
```

4.3.6 Numerische Werte

Numerische Werte sind Ganzzahlen, Festpunktzahlen oder Gleitpunktzahlen.

4.3.6.1 Numerische Literale

Die Syntax für numerische Literale ist wie folgt definiert:

numerisches_literal ::= { *ganzzahl* | *festpunktzahl* | *gleitpunktzahl* }

ganzzahl ::= [{+|-}] *vorzeichenlose_ganzzahl* [.]

festpunktzahl ::= [{+|-}]

{

vorzeichenlose_ganzzahl [.*vorzeichenlose_ganzzahl*] |

vorzeichenlose_ganzzahl. |

.*vorzeichenlose_ganzzahl*

}

gleitpunktzahl ::= *festpunktzahl* E[{+|-}] *vorzeichenlose_ganzzahl*

vorzeichenlose_ganzzahl ::= *ziffer*...

ziffer

Dezimalziffer 0 bis 9.
Ganzzahl- und Festpunktzahl-Literale dürfen maximal 31 Ziffern enthalten.

Der Datentyp des Literals ist Ganzzahl, Festpunktzahl oder Gleitpunktzahl mit der angegebenen Anzahl von Vor- und Nachkommastellen.

4.3.6.2 Numerische Werte verwenden

Ein numerischer Wert kann verwendet werden in:

- Zuweisungen:
  (siehe Abschnitt „Zuweisungsregeln“)
- Mengenfunktionen:
  ein numerischer Wert kann in den Mengenfunktionen AVG(), COUNT(), MIN(), MAX() und SUM() verwendet werden.
- Zeitfunktionen:
  ein numerischer Wert kann in der Zeitfunktion DATE_OF_JULIAN_DAY() verwendet werden.
- Ausdrücken:
  ein numerischer Wert kann in Berechnungen mit den Operatoren +, -, * und / verwendet werden. Alle beteiligten Werte müssen numerisch sein.
- Prädikaten:
  ein numerischer Wert kann in Vergleichen mit einem anderen Wert oder mit einer Ergebnisspalte, in Bereichsabfragen und in Elementabfragen verwendet werden. Alle beteiligten Werte müssen numerisch sein. Die Vergleichsregeln sind im Abschnitt „Vergleich von zwei Zeilen“ beschrieben.

Funktionen, Ausdrücke und Prädikate sind ausführlich im Kapitel „Zusammengesetzte Sprachelemente“ beschrieben.

Beispiele

Die folgenden Beispiele beziehen sich auf die Tabelle LEISTUNG.

Auftragsnummer eintragen:

```
INSERT INTO leistung (lnr, anr, lanz, lsatz)
VALUES (5000, 250, 1, NULL)
```

Auftragsmenge ändern:

```
UPDATE leistung SET lanz=34.75 WHERE lnr=5000
```

Der angegebene Wert wird in eine Ganzzahl konvertiert.

```
UPDATE leistung SET lanz='viel' WHERE lnr=5000
```

Das ist ein Fehler: Der angegebene Wert ist nicht numerisch.

4.3.7 Zeitwerte

SESAM/SQL unterscheidet folgende Arten von Zeitwerten:

- Datum: Ein Datum enthält die Angaben Jahr, Monat und Tag.
- Uhrzeit: Eine Uhrzeit enthält die Angaben Stunden, Minuten, Sekunden und Sekundenbruchteile.
- Zeitstempel: Ein Zeitstempel enthält Datum und Uhrzeit.

4.3.7.1 Zeitliterale

Die Syntax für Zeitliterale ist wie folgt definiert:

```
zeit_literal ::=
{
    DATE ' jahr-monat-tag ' |
    TIME ' stunde:minute:sekunde ' |
    TIMESTAMP' jahr-monat-tag stunde:minute:sekunde '
}
```

DATE

Datum. Der Datentyp des Zeitliterals ist DATE.

TIME

Uhrzeit. Der Datentyp des Zeitliterals ist TIME(3).

TIMESTAMP

Zeitstempel. Der Datentyp des Zeitliterals ist TIMESTAMP(3).

jahr

Vierstellige vorzeichenlose Ganzzahl zwischen 0001 und 9999, die das Jahr angibt.

monat

Zweistellige vorzeichenlose Ganzzahl zwischen 01 und 12, die den Monat angibt.

tag

Zweistellige vorzeichenlose Ganzzahl zwischen 01 und 31 (passend zu Monat und Jahr), die den Tag angibt.

stunde

Zweistellige vorzeichenlose Ganzzahl zwischen 00 und 23, die die Stunde angibt.

minute

Zweistellige vorzeichenlose Ganzzahl zwischen 00 und 59, die die Minute angibt.

sekunde

Vorzeichenlose Festpunktzahl zwischen 00.000 und 60.999, die die Sekunde und die Sekundenbruchteile angibt. Die Angabe für die Sekunden muss zweistellig, die Angabe für die Sekundenbruchteile muss dreistellig sein. Der Wertebereich erlaubt die Angabe von einer Schaltsekunde.

Eine Datumsangabe muss die Regeln des Gregorianischen Kalenders erfüllen, auch wenn sie ein Datum vor Einführung des Gregorianischen Kalenders bezeichnet.

SESAM/SQL erlaubt eine verkürzte Schreibweise, ohne einleitendes Zeit-Schlüsselwort, wenn aus dem Kontext unmittelbar hervorgeht, dass es sich um ein Zeitliteral handelt und nicht um ein alphanumerisches Literal.

Beispiele

Aus der Tabelle AUFTRAG alle Aufträge ausgeben, die vor dem angegebenen Datum erledigt wurden.

```
SELECT * FROM auftrag WHERE fertigist < '2013-01-01'
```

Die Spalte FERTIGIST wurde beim Erzeugen der Tabelle mit Datentyp DATE definiert. Aus dem linken Vergleichsoperanden geht daher unmittelbar hervor, dass es sich bei dem angegebenen Literal um ein Zeitliteral handelt. Auf der rechten Seite kann somit das Schlüsselwort DATE entfallen.

Literal in der SELECT-Liste.

```
SELECT COUNT(*) AS anzahl, '2013-01-01' AS datum FROM auftrag
```

Die Ergebnistabelle enthält eine Zeile mit der Anzahl der Aufträge und mit der Spalte DATUM. Der Datentyp ergibt sich aus dem angegebenen Ausdruck. Also hat die Spalte DATUM den Datentyp CHAR(10).

Um Fehlerquellen auszuschließen, wird empfohlen, Zeitliterale stets mit einleitendem Zeit-Schlüsselwort (DATE, TIME, TIMESTAMP) anzugeben.

> ! **ACHTUNG!** Die Trennzeichen zwischen den Komponentenwerten müssen genau eingehalten werden: Bindestrich „-“ zwischen Jahr, Monat und Tag Leerzeichen „ “ zwischen Tag und Stunde Doppelpunkt „:“ zwischen Stunde, Minute und Sekunde Punkt „.“ zwischen Sekunde und Sekundenbruchteilen.

4.3.7.2 Zeitwerte verwenden

Ein Zeitwert kann verwendet werden in:

- Zuweisungen:
  (siehe Abschnitt „Zuweisungsregeln“)
- Mengenfunktionen:
  ein Zeitwert kann in den Mengenfunktionen COUNT(), MIN() und MAX() verwendet werden.
- Numerische Funktionen:
  ein Datum-Zeitwert kann in der numerischen Funktion JULIAN_DAY_OF_DATE() verwendet werden.
- Prädikaten:
  ein Zeitwert kann in Vergleichen mit einem anderen Wert oder mit einer Ergebnisspalte, in Bereichsabfragen und in Elementabfragen verwendet werden. Alle beteiligten Werte müssen denselben Zeit-Datentyp haben. Die Vergleichsregeln sind im Abschnitt „Vergleich von zwei Zeilen“ beschrieben.
- CAST-Ausdrücken:
  ein Zeitwert kann in einen Wert eines anderen Datentyps umgewandelt werden.

Funktionen und Prädikate sind ausführlich im Kapitel „Zusammengesetzte Sprachelemente“ beschrieben.

Beispiele

Die folgenden Beispiele beziehen sich auf die Tabelle AUFTRAG und auf eine fiktive Tabelle BEISPIEL.

Lieferdatum für Auftrag 300 ändern:

```
UPDATE auftrag SET adatum=DATE'2013-10-06' WHERE anr=300
```

```
UPDATE auftrag SET adatum=DATE'2013-10-6' WHERE anr=300
```

Das ist ein Fehler: Der einstellige Wert 6 für einen Tag ist nicht erlaubt. Richtig wäre 06.

In die Spalte WECKZEIT wird die Zeit 7 Uhr 51 Minuten und 19,77 Sekunden eingetragen:

```
CREATE TABLE beispiel (weckzeit TIME (3), termin TIMESTAMP (3))
```

```
INSERT INTO beispiel (weckzeit) VALUES (TIME'07:51:19.770')
```

In die Spalte TERMIN wird der Zeitstempel 16 Uhr am 24.11.2010 eingetragen:

```
INSERT INTO beispiel (termin) VALUES (TIMESTAMP'2013-10-06 16:00:00.000')
```

```
INSERT INTO beispiel (termin) VALUES (TIMESTAMP'2013-10-06 16:00')
```

Das ist ein Fehler: Es fehlt die Angabe für die Sekunden.

4.4 Zuweisungsregeln

Bei der Zuweisung bzw. Übertragung von Werten müssen der Ausgangsdatentyp und der Zieldatentyp verträglich sein (siehe Abschnitt „Verträglichkeit von Datentypen“).

Weitere Regeln sind abhängig davon, wohin bzw. woher die Werte übertragen werden.

Folgende Fälle werden unterschieden:

- Werte in Tabellenspalten eintragen
- Defaultwerte für Tabellenspalten
- Werte für Platzhalter
- Werte in Benutzervariable oder Deskriptorbereich ablegen
- Werte zwischen Benutzervariablen und Deskriptorbereich übertragen
- Zieldatentyp anpassen mit dem CAST-Operator
- Eingabeparameter für Routinen versorgen
- Werte in Prozedurparameter (Ausgabe) oder lokale Variable eintragen

Die Zuweisungsregeln für die genannten Fälle sind im Folgenden zusammengestellt.

4.4.1 Werte in Tabellenspalten eintragen

Wenn Werte in Tabellenspalten mit INSERT, MERGE oder UPDATE eingetragen bzw. geändert werden, gelten folgende Regeln:

- Atomare Werte und multiple Werte mit Dimension 1 können in atomare Spalten und multiple Spalten (bzw. Teilbereiche) mit Dimension 1 eingetragen werden.
- Multiple Werte mit einer Dimension größer als 1 können in multiple Spalten (bzw. Teilbereiche) mit gleicher Dimension eingetragen werden.
- Abhängig vom Datentyp gelten zusätzlich datentypspezifische Regeln. Diese sind im Folgenden zusammengestellt.

Zeichenketten

Sie können einen alphanumerischen Wert in eine Spalte mit alphanumerischem Datentyp oder einen National-Wert in eine Spalte mit National-Datentyp eintragen. Dabei gilt:

- Ist der Zieldatentyp CHAR oder NCHAR und die Länge des Werts kleiner als die Länge des Zieldatentyps, wird der Wert am Ende mit Leerzeichen ergänzt.
- Ist der Zieldatentyp CHAR oder NCHAR und die Länge des Werts größer als die Länge des Zieldatentyps, wird der Wert am Ende auf die Länge des Zieldatentyps gekürzt. Werden Zeichen entfernt, die keine Leerzeichen sind, wird der Wert nicht eingetragen, sondern eine Fehlermeldung erzeugt.
- Ist der Zieldatentyp VARCHAR oder NVARCHAR und die Länge des Werts größer als die maximale Länge des Zieldatentyps, wird der Wert am Ende auf die maximale Länge des Zieldatentyps gekürzt. Werden Zeichen entfernt, die keine Leerzeichen sind, wird der Wert nicht eingetragen, sondern eine Fehlermeldung erzeugt.

Numerische Werte

Sie können einen numerischen Wert in eine Spalte mit numerischem Datentyp eintragen. Stimmen die numerischen Datentypen nicht überein, wird der Wert in den Datentyp der Spalte konvertiert. Dabei gilt:

- Ist die Nachkommastellenzahl des Werts für den Datentyp der Spalte zu groß, wird der Wert gerundet.
- Ist der Betrag des Werts für den Datentyp der Spalte zu groß, wird der Wert nicht eingetragen, sondern eine Fehlermeldung erzeugt.

Zeitwerte

Sie können einen Zeitwert nur in eine Spalte mit gleichem Datentyp eintragen:

- ein Datum in eine DATE-Spalte
- eine Uhrzeit in eine TIME-Spalte
- einen Zeitstempel in eine TIMESTAMP-Spalte

4.4.2 Defaultwerte für Tabellenspalten

Für den voreingestellten Wert für eine Spalte, der mit der DEFAULT-Klausel bei CREATE TABLE und ALTER TABLE angegeben werden kann, gelten strengere Regeln als beim Eintragen von Werten in Tabellenspalten. Die Regeln gelten auch für die Definition lokaler Variablen (in Routinen). Sie sind in der folgenden Tabelle zusammengestellt:

SQL-Datentyp der Spalte	**möglicher SQL-Defaultwert**
CHAR(*länge*) VARCHAR(*max*)	• alphanumerisches Literal mit Länge <= *länge* bzw. *max* • Spezial-Literal (nur [CURRENT_]USER und SYS-TEM_USER (nur für *länge* bzw. *max* <= 128 empfohlen)) • NULL
NCHAR(*cu_länge*) NVARCHAR(*cu_max*)	• National-Literal mit Länge <= *cu_länge* bzw. *cu_max* • NULL
REF(*tabelle*)	• wie CHAR(237)
DECIMAL(*stellen,bruchteil*) NUMERIC(*stellen,bruchteil*) INTEGER SMALLINT	• Ganz- bzw. Festpunktzahl, die zum Wertebereich der Spalte gehört • NULL
REAL, DOUBLE PRECISION FLOAT(*stellen*)	• numerisches Literal (der Wert wird gerundet, wenn nötig) • NULL
DATE	• Literal vom Datentyp DATE • CURRENT_DATE • NULL
TIME(3)	• Literal vom Datentyp TIME(3) • CURRENT_TIME(3) • NULL
TIMESTAMP(3)	• Literal vom Datentyp TIMESTAMP(3) • CURRENT_TIMESTAMP(3) • NULL

Tabelle 12: Defaultwerte für Tabellenspalten

4.4.3 Werte für Platzhalter

Wenn Werte für Platzhalter in Benutzervariablen oder in einem Deskriptorbereich bereitgestellt werden (EXECUTE... USING, OPEN...USING), gelten folgende Regeln:

- Der Datentyp des Eingabewerts muss mit dem Datentyp des Platzhalters verträglich sein, der sich aus der Position des Platzhalters ergibt (siehe Abschnitt „Regeln für Platzhalter“).
- Werte für atomare Platzhalter und multiple Platzhalter mit Dimension 1 können über eine einfache Benutzervariable, einen Vektor mit einem Element oder über einen Deskriptorbereichseintrag bereitgestellt werden.
- Platzhalter für Aggregate mit einer Dimension $d > 1$ können über einen Vektor mit d Elementen bereitgestellt werden oder über d aufeinanderfolgende Deskriptorbereichseinträge..
- Abhängig vom Datentyp gelten zusätzlich datentypspezifische Regeln. Diese sind im Folgenden zusammengestellt.

Zeichenketten

Sie können für einen Platzhalter mit alphanumerischem Datentyp den Wert aus einer Benutzervariable oder einem Deskriptorbereichseintrag mit alphanumerischem Datentyp verwenden. Für einen Platzhalter mit National-Datentyp können Sie den Wert aus einer Benutzervariable oder einem Deskriptorbereichseintrag mit National-Datentyp verwenden. Dabei gilt:

- Ist der Zieldatentyp CHAR oder NCHAR und die Länge des Werts kleiner als die Länge des Zieldatentyps, wird der Wert am Ende mit Leerzeichen ergänzt.
- Ist der Zieldatentyp CHAR oder NCHAR und die Länge des Werts größer als die Länge des Zieldatentyps, wird der Wert am Ende auf die Länge des Zieldatentyps gekürzt. Werden Zeichen entfernt, die keine Leerzeichen sind, wird eine Warnung erzeugt.
- Ist der Zieldatentyp VARCHAR oder NVARCHAR und die Länge des Werts größer als die maximale Länge des Zieldatentyps, wird der Wert am Ende auf die maximale Länge des Zieldatentyps gekürzt. Werden Zeichen entfernt, die keine Leerzeichen sind, wird eine Warnung erzeugt.

Numerische Werte

Sie können für einen Platzhalter mit numerischem Datentyp den Wert aus einer Benutzervariable oder einem Deskriptorbereichseintrag mit numerischem Datentyp verwenden. Stimmen die numerischen Datentypen nicht überein, wird der Wert in den Zieldatentyp konvertiert. Dabei gilt:

- Ist die Nachkommastellenzahl des Werts für den Zieldatentyp zu groß, wird der Wert gerundet.
- Ist der Betrag des Werts für den Zieldatentyp zu groß, wird der Wert nicht eingetragen, sondern eine Fehlermeldung erzeugt.

Zeitwerte

Sie können für einen Platzhalter mit einem Zeit-Datentyp nur den Wert aus einer Benutzervariable oder einem Deskriptorbereichseintrag mit gleichem Datentyp verwenden:

- ein Datum für einen Platzhalter mit Datentyp DATE
- eine Uhrzeit für einen Platzhalter mit Datentyp TIME
- einen Zeitstempel für einen Platzhalter mit Datentyp TIMESTAMP

4.4.4 Werte in Benutzervariable oder Deskriptorbereich lesen

Wenn Werte aus Tabellenspalten oder Ausgabeparametern einer Routine in einer Benutzervariable oder in einem Deskriptorbereich abgelegt werden (SELECT...INTO, EXECUTE...INTO, FETCH...INTO, INSERT...RETURN INTO, CALL), gelten folgende Regeln:

- Werte aus atomaren Spalten, multiplen Spalten mit Dimension 1 oder Ausgabeparametern einer Prozedur können in einer einfachen Benutzervariable, einem Vektor mit einem Element oder in einem Deskriptorbereichseintrag abgelegt werden.
- Aggregate von multiplen Spalten mit einer Dimension *d* größer als 1 können in einem Vektor mit *d* Elementen oder in *d* aufeinanderfolgenden Deskriptorbereichseinträgen abgelegt werden.
- Ist der zu übertragende Wert ein NULL-Wert, wird die Indikatorvariable bzw. das Deskriptorbereichsfeld INDICATOR auf -1 gesetzt. Ist bei einer Benutzervariable keine Indikatorvariable angegeben, wird eine Fehlermeldung erzeugt.
- Abhängig vom Datentyp gelten zusätzlich datentypspezifische Regeln, die im Folgenden zusammengestellt sind.

Zeichenketten

Sie können einen alphanumerischen Spaltenwert oder einen alphanumerischen Ausgabeparameter einer Prozedur in eine Benutzervariable oder einen Deskriptorbereichseintrag mit alphanumerischem Datentyp lesen. Sie können einen National-Spaltenwert oder einen National-Ausgabeparameter einer Prozedur in eine Benutzervariable oder einen Deskriptorbereichseintrag mit National-Datentyp lesen. Dabei gilt:

- Ist der Zieldatentyp CHAR oder NCHAR und die Länge des Werts kleiner als die Länge des Zieldatentyps, wird der Wert am Ende mit Leerzeichen ergänzt.
- Ist der Zieldatentyp CHAR oder NCHAR und die Länge des Werts größer als die Länge des Zieldatentyps, wird der Wert am Ende auf die Länge des Zieldatentyps gekürzt und eine Warnung erzeugt. Die Indikatorvariable (sofern angegeben) bzw. das Deskriptorbereichsfeld INDICATOR wird auf die Originallänge des Spaltenwerts gesetzt.
- Ist der Zieldatentyp VARCHAR oder NVARCHAR und die Länge des Werts größer als die maximale Länge des Zieldatentyps, wird der Wert am Ende auf die maximale Länge des Zieldatentyps gekürzt und eine Warnung erzeugt. Die Indikatorvariable (sofern angegeben) bzw. das Deskriptorbereichsfeld INDICATOR wird auf die Originallänge des Spaltenwerts gesetzt.

Numerische Werte

Sie können einen numerischen Spaltenwert oder einen numerischen Ausgabeparameter einer Prozedur in eine Benutzervariable oder einen Deskriptorbereichseintrag mit numerischem Datentyp lesen. Stimmen die numerischen Datentypen nicht überein, wird der Wert in den Zieldatentyp konvertiert. Dabei gilt:

- Ist die Nachkommastellenzahl des Werts für den Zieldatentyp zu groß, wird der Wert gerundet.
- Ist der Betrag des Werts für den Zieldatentyp zu groß, wird der Wert nicht eingetragen, sondern eine Fehlermeldung erzeugt.

Zeitwerte

Sie können einen Spaltenwert mit Zeit-Datentyp oder einen Ausgabeparameter einer Prozedur mit Zeit-Datentyp nur in eine Benutzervariable oder einen Deskriptorbereichseintrag mit gleichem Datentyp lesen:

- ein Datum in eine Benutzervariable oder einen Deskriptorbereichseintrag mit Datentyp DATE
- eine Uhrzeit in eine Benutzervariable oder einen Deskriptorbereichseintrag mit Datentyp TIME
- einen Zeitstempel in eine Benutzervariable oder einen Deskriptorbereichseintrag mit Datentyp TIMESTAMP

4.4.5 Werte zwischen Benutzervariablen und Deskriptorbereich übertragen

Bei der Übertragung von Werten zwischen Benutzervariablen und Deskriptorbereich gelten strengere Regeln als bei der Übertragung von Werten zwischen Benutzervariablen (bzw. Deskriptorbereich) und Tabellenspalten:

- Für alle Felder außer NAME und DATA gilt: der SQL-Datentyp der Benutzervariable, in die der Wert eines Feldes abgelegt oder von der der Wert gelesen wird, muss SMALLINT sein.
- Wird der Wert des Felds NAME gelesen, muss die Benutzervariable den SQL-Datentyp CHAR(*n*) oder VARCHAR(*n*) haben, mit *n* >= 128.
- Wird der Wert des Felds DATA in eine Benutzervariable abgelegt oder aus einer Benutzervariable gelesen, muss der SQL-Datentyp der Benutzervariable mit dem durch die Felder TYPE, DATETIME_INTERVAL_CODE, LENGTH, PRECISION und SCALE beschriebenen Datentyp desselben Eintrags übereinstimmen. Abhängig vom Datentyp sind im Folgenden die Regeln zusammengestellt.

Zeichenketten

Bei den SQL-Datentypen CHAR und NCHAR muss die Länge der Benutzervariable gleich dem Wert des Deskriptorfelds LENGTH sein.

Bei den SQL-Datentypen VARCHAR und NVARCHAR muss bei der Übertragung des Werts von der Benutzervariable in den Deskriptorbereich die maximale Länge der Benutzervariable gleich dem Wert des Deskriptorfelds LENGTH sein. Bei der Übertragung des Werts vom Deskriptorbereich in die Benutzervariable muss die maximale Länge der Benutzervariable mindestens so groß sein wie der Wert des Deskriptorfelds LENGTH.

Numerische Werte

Bei SQL-Datentyp NUMERIC oder DECIMAL muss die Gesamtstellenzahl der Benutzervariable gleich dem Wert des Deskriptorfelds PRECISION und die Nachkommastellenzahl gleich dem Wert des Deskriptorfelds SCALE sein.

Zeitwerte

Der SQL-Datentyp der Benutzervariable muss mit dem Datentyp im Deskriptorfeld DATETIME_INTERVAL_CODE übereinstimmen.
Bei den SQL-Datentypen TIME und TIMESTAMP muss das Deskriptorfeld PRECISION den Wert 3 enthalten.

Empfohlene Vorgehensweise

Um nicht für jeden möglichen Datentyp Benutzervariablen definieren zu müssen, empfiehlt sich folgende Vorgehensweise:

1. Datentypbeschreibung für den Wert in DATA mit DESCRIBE im Deskriptorbereichseintrag ablegen
2. Datentyp des Eintrags mit GET DESCRIPTOR abfragen
3. Datentyp des Eintrags mit SET DESCRIPTOR an den Datentyp der Benutzervariable anpassen
4. Wert von DATA aus der oder in die Benutzervariable übertragen.

Beispiel

Folgende dynamische Anweisung wird vorbereitet:

```
SELECT strasse, land, plz, ort FROM kunde WHERE firma='Siemens'
```

Nach DESCRIBE OUTPUT erhalten Sie mit GET DESCRIPTOR folgende Datentypbeschreibungen:

VALUE	REPETITIONS	TYPE	LENGTH	PRECISION	SCALE	entsprechender Datentyp
1	1	1	40	0	0	CHAR(40)
2	1	1	3	0	0	CHAR(3)
3	1	2		5	0	NUMERIC(5,0)
4	1	1	40	0	0	CHAR(40)

Um Benutzervariablen, die mit dem Datentyp CHAR(100) und NUMERIC(15,5) definiert sind, für die Aufnahme der Werte zu verwenden, setzen Sie die Deskriptorbereichsfelder mit SET DESCRIPTOR auf folgende Werte:

VALUE	REPETITIONS	TYPE	LENGTH	PRECISION	SCALE
1	1	1	100		
2	1	1	100		
3	1	2		15	5
4	1	1	100		

Jetzt können Sie die vorbereitete Anweisung mit EXECUTE ausführen. Die Werte werden im Deskriptorbereich abgelegt. STRASSE, LAND, und ORT werden durch Anhängen von Leerzeichen auf die Länge 100 gebracht. PLZ wird mit fünf führenden Nullen und fünf Nachkomma-Nullen ergänzt.
Sie können die Werte mit GET DESCRIPTOR in die entsprechenden Benutzervariablen übertragen und weiterverarbeiten.

4.4.6 Zieldatentyp anpassen mit dem CAST-Operator

In einigen Fällen können Sie mit Hilfe des CAST-Operators (siehe Abschnitt „CAST-Ausdruck“) einen geeigneten Zieldatentyp festlegen, auch wenn SESAM/SQL intern einen anderen Datentyp ermittelt.

Beispiel

Die folgende dynamisch formulierte Anweisung enthält einen zweistelligen Operator mit Platzhalter (?).

```
UPDATE leistung SET mwsatz=0.15+?
```

SESAM/SQL ermittelt den Datentyp des Platzhalters bei diesem zweistelligen Operator aus dem Datentyp des anderen Operators mit NUMERIC(3,2). Wünscht der Anwender einen anderen Datentyp, etwa NUMERIC(4,2), so kann er diesen über den CAST-Operator festlegen:

```
UPDATE leistung SET mwsatz=CAST(? AS NUMERIC(4,2))
```

4.4.7 Eingabeparameter für Routinen versorgen

Wenn Sie in einer CALL-Anweisung (Prozeduraufruf) oder bei Aufruf einer User Defined Function (UDF) den Eingabeparametern für die Routine Werte zuweisen, dann gelten datentyp-spezifische Regeln. Diese sind im Folgenden zusammengestellt.

Zeichenketten

Sie können einen alphanumerischen Wert einem Eingabeparameter mit alphanumerischem Datentyp oder einen National-Wert einem Eingabeparameter mit National-Datentyp zuweisen. Dabei gilt:

- Ist der Zieldatentyp CHAR oder NCHAR und die Länge des Werts kleiner als die Länge des Zieldatentyps, wird der Wert am Ende mit Leerzeichen ergänzt.
- Ist der Zieldatentyp CHAR oder NCHAR und die Länge des Werts größer als die Länge des Zieldatentyps, wird der Wert am Ende auf die Länge des Zieldatentyps gekürzt. Werden Zeichen entfernt, die keine Leerzeichen sind, wird der Wert nicht eingetragen, sondern eine Fehlermeldung erzeugt.
- Ist der Zieldatentyp VARCHAR oder NVARCHAR und die Länge des Werts größer als die maximale Länge des Zieldatentyps, wird der Wert am Ende auf die maximale Länge des Zieldatentyps gekürzt. Werden Zeichen entfernt, die keine Leerzeichen sind, wird der Wert nicht eingetragen, sondern eine Fehlermeldung erzeugt.

Numerische Werte

Sie können einen numerischen Wert einem Eingabeparameter mit numerischem Datentyp zuweisen. Stimmen die numerischen Datentypen nicht überein, wird der Wert in den Datentyp des Eingabeparameters konvertiert. Dabei gilt:

- Ist die Nachkommastellenzahl des Werts für den Datentyp des Eingabeparameters zu groß, wird der Wert gerundet.
- Ist der Betrag des Werts für den Datentyp des Eingabeparameters zu groß, wird der Wert nicht eingetragen, sondern eine Fehlermeldung erzeugt.

Zeitwerte

Sie können einen Zeitwert nur einem Eingabeparameters mit gleichem Datentyp zuweisen:

- ein Datum in einen Eingabeparameter mit Datentyp DATE
- eine Uhrzeit in einen Eingabeparameter mit Datentyp TIME
- einen Zeitstempel in einen Eingabeparameter mit Datentyp TIMESTAMP

4.4.8 Werte in Prozedurparameter (Ausgabe) oder lokale Variable eintragen

Wenn Sie in einer Prozedur den Ausgabeparametern oder in einer Routine den lokalen Variablen oder dem Funktionswert einer UDF Werte zuweisen (SET, RETURN, SELECT...INTO, FETCH...INTO, INSERT...RETURN INTO), dann gelten datentyp-spezifische Regeln. Diese sind im Folgenden zusammengestellt.

Zeichenketten

Sie können einen alphanumerischen Wert in einen Ausgabeparameter oder eine lokale Variable mit alphanumerischem Datentyp eintragen. Sie können einen National-Wert in einen Ausgabeparameter oder eine lokale Variable mit National-Datentyp eintragen. Dabei gilt:

- Ist der Zieldatentyp CHAR oder NCHAR und die Länge des Werts kleiner als die Länge des Zieldatentyps, wird der Wert am Ende mit Leerzeichen ergänzt.
- Ist der Zieldatentyp CHAR oder NCHAR und die Länge des Werts größer als die Länge des Zieldatentyps, wird der Wert am Ende auf die Länge des Zieldatentyps gekürzt und eine Warnung erzeugt.
- Ist der Zieldatentyp VARCHAR oder NVARCHAR und die Länge des Werts größer als die maximale Länge des Zieldatentyps, wird der Wert am Ende auf die maximale Länge des Zieldatentyps gekürzt und eine Warnung erzeugt.

Numerische Werte

Sie können einen numerischen Wert in einen Ausgabeparameter oder eine lokale Variable mit numerischem Datentyp eintragen. Stimmen die numerischen Datentypen nicht überein, wird der Wert in den Zieldatentyp konvertiert. Dabei gilt:

- Ist die Nachkommastellenzahl des Werts für den Zieldatentyp zu groß, wird der Wert gerundet.
- Ist der Betrag des Werts für den Zieldatentyp zu groß, wird der Wert nicht eingetragen, sondern eine Fehlermeldung erzeugt.

Zeitwerte

Sie können einen Wert mit Zeit-Datentyp nur in einen Ausgabeparameter oder eine lokale Variable mit gleichem Datentyp eintragen:

- ein Datum in einen Ausgabeparameter oder eine lokale Variable mit Datentyp DATE
- eine Uhrzeit in einen Ausgabeparameter oder eine lokale Variable mit Datentyp TIME
- einen Zeitstempel in einen Ausgabeparameter oder eine lokale Variable mit Datentyp TIMESTAMP

5 Zusammengesetzte Sprachelemente

Dieses Kapitel beschreibt die zusammengesetzten Sprachelemente, die in SESAM/SQL-Anweisungen vorkommen können. Es gliedert sich in folgende Teile:

- Ausdruck
- Funktion
- Prädikat
- Suchbedingung
- CASE-Ausdruck
- CAST-Ausdruck
- Integritätsbedingung
- Spaltendefinition

Die genannten Sprachelemente setzen sich aus Grundelementen wie Namen, Literalen und anderen Sprachelementen zusammen. Sie sind in der Reihenfolge beschrieben, in der sie aufeinander aufbauen.

5.1 Ausdruck

Die Auswertung eines Ausdrucks ergibt einen Wert oder liefert eine Tabelle (Tabellenfunktionen)..

Ausdrücke können vorkommen in:

- Spaltenauswahl (SELECT-Ausdruck, SELECT-Anweisung)
- Prädikaten von Suchbedingungen (z.B. WHERE-, HAVING-Klausel)
- Zuweisungen (INSERT-, MERGE- oder UPDATE-Anweisung)
- SQL-Anweisungen, die in Routinen verwendet werden (z.B. CASE-Anweisung)

Ein Ausdruck besteht aus Operanden und eventuell Operatoren. Die Operatoren werden auf die Ergebnisse der Operanden angewendet.

Das Ergebnis der Auswertung ist ein alphanumerischer Wert, ein National-Wert, ein numerischer Wert oder ein Zeitwert.

Eine Tabellenfunktion liefert als Ergebnis eine Tabelle.

Die Reihenfolge der Auswertung der Operanden ist nicht festgelegt. In bestimmten Fällen wird ein Teilausdruck nicht berechnet, wenn er für die Berechnung des Gesamtergebnisses nicht benötigt wird.

Bei Auswertung eines Ausdrucks wird eine darin enthaltene Funktion ausgeführt und durch den berechneten Wert oder die gelieferte Tabelle ersetzt.

Syntaxdiagramm eines Ausdrucks:

ausdruck ::=

```
{
  wert |
  [ tabelle .] {
                  spalte |
                    { spalte ( posnr ) | spalte[ posnr ] } |
                    { spalte ( min..max ) | spalte[ min..max ] }
                } |
  funktion |
  unterabfrage |
  un_op ausdruck |
  ausdruck bin_op ausdruck |
  case_ausdruck |
  cast_ausdruck |
  ( ausdruck )
}
```

spalte ::= *einf_name*

posnr ::= *vorzeichenlose_ganzzahl*
min ::= *vorzeichenlose_ganzzahl*
max ::= *vorzeichenlose_ganzzahl*

un_op ::= { + | - }

bin_op ::= { * | / | + | = | || }

wert

Alphanumerischer Wert, National-Wert, numerischer Wert oder Zeitwert (siehe Abschnitt „Werte“).

tabelle

Name einer Tabelle, die *spalte* enthält. Statt des Tabellennamens geben Sie den Korrelationsnamen an, wenn für die Tabelle ein Korrelationsname definiert ist.

spalte

Name einer Spalte, aus der die Werte genommen werden.

> Die eckigen Klammern, die in der Syntax kursiv gedruckt sind, sind Sonderzeichen und müssen in der Anweisung angegeben werden.

posnr

Vorzeichenlose Ganzzahl.

Der Wert wird aus dem (*posnr*-*sp* $_{min}$+1)-ten Spaltenelement der multiplen Spalte *spalte* genommen und kann wie ein einfacher Wert verwendet werden.

Ist *spalte* keine multiple Spalte, *posnr* kleiner als *sp* $_{min}$ oder *posnr* größer als *sp* $_{max}$, erfolgt eine Fehlermeldung.

sp $_{min}$ bzw. *sp* $_{max}$ ist die kleinste bzw. größte Positionsnummer der multiplen Spalte.

min **..** *max*

Vorzeichenlose Ganzzahlen.

Der Wert ist das Aggregat aus den Spaltenelementen (*min*-*sp* $_{min}$+1) bis (*max*- *sp* $_{min}$+1) der multiplen Spalte *spalte*.

Ist *spalte* keine multiple Spalte, *min* nicht kleiner als *max*, *min* kleiner als *sp* $_{min}$ oder *max* größer als *sp* $_{max}$, erfolgt eine Fehlermeldung.

sp $_{min}$ bzw. *sp* $_{max}$ ist die kleinste bzw. größte Positionsnummer der multiplen Spalte.

posnr bzw. *min* **..** *max* nicht angegeben:

spalte darf keine multiple Spalte sein.

funktion

Funktion (siehe Abschnitt „Funktion“).

unterabfrage

Unterabfrage (siehe Abschnitt „Unterabfrage“), die genau einen Wert liefert.

un_op

Einstelliger Operator, der das Vorzeichen setzt. *ausdruck* muss numerisch sein und darf kein multipler Wert mit Dimension > 1 sein.

+ Der Wert bleibt gleich.

- Der Wert wird negiert.

bin_op

Zweistelliger Operator. Die beiden Operandenausdrücke dürfen keine multiplen Werte mit Dimension > 1 sein.

a ***** *b*

Multiplikation von *a* mit *b*.

Die Ausdrücke *a* und *b* müssen numerisch sein.

Wenn *a* und *b* Ganz- oder Festpunktzahlen sind, ist das Ergebnis eine Ganz- oder Festpunktzahl mit der Gesamtstellenzahl g_a+g_b, höchstens jedoch 31.

Die Nachkommastellenzahl beträgt n_a+n_b, höchstens jedoch 31.

g_a und g_b sind die Gesamtstellenzahlen von *a* und *b*.

n_a und n_b sind die Nachkommastellenzahlen von *a* und *b*.

Wenn *a* oder *b* Gleitpunktzahlen sind, ist das Ergebnis eine Gleitpunktzahl mit der Gesamtstellenzahl 24 Bit bei REAL bzw. 56 Bit bei DOUBLE PRECISION.

Ist der Betrag des Ergebnisses für den Ergebnisdatentyp zu groß, erfolgt eine Fehlermeldung. Ist die Gesamtstellenzahl zu groß, wird das Ergebnis gerundet.

a **/** *b*

Division von *a* durch *b*.

Die Ausdrücke *a* und *b* müssen numerisch sein.

Wenn *a* und *b* Ganz- oder Festpunktzahlen sind, ist das Ergebnis eine Ganz- oder Festpunktzahl mit der Gesamtstellenzahl 31.

Die Nachkommastellenzahl beträgt $31-v_a-n_b$, mindestens jedoch 0.

v_a ist die Vorkommastellenzahl von *a*. n_b ist die Nachkommastellenzahl von *b*.

Wenn *a* oder *b* Gleitpunktzahlen sind, ist das Ergebnis eine Gleitpunktzahl mit der Gesamtstellenzahl 24 Bit bei REAL bzw. 56 Bit bei DOUBLE PRECISION.

Ist der Betrag des Ergebnisses für den Ergebnisdatentyp zu groß, oder ist der Wert von *b* gleich 0, erfolgt eine Fehlermeldung. Ist die Gesamtstellenzahl zu groß, wird das Ergebnis gerundet.

a **+** *b*

Addition von *a* und *b*.

Die Ausdrücke *a* und *b* müssen numerisch sein.

Wenn *a* und *b* Ganz- oder Festpunktzahlen sind, ist das Ergebnis eine Ganz- oder Festpunktzahl mit der Gesamtstellenzahl $v_{max}+n_{max}+1$, höchstens jedoch 31.

Die Nachkommastellenzahl beträgt n_{max}.

v_{max} ist die größere der beiden Vorkommastellenzahlen von *a* und *b*.

n_{max} ist die größere der beiden Nachkommastellenzahlen von *a* und *b*.

Wenn *a* oder *b* Gleitpunktzahlen sind, ist das Ergebnis eine Gleitpunktzahl mit der Gesamtstellenzahl 24 Bit bei REAL bzw. 56 Bit bei DOUBLE PRECISION.

Ist der Betrag des Ergebnisses für den Ergebnisdatentyp zu groß, erfolgt eine Fehlermeldung. Ist die Gesamtstellenzahl zu groß, wird das Ergebnis gerundet.

a **-** *b*

Subtraktion *b* von *a*.

Die Ausdrücke *a* und *b* müssen numerisch sein.

Wenn *a* und *b* Ganz- oder Festpunktzahlen sind, ist das Ergebnis eine Ganz- oder Festpunktzahl mit der Gesamtstellenzahl $v_{max}+n_{max}+1$, höchstens jedoch 31.

Die Nachkommastellenzahl beträgt n_{max}.

v_{max} ist die größere der beiden Vorkommastellenzahlen von *a* und *b*.

n_{max} ist die größere der beiden Nachkommastellenzahlen von *a* und *b*.

Wenn *a* oder *b* Gleitpunktzahlen sind, ist das Ergebnis eine Gleitpunktzahl mit der Gesamtstellenzahl 24 Bit bei REAL bzw. 56 Bit bei DOUBLE PRECISION.

Ist der Betrag des Ergebnisses für den Ergebnisdatentyp zu groß, erfolgt eine Fehlermeldung. Ist die Gesamtstellenzahl zu groß, wird das Ergebnis gerundet.

a **||** *b*

Konkatenation von *a* mit *b*.

Die Ausdrücke *a* und *b* müssen entweder alphanumerische Werte oder NationalWerte ergeben.

Wenn *a* und *b* vom Datentyp CHAR sind, ist das Ergebnis vom Datentyp CHAR mit der Länge $/_a+/_b$ (in Zeichen), und diese Summe darf nicht größer als 256 sein.

Wenn *a* und *b* vom Datentyp NCHAR sind, ist das Ergebnis vom Datentyp NCHAR mit der Länge $/_a+/_b$ (in Code Units), und diese Summe darf nicht größer als 128 sein.

Wenn *a* oder *b* vom Datentyp VARCHAR sind, ist das Ergebnis vom Datentyp VAR-CHAR mit der Länge $/_a+/_b$ (in Zeichen), höchstens jedoch 32000.

Wenn *a* oder *b* vom Datentyp NVARCHAR sind, ist das Ergebnis vom Datentyp NV-ARCHAR mit der Länge $/_a+/_b$ (in Code Units), höchstens jedoch 16000.

$/_a$ und $/_b$ sind die Längen von *a* und *b*.

Wenn das Ergebnis vom Typ CHAR länger als 256 Zeichen oder das Ergebnis vom Typ NCHAR länger als 128 Zeichen ist, dann erfolgt eine Fehlermeldung.

Wenn das Ergebnis vom Typ VARCHAR länger als 32000 Zeichen ist, wird die Zeichenkette rechts auf die Länge 32000 gekürzt und wenn das Ergebnis vom Typ NV-ARCHAR länger als 16000 Zeichen ist, wird die Zeichenkette rechts auf die Länge 16000 gekürzt. Wenn Zeichen entfernt werden, die keine Leerzeichen sind, erfolgt eine Fehlermeldung.

case_ausdruck

CASE-Ausdruck (siehe Abschnitt „CASE-Ausdruck“).

cast_ausdruck

CAST-Ausdruck (siehe Abschnitt „CAST-Ausdruck“).

Prioritäten

- Klammerausdrücke haben die höchste Priorität.
- Die einstelligen Operatoren haben Vorrang vor den zweistelligen.
- Die multiplikativen Operatoren * und / haben höhere Priorität als die additiven Operatoren + und -.
- Die multiplikativen Operatoren haben gleiche Priorität.
- Die additiven Operatoren haben gleiche Priorität.
- Operatoren gleicher Priorität werden von links nach rechts angewendet.
- Wenn *ausdruck* ein einfacher Name *einf_name* ist, für den es im Gültigkeitsbereich sowohl eine Spalte als auch einen Routinenparameter oder eine lokale Variable mit diesem Namen gibt, dann wird der Routinenparameter bzw. die lokale Variable verwendet.

> *Empfehlung* Die Namen von Routinenparametern und lokale Variablen sollten sich (z.B. durch Vergabe eines Präfixes wie `par_` oder `var_`) von Spaltennamen unterscheiden.

5.2 Funktion

Eine Funktion berechnet einen Wert oder liefert eine Tabelle (Tabellenfunktion). Funktionen können innerhalb von Ausdrücken vorkommen. Bei Auswertung eines Ausdrucks wird eine darin enthaltene Funktion ausgeführt und durch den berechneten Wert oder die gelieferte Tabelle ersetzt. Die SESAM/SQL-Funktionen sind in folgende Gruppen eingeteilt:

- Zeitfunktionen
- Zeichenkettenfunktionen
- Numerische Funktionen
- Mengenfunktionen
- Tabellenfunktionen
- kryptografische Funktionen
- User Defined Functions (UDF)

---

funktion : : =

{

zeitfunktion |

zeichenkettenfunktion |

numerische_funktion |

mengenfunktion |

tabellenfunktion |

kryptofunktion |

user_defined_function

}

---

zeitfunktion

Zeitfunktion (siehe Abschnitt „Zeitfunktionen“).

zeichenkettenfunktion

Zeichenkettenfunktion (siehe Abschnitt „Zeichenkettenfunktionen“).

numerische_funktion

Numerische Funktion (siehe Abschnitt „Numerische Funktionen“).

mengenfunktion

Mengenfunktion (siehe Abschnitt „Mengenfunktionen“).

tabellenfunktion

Tabellenfunktion (siehe Abschnitt „Tabellenfunktionen“).

kryptofunktion

Kryptografische Funktion (siehe Abschnitt „Kryptografische Funktionen“).

user_defined_function

User Defined Function (siehe Abschnitt „User Defined Functions (UDF)“).

5.2.1 Zeitfunktionen

Zeitfunktionen ermitteln folgende Daten

- das aktuelle Datum (CURRENT_DATE)
- die aktuelle Uhrzeit (CURRENT_TIME(3) oder LOCALTIME(3))
- einen Zeitstempel mit aktuellem Datum und aktueller Uhrzeit (CURRENT_TIMESTAMP(3) oder LOCALTIMESTAMP(3))
- das einem ganzzahligen Wert entsprechende Datum (DATE_OF_JULIAN_DAY) (siehe auch die inverse Funktion JULIAN_DAY_OF_DATE auf "JULIAN_DAY_OF_DATE() - Datum umwandeln").

LOCALTIMESTAMP(3) und CURRENT_TIMESTAMP(3) bzw. LOCALTIME(3) und CURRENT_TIME(3) sind in SESAM/SQL äquivalent.

---

zeitfunktion `::=`

```
{
   CURRENT_DATE |
   CURRENT_TIME(3) |
   LOCALTIME(3) |
   CURRENT_TIMESTAMP(3) |
   LOCALTIMESTAMP(3) |
   DATE_OF_JULIAN_DAY( ausdruck )
}
```

---

ausdruck

> Ganzzahliger numerischer Ausdruck, den SESAM/SQL als Julianische Tagesnummer interpretiert. *ausdruck* darf kein multipler Wert mit Dimension > 1 sein.

Kommen die Zeitfunktionen CURRENT_DATE, CURRENT_TIME(3), LOCALTIME(3), CURRENT_TIMESTAMP(3) und LOCALTIMESTAMP(3) innerhalb einer Anweisung mehrmals vor, werden diese simultan ausgewertet. Das gilt auch für alle Zeitfunktionen, die als Folge der Anweisung ausgewertet werden:

- Zeitfunktionen in der DEFAULT-Klausel der Spaltendefinition, wenn die Voreinstellung verwendet wird.
- Zeitfunktionen, die im SELECT-Ausdruck eines View vorkommen, wenn der View angesprochen wird.

Alle zurückgelieferten Werte enthalten gleiches Datum und/oder gleiche Uhrzeit. Daher können Sie Zeitfunktionen nicht dazu verwenden, Ausführungszeitpunkte innerhalb der Anweisung festzustellen.

Zeitfunktionen in dynamisch formulierten Anweisungen und in Cursorbeschreibungen werden zum Zeitpunkt der EXECUTE-, EXECUTE IMMEDIATE- bzw. OPEN-Anweisung ausgewertet.

5.2.2 Zeichenkettenfunktionen

Zeichenkettenfunktionen führen folgende Aufgaben aus:

- sie extrahieren Teilzeichenketten (SUBSTRING)
- sie transliterieren alphanumerische Zeichenketten in National-Zeichenketten und umgekehrt (TRANSLATE)
- sie transcodieren National-Zeichenketten von UTFE nach UTF-16 und umgekehrt (TRANSLATE)
- sie entfernen führende oder nachgestellte Zeichen von Zeichenketten (TRIM)
- sie wandeln Groß- in Kleinbuchstaben bzw. Klein- in Großbuchstaben um (LOWER, UPPER)
- sie konvertieren einen Wert von beliebigem Datentyp in die interne Darstellung (als alphanumerische Zeichenkette oder hexadezimal) und umgekehrt
  (HEX_OF_VALUE, VALUE_OF_HEX, REP_OF_VALUE, VALUE_OF_REP)
- sie liefern für National-Zeichenketten das Collation-Element gemäß der Default Unicode Translation Table (COLLATE)
- sie bringen National-Zeichenketten in Normalform (NORMALIZE)

---

zeichenkettenfunktion ::=

```
{
   SUBSTRING ( ausdruck FROM startposition [FOR teilkettenlänge ][USING CODE_UNITS] ) |
   TRANSLATE ( ausdruck USING [[ catalog .]INFORMATION_SCHEMA.] transname

                      [DEFAULT zeichen ] [ ,länge ] ) |
   TRIM ([[LEADING |TRAILING | BOTH] [ zeichen ] FROM] ausdruck ) |
   LOWER ( ausdruck ) |
   UPPER ( ausdruck ) |
   HEX_OF_VALUE ( ausdruck2 ) |
   VALUE_OF_HEX ( ausdruck3 , datentyp ) |
   REP_OF_VALUE ( ausdruck2 ) |
   VALUE_OF_REP ( ausdruck3 , datentyp ) |
   COLLATE ( ausdruck USING { DUCET_WITH_VARS | DUCET_NO_VARS } [ ,länge ] ) |
   NORMALIZE ( ausdruck [,NFC | NFD [, länge ]] )
}
```

zeichen ::= *ausdruck*

länge ::= *vorzeichenlose_ganzzahl*

---

ausdruck

Alphanumerischer Ausdruck oder National-Ausdruck. Seine Auswertung ergibt entweder eine alphanumerische Zeichenkette (Datentyp CHAR oder VARCHAR) oder eine National-Zeichenkette (Datentyp NCHAR oder NVARCHAR). *ausdruck* darf kein multipler Wert mit Dimension > 1 sein. Siehe auch Abschnitt „Verträglichkeit von Datentypen“ .

Einschränkungen sind in der Funktionsbeschreibung jeweils bei der entsprechenden Funktion beschrieben.

ausdruck2

Ausdruck von beliebigem Datentyp. Die interne Darstellung dieses Werts wird als alphanumerische Zeichenkette bzw. in Hexadezimalform geliefert.
ausdruck2 darf kein multipler Wert mit Dimension > 1 sein.

ausdruck3

Alphanumerischer Ausdruck, der die interne Darstellung eines Werts vom Typ *datentyp* ist. Dieser Wert ist das Ergebnis.
ausdruck3 darf kein multipler Wert mit Dimension > 1 sein.

startposition

Ganzzahliger numerischer Ausdruck für die Position des Anfangs der Teilzeichenkette.

teilkettenlänge

Ganzzahliger numerischer Ausdruck für die Länge der Teilzeichenkette.

datentyp

Datentyp des Ergebnisses.

länge

Maximale Länge der Ergebnis-Zeichenkette.

5.2.3 Numerische Funktionen

Numerische Funktionen erfüllen unterschiedliche Zwecke:

- ABS(), CEILING(), FLOOR(), MOD(), SIGN() und TRUNC() führen die entsprechenden mathematischen Funktionen auf den angegebenen numerischen Ausdrücken aus.
- CHARACTER_LENGTH( ), OCTET_LENGTH( ) und POSITION( ) berechnen die Anzahl der Bytes oder Code Units einer Zeichenkette bzw. die Position einer Zeichenkette innerhalb einer anderen Zeichenkette.
- JULIAN_DAY_OF_DATE( ) wandelt ein Datum in einen ganzzahligen Wert um.
- EXTRACT() extrahiert bestimmte Bestandteile eines Zeitwertes.

Bei Auswertung einer numerischen Funktion wird ein numerischer Wert zurückgeliefert.

---

numerische_funktion ::=

```
{
    ABS ( ausdruck ) |
    CEIL[ING] ( ausdruck ) |
    FLOOR ( ausdruck ) |
    MOD ( dividend,divisor ) |
    SIGN ( ausdruck ) |
    TRUNC ( ausdruck ) |
    { CHAR_LENGTH | CHARACTER_LENGTH } ( ausdruck [USING { CODE_UNITS | OCTETS }]) |
    OCTET_LENGTH ( ausdruck ) |
    POSITION ( ausdruck IN ausdruck [USING CODE_UNITS]) |
    JULIAN_DAY_OF_DATE ( ausdruck ) |
    EXTRACT ( bestandteil FROM ausdruck )
}
```

---

ausdruck

Bei ABS(), CEILING(), FLOOR(), MOD(), SIGN() und TRUNC(): numerischer Ausdruck.

Bei EXTRACT() und JULIAN_DAY_OF_DATE( ): Zeitwerte-Ausdruck.

Sonst: alphanumerischer Ausdruck oder National-Ausdruck.

ausdruck darf kein multipler Wert mit Dimension > 1 sein.

5.2.4 Mengenfunktionen

Mengenfunktionen bestimmen Durchschnitt, Anzahl, Maximum, Minimum und Summe einer Menge von Werten bzw. die Anzahl der Sätze einer Ergebnistabelle.

mengenfunktion ::= { *operator* ([ALL | DISTINCT] *ausdruck*) | COUNT(*) }

operator ::= {AVG | COUNT | MAX | MIN | SUM }

ausdruck

Ausdruck, der die Werte in der Menge bestimmt (siehe Abschnitt „Ausdruck“).

Für jede Mengenfunktion außer COUNT(*) ist festgelegt, welche Datentypen *ausdruck* haben darf. In der Funktionsbeschreibung sind für jede Funktion die erlaubten Datentypen angegeben.

Für *ausdruck* gelten folgende Einschränkungen:

- *ausdruck* darf keine multiple Spalte enthalten.
- *ausdruck* darf keine Mengenfunktion enthalten.
- *ausdruck* darf keine Unterabfrage enthalten.
- Wenn ein Spaltenname in *ausdruck* eine Spalte eines übergeordneten Abfrage-Ausdrucks angibt (Außenreferenz), darf *ausdruck* nur diesen Spaltennamen enthalten.

  Die Mengenfunktion muss dann eine der beiden folgenden Bedingungen erfüllen:

  - Die Mengenfunktion ist in einer SELECT-Liste enthalten.
  - Die Mengenfunktion ist in einer Unterabfrage einer HAVING-Klausel enthalten. Der Spaltenname muss eine Spalte des SELECT-Ausdrucks angeben, der die HAVING-Klausel enthält.

> **i** Die Mengenfunktionen MIN() und MAX() beziehen sich auf die Menge aller Werte einer Spalte in einer Tabelle. Sie unterscheiden sich dadurch von einem CASE-Ausdruck mit MIN / MAX (siehe "CASE-Ausdruck mit MIN / MAX"), der sich auf unterschiedliche Ausdrücke bezieht.

Mengenfunktion berechnen

Bei allen Mengenfunktionen außer COUNT(*) bestimmt der als Funktionsargument angegebene Ausdruck die Menge von Werten, auf die die Mengenfunktion angewendet wird.

Enthält der SELECT-Ausdruck bzw. die SELECT-Anweisung, in dem (der) die Mengenfunktion vorkommt, keine GROUP BY-Klausel, wird der Argumentausdruck auf alle Sätze (bzw. auf die Sätze, die die WHERE-Klausel erfüllen) der Tabelle angewendet, auf die sich die Spaltenangaben im Argumentausdruck beziehen. Enthält der Argumentausdruck keine Spaltenangabe, wird der Argumentausdruck auf alle Sätze der Tabelle des SELECT-Ausdrucks angewendet. Ergebnis ist eine einspaltige Tabelle.

Wenn diese Tabelle NULL-Werte enthält, werden diese vor Anwendung der Mengenfunktion entfernt. Dabei wird eine Warnung erzeugt.
Ist in der Mengenfunktion DISTINCT angegeben, werden nur verschiedene Werte berücksichtigt, d.h., wenn Werte mehrmals in der Tabelle vorkommen, werden die Duplikate vor Anwendung der Mengenfunktion entfernt.
Die Mengenfunktion wird dann auf die verbleibenden Werte der einspaltigen Tabelle angewendet und liefert als Ergebnis genau einen Wert.

Enthält der zugehörige SELECT-Ausdruck (bzw. die SELECT-Anweisung) eine GROUP BY-Klausel, wird die Mengenfunktion wie beschrieben für jede Gruppe gesondert berechnet und liefert pro Gruppe genau einen Wert.

Beispiele

Ohne GROUP BY: Der folgende Ausdruck berechnet die Summe des dreifachen Preises der Artikel aus der Tabelle ARTIKEL:

```
SELECT SUM (3*preis) FROM artikel
```

Zur Berechnung des Ausdrucks wird zunächst der Argumentausdruck `3*preis` auf alle Sätze der Tabelle ARTIKEL angewendet. Das ergibt folgende Ergebnisspalte:

```
2101.50
 690,00
 450.00
 450.00
 120.00
 120.00
 180.00
  15.00
  15.00
  30.00
   3.00
   3.30
   2.25
```

Als Summe ergibt sich der Wert 41880.05.

Mit GROUP BY: Der folgende Ausdruck berechnet aus der Tabelle LAGER den Gesamtbestand pro Lagerort:

```
SELECT ort, SUM (bestand) FROM lager GROUP BY ort
```

Zur Berechnung des Ausdrucks werden zunächst die Bestände nach Lagerorten gruppiert:

```
ort               bestand
Hauptlager             2
                       1
                      10
                      10
                       3
                       3
                       1
                      15
                       8
                       6
                      11
                     120
                     248
Teilelager             9
                       6
                       3
                     200
                     180
                      47
```

Anschließend werden für jeden Lagerort die Bestände summiert.

```
ort
Hauptlager           438
Teilelager           445
```

5.2.5 Tabellenfunktionen

Tabellenfunktionen erzeugen Tabellen, deren Inhalt von den Aufrufparametern abhängt oder aus externe Datenquellen, z.B. Dateien, stammt.

tabellenfunktion ::= {

    CSV ([FILE] *datei* DELIMITER *delimiter* [QUOTE *quote*] [ESCAPE *escape*], *datentyp* ,...) |
    DEE [()]

}

Die Tabellenfunktionen sind in "CSV() - BS2000-Datei als Tabelle lesen" und "DEE() - Tabelle ohne Spalten" beschrieben.

5.2.6 Kryptografische Funktionen

Die Funktionen ENCRYPT() und DECRYPT() dienen der Verschlüsselung und Entschlüsselung von einzelnen Werten. Sensitive Daten werden durch die Verschlüsselung vor unbefugtem Zugriff geschützt. Nur die Benutzer, die den sogenannten „Schlüssel“ („key“) kennen, können die Daten entschlüsseln.

Die Funktionen REP_OF_VALUE() und VALUE_OF_REP() können verwendet werden, um mehrere Werte gemeinsam zu verschlüsseln und ihn wieder zu entschlüsseln.

Einführende Informationen zum Zugriffsschutz durch Datenverschlüsselung in SESAM/SQL finden Sie im „ Basishandbuch“.

---

kryptofunktion `::= { ENCRYPT (` *ausdruck* `,` *schlüssel* `) | DECRYPT (` *ausdruck2* `,` *schlüssel* `,` *datentyp* `) }`

schlüssel `::=` *ausdruck*

---

ausdruck

Ausdruck, dessen Wert verschlüsselt werden soll.*ausdruck* darf kein multipler Wert mit Dimension > 1 sein.

ausdruck2

Alphanumerischer Ausdruck, dessen Wert entschlüsselt werden soll. *ausdruck2* darf kein multipler Wert mit Dimension > 1 sein.

schlüssel

Schlüssel zur Ver- und Entschlüsselung.

datentyp

Datentyp des entschlüsselten Wertes. *datentyp* darf kein Aggregat (siehe "Werte für multiple Spalten") sein.

Anwendungshinweise

Da der Verschlüsselungsalgorithmus AES (siehe „ Basishandbuch“), so wie er in SESAM/SQL verwendet wird, Blöcke von 16 Zeichen bearbeitet, ist die Länge des Ausgabewerts stets ein Vielfaches von 16 Zeichen. Wenn sich zwei Eingabewerte auch nur in einem Bit unterscheiden, dann unterscheiden sich ihre verschlüsselten Werte in allen Zeichen.

Verschlüsselte Werte können auf Gleichheit oder Ungleichheit verglichen werden. Sie sind genau dann gleich oder ungleich, wenn die unverschlüsselten Werte gleich oder ungleich sind. Dabei müssen die unverschlüsselten Werte denselben Datentyp haben. Bei Zeichenketten müssen die unverschlüsselten Werte auch gleich lang sein.

> **i** Die Vergleiche 01 = 1.0 und 'abc' = 'abc ' liefern aber jeweils den Wahrheitswert TRUE, die Verschlüsselungen dieser vier Werte sind aber alle unterschiedlich.

Andere Vergleiche (z.B. mit < oder <=) verschlüsselter Werte liefern Resultate, die nichts mit den entsprechenden Vergleichen der unverschlüsselten Werte zu tun haben. Auch die Prädikate BETWEEN und LIKE sind für verschlüsselte Daten nicht sinnvoll. Das gilt auch für eien Sortierung mit ORDER BY.

Die Verschlüsselung eines NULL-Wertes liefert den NULL-Wert des entsprechenden Datentyps. Ob ein Wert ein NULL-Wert ist oder nicht, ist daher auch bei Verschlüsselung keine vertrauliche Information. Die Verschlüsselung einer Zeichenkette der Länge 0 liefert hingegen eine Zeichenkette der Länge 16. Ohne Kenntnis des Schlüssels können sie nicht von den Verschlüsselungen von Zeichenketten mit 1 bis 14 alphanumerischen Zeichen unterschieden werden.

ACHTUNG! Verschlüsselte Werte können normalerweise nicht mehr entschlüsselt werden, wenn sie verkürzt oder verlängert werden (auch wenn die neue Länge ein Vielfaches von 16 ist). Eine Spalte mit verschlüsselten Werten sollte also z.B. nicht den Datentyp CHAR(20) haben, weil dann jedem verschluüsselten Wert 4 Leerzeichen angefügt wuerden. Vor dem Entschlüsseln müßten diese Leerzeichen wieder entfernt werden.

5.2.7 User Defined Functions (UDF)

UDFs haben einen fast identischen Funktionsumfang wie Prozeduren. Sie sind detailliert im Kapitel „Routinen“ beschrieben.

Der aktuelle Berechtigungsschlüssel muss das EXECUTE-Privileg für die UDF besitzen.

CHECK-Bedingungen dürfen keine UDF enthalten.

---

user_defined_function `::=` *einf_routinenname argumente*

argumente `::= ([`*ausdruck* `[{,`*ausdruck*`}...]])`

---

einf_routinenname

Name der auszuführenden UDF. Der einfache UDF-Name kann durch einen Datenbank und Schemanamen qualifiziert werden.

`([`*ausdruck* `[{,`*ausdruck*`}...]])`

Liste der Argumente. Die Anzahl der Argumente muss mit der Anzahl der UDF-Parameter aus der UDF-Definition übereinstimmen. Die Argumente müssen in ihrer Reihenfolge mit den Parametern korrespondieren. Wenn kein Parameter für die UDF definiert ist, dann besteht die Liste nur aus den runden Klammern.

Dem n-ten Parameter wird vor Ausführung der UDF der Wert des n-ten Arguments zugewiesen.

Der Datentyp des n-ten Arguments muss mit dem Datentyp des n-ten Parameters verträglich sein. Für Eingabeparameter siehe die Hinweise im Abschnitt „Eingabeparameter für Routinen versorgen“.

5.2.8 Alphabetischer Nachschlageteil Funktionen

In den folgenden Abschnitten sind die Funktionen in alphabetischer Reihenfolge beschrieben..

5.2.8.1 ABS() - Absolutwert

Funktionsgruppe: Numerische Funktion

ABS() bestimmt den Absolutwert eines numerischen Wertes.

ABS (*ausdruck*)

---

ausdruck

Numerischer Ausdruck.
ausdruck darf kein multipler Wert mit Dimension > 1 sein.

Ergebnis

Wenn *ausdruck* den NULL-Wert ergibt, ist das Ergebnis der NULL-Wert.

Sonst: der Absolutwert von *ausdruck*.
Also der Wert von *ausdruck*, wenn *ausdruck* positiv ist, sonst der Wert von -(*ausdruck*).

Datentyp: wie *ausdruck*

Beispiele

ABS `(3,14)` ergibt den Wert 3,14.

ABS `(-3,14)` ergibt den Wert 3,14.

5.2.8.2 AVG() - Arithmetisches Mittel

Funktionsgruppe: Mengenfunktion

AVG() berechnet das arithmetische Mittel aus einer Menge von numerischen Werten. NULL-Werte werden nicht berücksichtigt.

```
AVG ([ALL | DISTINCT] ausdruck )
```

ALL

Alle Werte werden berücksichtigt, auch solche die doppelt vorkommen.

DISTINCT

Nur verschiedene Werte werden berücksichtigt. Duplikate werden ignoriert.

ausdruck

Numerischer Ausdruck (Einschränkungen siehe Abschnitt „Mengenfunktionen“).

Ergebnis

Ist die Menge der aus *ausdruck* berechneten Werte leer, ist das Ergebnis bzw. das Ergebnis für diese Gruppe der NULL-Wert.

Sonst:

Ohne GROUP BY-Klausel:

Arithmetisches Mittel der Werte in der aus *ausdruck* berechneten Menge (siehe „Mengenfunktion berechnen“).

Mit GROUP BY-Klausel:

Pro Gruppe das arithmetische Mittel der Werte für diese Gruppe.

Datentyp: wie *ausdruck* mit folgender Stellenzahl:

- Ganzzahl oder Festpunktzahl:

  Die Gesamtstellenzahl ist 31, die Nachkommastellenzahl ist 31-*g*+*n*.
  g und *n* sind die Gesamtstellenzahl und Nachkommastellenzahl von *ausdruck*.
- Gleitpunktzahl:

  Die Gesamtstellenzahl entspricht 21 Binärstellen bei REAL und 53 Binärstellen bei DOUBLE PRECISION.

Beispiele

SELECT ohne GROUP BY:

Durchschnittssatz der Leistungen in der Tabelle LEISTUNG der Beispieldatenbank berechnen (Ergebnis: 783,33):

```
SELECT AVG(lsatz) FROM leistung
```

Wenn Sie in die Tabelle einen Satz eintragen, der in der Spalte LSATZ den NULL-Wert enthält, ändert sich das Ergebnis nicht.

SELECT mit GROUP BY:

Für jede Auftragsnummer wird der Durchschnittssatz berechnet:

```
SELECT anr, AVG(lsatz) FROM leistung GROUP BY anr
   anr
   200      1025
   211      662.5
   250      662.5
```

5.2.8.3 CEILING() - kleinste ganze Zahl größer als der Wert

Funktionsgruppe: Numerische Funktion

CEILING() („Runden zur Decke“) bestimmt die kleinste ganze Zahl, die größer oder gleich ist als der angegebene numerische Wert. CEILING() rundet bei nicht-ganzzahligen numerischen Werten stets auf.

```
CEIL[ING] ( ausdruck )
```

ausdruck

Festpunktwert vom Typ NUMERIC(p,s) oder DECIMAL(p,s) wenn die Zahl der Nachkommastellen s größer als 0 ist, sonst numerischer Ausdruck.
ausdruck darf kein multipler Wert mit Dimension > 1 sein.

Ergebnis

Wenn *ausdruck* den NULL-Wert ergibt, ist das Ergebnis der NULL-Wert.

Sonst:

Die kleinste ganze Zahl, die größer ist als der angegebene numerische Wert.

Datentyp: NUMERIC(q+1,0) oder DECIMAL(q+1,0) mit q=MIN(31,p+1) wenn die Zahl der Nachkommastellen s größer als 0 war, sonst wie *ausdruck*.

Beispiele

`CEILING (3,14)` ergibt den Wert 4.

`CEILING (-3,14)` ergibt den Wert -3.

`CEILING (10,14)` ergibt den Wert 11.

5.2.8.4 CHAR_LENGTH() - Zeichenkettenlänge bestimmen

Funktionsgruppe: Numerische Funktion

CHAR_LENGTH() bzw. CHARACTER_LENGTH() bestimmt die Anzahl der Bytes oder Code Units einer Zeichenkette.

```
{ CHAR_LENGTH | CHARACTER_LENGTH }( ausdruck [USING [CODE_UNITS | OCTETS]])
```

ausdruck

Alphanumerischer Ausdruck oder National-Ausdruck. Seine Auswertung ergibt entweder eine alphanumerische Zeichenkette (Datentyp CHAR oder VARCHAR) oder eine National-Zeichenkette (Datentyp NCHAR oder NVARCHAR).

Bei den alphanumerischen Datentypen CHAR bzw. VARCHAR liefern CHAR_LENGTH( ) und OCTET_LENGTH( ) (siehe Abschnitt „OCTET_LENGTH() - Zeichenkettenlänge bestimmen“) dieselben Werte, weil jedes Zeichen in genau einem Byte (Octet) dargestellt ist.

Bei den National-Datentypen NCHAR und NVARCHAR kann die Länge entweder in Bytes (Funktionen OCTET_LENGTH und CHAR_LENGTH ... USING OCTETS) bestimmt werden, oder in UTF-16 Code Units (Funktion CHAR_LENGTH ... USING CODE_UNITS). Eine Code Unit in UTF-16 = 2 Byte. Die Anzahl der Unicode-Zeichen in einer National-Zeichenkette kann kleiner sein als der Anzahl Code Units in UTF-16, da manche Unicode-Zeichen durch zwei aufeinanderfolgenden Code Units in UTF-16 dargestellt werden (surrogate pairs).

ausdruck darf kein multipler Wert mit Dimension > 1 sein. Siehe auch Abschnitt „Verträglichkeit von Datentypen “.

USING CODE_UNITS

Die Länge soll in Code Units ausgegeben werden.
In den Datentypen CHAR und VARCHAR ist 1 Code Unit = 1 Byte.
In den Datentypen NCHAR und NVARCHAR ist 1 Code Unit = 2 Byte.

USING OCTETS

Die Länge soll in Bytes ausgegeben werden.
In den Datentypen CHAR und VARCHAR ist 1 Zeichen = 1 Byte.
In den Datentypen NCHAR und NVARCHAR ist 1 Zeichen = 1 oder 2 Code Units = 2 bzw. 4 Byte.

Ergebnis

Enthält die Zeichenkette den NULL-Wert, ist das Ergebnis der NULL-Wert.

Sonst:

Das Ergebnis ist die Anzahl der Bytes oder Code Units in der Zeichenkette.

Datentyp: INTEGER

Beispiele

Anzahl der Bytes (Zeichen) bestimmen, die in der alphanumerischen Zeichenkette 'fuer' enthalten sind (Ergebnis: 4).

```
CHAR_LENGTH ('fuer') USING OCTETS
```

Anzahl der Bytes bestimmen, die in der National-Zeichenkette 'nur' enthalten sind (Ergebnis: 6).

```
CHAR_LENGTH (N'nur') USING OCTETS
```

Anzahl der Code Units bestimmen, die in der National-Zeichenkette 'nur' enthalten sind (Ergebnis: 3).

```
CHAR_LENGTH (N'nur') USING CODE_UNITS
```

Anzahl der Code Units bestimmen, die in der National-Zeichenkette 'München' enthalten sind (Ergebnis: 7).

```
CHAR_LENGTH (U&'M\00FCnchen')
```

5.2.8.5 COLLATE() - Collation-Element für National-Zeichenketten ermitteln

Funktionsgruppe: Zeichenkettenfunktion

COLLATE() liefert für National-Zeichenketten das Collation-Element (Sortierelement) gemäß der Default Unicode Collation Table (DUCET), siehe „ Basishandbuch“.

Nicht zugeordnete Code Points und Code Points > U+2FFF werden ignoriert. Collation-Elemente reichen bis zur Vergleichsebene 3, die Ebene 4 wird ignoriert.

`COLLATE ( `*ausdruck*` USING [[ `*catalog*` .]INFORMATION_SCHEMA.] [ `*collation*` , `*länge*` ])`

collation ::= `{ DUCET_WITH_VARS | DUCET_NO_VARS }`

länge `::=` *vorzeichenlose_ganzzahl*

---

ausdruck

National-Ausdruck.

DUCET_WITH_VARS | DUCET_NO_VARS

Name der anzuwendenden Collation (Sortierreihenfolge).
In SESAM/SQL sind alle Collation-Namen vordefiniert. Es sind die Namen, die auch im BS2000-Softwareprodukt SORT für die Sortierung von Zeichenketten definiert sind.

Bei DUCET_NO_VARS werden die variablen Collation-Elemente, z.B. Leerzeichen, Satzzeichen und zu ignorierende Folgezeichen, ignoriert.

Bei DUCET_WITH_VARS werden sie berücksichtigt.
Die Zeichenketten U&'dieselbe' und U&'die selben' sortieren sich bei DUCET_NO_VARS in dieser Reihenfolge, bei DUCET_WITH_VARS in umgekehrter Reihenfolge.

Die Sortierreihenfolge kann durch einen Datenbanknamen und den Schema-Namen INFORMATION_SCHEMA qualifiziert werden, ansonsten wird das INFORMATION_SCHEMA der voreingestellten Datenbank angenommen.

länge

Maximale Länge des Collation-Elements mit 1 <= *länge* <= 32000.

Länge nicht angegeben:
Das Ergebnis kann abhängig von *ausdruck* bis zu 32000 Bytes lang werden.

Ergebnis

Wenn *ausdruck* den NULL-Wert ergibt, dann ist das Ergebnis der NULL-Wert.

Sonst:

Das Ergebnis ist das Collation-Element für *ausdruck* gemäß der Default Unicode Collation Table (DUCET) in der Länge *n* = 4 + 6 * (Länge von *ausdruck* in Code Units), mit *n* <= 32000.

Wenn die Länge des Collation-Elements größer ist als die angegebene oder die maximale Länge, dann wird die Funktion mit SQLSTATE abgebrochen.

Datentyp: VARCHAR(*n*)

Beispiele

Ausgabe einer Liste von Kundenkontakten, sortiert nach der Default Unicode Collation Table unter Berücksichtigung der variablen Collation-Elemente:

```
UNLOAD ONLINE DATA KONTAKT (NACHNAME,VORNAME,ANREDE,KOTELEFON,FUNKTION) -
    INTO FILE 'DAT.070.C.DUCETWITHVARS' -
    CSV_FORMAT DELIMITER ';' QUOTE '"' ESCAPE '\' EBCDIC -
    ORDER BY COLLATE(TRANSLATE(NACHNAME USING EDF041 DEFAULT N'?') -
                    USING DUCET_WITH_VARS,200) -
             ASC, -
             COLLATE(TRANSLATE(VORNAME USING EDF041 DEFAULT N'?') -
                    USING DUCET_WITH_VARS,200) -
             ASC
```

Ausgabe des Collation-Elements für einen Buchstaben:

```
HEX_OF_VALUE(COLLATE(TRANSLATE ('A' USING EDF041) USING DUCET_NO_VARS))

0E33000020000800
```

5.2.8.6 COUNT(*) - Tabellensätze zählen

Funktionsgruppe: Mengenfunktion

COUNT(*) zählt die Sätze einer Tabelle. Sätze, die NULL-Werte enthalten, werden mitgezählt.

```
COUNT (*)
```

Ergebnis

Ohne GROUP BY-Klausel:

Anzahl der Sätze der Ergebnistabelle des zugehörigen SELECT-Ausdrucks (bzw. der zugehörigen SELECT-Anweisung). Doppelte Sätze und Sätze, die nur NULL-Werte enthalten, werden mitgezählt.

Mit GROUP BY-Klausel:

Pro Gruppe in der Ergebnistabelle die Anzahl der Sätze in dieser Gruppe.

Datentyp: DECIMAL(31,0)

Beispiele

SELECT ohne GROUP BY:

Aus der Tabelle KUNDE abfragen, wie viele Kunden in München wohnen (Ergebnis: 3):

```
SELECT COUNT(*) FROM kunde WHERE ort='Muenchen'
```

SELECT mit GROUP BY:

Die Kunden getrennt nach Orten zählen:

```
SELECT ort, COUNT(*) FROM kunde GROUP BY ort
   ort
   Berlin                    1
   Bern 33                   1
   Hannover                  1
   Moenchengladbach          1
   Muenchen                  3
   New York, NY              1
```

5.2.8.7 COUNT() - Elemente zählen

Funktionsgruppe: Mengenfunktion

COUNT() zählt die Elemente einer Menge von Werten. NULL-Werte werden nicht mitgezählt.

```
COUNT ([ALL | DISTINCT ] ausdruck )
```

ALL

Alle Werte werden berücksichtigt, auch solche, die doppelt vorkommen.

DISTINCT

Nur verschiedene Werte werden berücksichtigt. Duplikate werden ignoriert.

ausdruck

Numerischer Ausdruck, alphanumerischer Ausdruck, National-Ausdruck oder Zeitwerte-Ausdruck (Einschränkungen siehe Abschnitt „Mengenfunktionen“).

Ergebnis

Ohne GROUP BY-Klausel:

Anzahl der Werte in der aus *ausdruck* berechneten Menge (siehe „Mengenfunktion berechnen“).

Mit GROUP BY-Klausel:

Pro Gruppe die Anzahl der Werte für diese Gruppe.

Datentyp: DECIMAL(31,0)

Beispiele

SELECT ohne GROUP BY:

Aus der Tabelle LEISTUNG die Anzahl verschiedener Leistungsbeschreibungen bestimmen (Ergebnis: 7):

```
SELECT COUNT(DISTINCT ltext) FROM leistung
```

SELECT mit GROUP BY:

Für jede Auftragsnummer die Anzahl verschiedener Leistungen zählen:

```
SELECT anr, COUNT(DISTINCT ltext) FROM leistung
   GROUP BY anr
   anr
   200                2
   211                4
   260                2
```

5.2.8.8 CSV() - BS2000-Datei als Tabelle lesen

Funktionsgruppe: Tabellenfunktion

Mit der Tabellenfunktion CSV() können Sie den Inhalt einer BS2000-Datei als „read-only“ Tabelle in beliebigen SQL-Anweisungen verwenden.

Für die Darstellung von SQL-Tabellen in Dateien wird dabei das CSV-Format verwendet (CSV: Comma Separated Values). Das ist ein standardisiertes Format für den plattform-unabhängigen Austausch von tabellarischen Daten, siehe „Format von CSV-Dateien“. Die Datei enthält die Folge von Zeilen der Tabelle, wobei jede Zeile der Reihe nach ihre Spaltenwerte als Zeichenketten enthält. Solche Dateien können mit vielen Softwareprodukten (z.B. mit Microsoft EXCEL) erzeugt werden.

```
CSV ([FILE] datei DELIMITER delimiter [QUOTE quote ] [ESCAPE escape ], datentyp ,...)
```

FILE *datei*

Name der Eingabedatei. *datei* muss als alphanumerisches Literal angegeben werden.

Die Eingabedatei muss eine SAM-Datei sein.
Liegt die Eingabedatei nicht in der Kennung des DBH, so muss die DBH-Kennung eine Leseberechtigung für diese Datei haben. Andernfalls kann der DBH nicht auf die Eingabedatei zugreifen.

Wenn ein Lesekennwort für die Datei benötigt wird, so muss dieses in der Form `?PASSWORD=<` *kennwort* `>` an den BS2000-Dateinamen angefügt werden,
z.B. `':8OSH:$ABC.MYFILE?PASSWORD=C''ABCD'''`.

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

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

DELIMITER *delimiter*

Trennzeichen (DELIMITER-Zeichen) zwischen den Spaltenwerten der CSV-Datei. Ein DELIMITER-Zeichen kann auch Bestandteil eines Wertes sein, siehe die nachfolgenden Beschreibungen zu *quote* und *escape*. *delimiter* muss als alphanumerisches Literal der Länge 1 angegeben werden.

QUOTE *quote*

Anführungszeichen (QUOTE-Zeichen), in das die Spaltenwerte in der CSV-Datei eingeschlossen sein können. Diese QUOTE-Zeichen gehören nicht zum Spaltenwert. Ein QUOTE-Zeichen im Spaltenwert muss in der CSV-Datei verdoppelt werden. Wenn ein Wert in QUOTE-Zeichen eingeschlossen ist, dann kann er auch NEWLINE-Zeichen (die nicht als Zeilenwechsel interpretiert werden) oder DELIMITER-Zeichen enthalten. Ein Wert, der lediglich aus einem öffnenden und schließenden QUOTE-Zeichen besteht, wird als Wert der Länge 0

interpretiert.
quote muss als alphanumerisches Literal der Länge 1 angegeben werden.Wenn QUOTE nicht angegeben ist, dann können die Spaltenwerte in der CSV-Datei nicht in QUOTE-Zeichen eingeschlossen werden.

ESCAPE *escape*

Entwertungszeichen (ESCAPE-Zeichen), mit dem ESCAPE-Folgen, bestehend aus zwei Zeichen, in der Eingabedatei beginnen.
Mit ESCAPE-Folgen können DELIMITER-Zeichen, QUOTE-Zeichen und ESCAPE-Zeichen als Teil eines Spaltenwerts geschrieben und NEWLINE-Zeichen als Trenner zwischen zwei Eingabezeilen ignoriert werden.
escape muss als alphanumerisches Literal der Länge 1 angegeben werden.Wenn ESCAPE nicht angegeben ist, dann können in der CSV-Datei keine ESACPE-Folgen verwendet werden.

> *delimiter*, *quote* und *escape* müssen unterschiedliche Zeichen sein.

datentyp,...

Datentypen der einzelnen Spalten der Tabelle, die aus der CSV-Datei gelesen wird.Jeder *datentyp* muss ein Datentyp CHARACTER(n) (mit 1 <= n <= 256) oder CHARACTER VARYING(n) (mit 1 <= n <= 32000) sein.

Ergebnis

Eine Tabelle mit ebensovielen Spalten, wie Datentypen angegeben sind, jeweils mit dem angegebenen Datentyp.

Beispiel

Es wird eine neue Basistabelle LEIST_ENCR aufgebaut. Ihr Inhalt wird einer CSV-Datei entnommen.

```
INSERT INTO leist_encr (letext, lesatz_encr)
SELECT a,b
   FROM TABLE(CSV(FILE 'out.leistung.070' DELIMITER ':',
              CHAR(25),VARCHAR(16)))
   AS t(a,b)
```

Format von CSV-Dateien

Das CSV-Format (CSV: Comma Separated Values) ist ein standardisiertes Format für den plattform-unabhängigen Austausch von tabellarischen Daten. Solche Dateien können mit vielen Softwareprodukten (z.B. mit Microsoft EXCEL) erzeugt und bearbeitet werden.

Tabellen werden in CSV-Dateien als Folge von Zeilen dargestellt, wobei die Zeilen in der Datei durch (ein oder mehrere) NEWLINE-Zeichen (Zeilenwechsel) getrennt werden. Auch der Übergang zum nächsten Satz in einer SAM-Datei ist ein solcher Zeilenwechsel, obwohl er kein EBCDIC-Zeichen ist. In einem Satz einer SAM-Datei können mehrere Zeilen, getrennt durch ein NEWLINE-Zeichen, stehen. Auch vor der ersten und nach der letzten Zeile dürfen Zeilenwechsel vorkommen.

In einer Zeile werden die einzelnen Spaltenwerte durch ein einzelnes DELIMITER-Zeichen getrennt. Auch nach dem letzten Spaltenwert einer Zeile darf ein DELIMITER-Zeichen stehen.

Für die Darstellung der einzelnen Spaltenwerte in jeder Zeile gibt es zwei Formen: die einzelnen Zeichen der Spalte können in QUOTE-Zeichen eingeschlossen werden oder nicht. Im ersten Fall können die Spaltenwerte auch NEWLINE- und das DELIMITER-Zeichen enthalten. Allerdings muss ein QUOTE-Zeichen im Spaltenwert verdoppelt werden (sonst beendet es den Spaltenwert). Spaltenwerte in QUOTE-Zeichen können nur dann verwendet werden, wenn in der CSV-Funktion der Operand QUOTE angegeben ist.Beginnt ein Spaltenwert nicht mit dem QUOTE-Zeichen (oder ist der Operand QUOTE nicht in der CSV-Funktion angegeben), dann endet der Spaltenwert vor dem nächsten DELIMITER- oder NEWLINE-Zeichen.

In SESAM/SQL können Sie auch ein ESCAPE-Zeichen definieren. Mit dem ESCAPE-Zeichen können Sie innerhalb des Spaltenwerts ESCAPE-Folgen verwenden, die wie folgt interpretiert werden:

Escape-Folge	**wird interpretiert als**
escape newline	„kein Zeichen“
escape delimiter	ein DELIMITER-Zeichen
escape quote	ein QUOTE-Zeichen
escape escape	ein ESCAPE-Zeichen

ESCAPE-Folgen sind auch in Spaltenwerten erlaubt, die in QUOTE-Zeichen eingeschlossen sind. Insbesondere ESCAPE NEWLINE ist nützlich, denn wenn am Ende eines SAM-Satzes ein ESCAPE-Zeichen steht, dann gilt damit die Zeile als noch nicht beendet und wird mit dem folgenden SAM-Satz fortgesetzt. Die Zeilen in einer CSV-Datei können damit länger sein als ein Satz in einer SAM-Datei von BS2000.

Wenn beim Lesen der CSV-Datei Fehler auftreten oder ein Verstoss gegen das CSV-Format festgestellt wird (z.B. bei Dateiende in einem Spaltenwert, der mit dem QUOTE-Zeichen beginnt, aber nicht damit endet), so wird dies mit Fehlercode angezeigt.

Anmerkung zu NEWLINE-Zeichen

Im CSV-Format werden vier EBCDIC Control Characters als NEWLINE-Zeichen interpretiert::

X'04' ist das Zeichen NEXT LINE

X'0D' ist das Zeichen CARRIAGE RETURN. Sein ASCII-Äquivalent wird in einigen Macintosh-Systemen als Zeilenwechsel verwendet.

X'15' ist das Zeichen LINE FEED. Sein ASCII-Äquivalent wird in POSIX- und LINUX-Systemen als Zeilenwechsel verwendet. In EBCDIC-Systemen von IBM wird es als NEXT LINE oder als LINE FEED verwendet. Das ASCII-Äquivalent von X'0D15' wird in Windows-Systemen als Zeichenfolge für (einen) Zeilenwechsel verwendet.

X'25' ist das Zeichen PRIVATE USE TWO. Es wird aber in EBCDIC-Systemen von IBM als LINE FEED oder als NEXT LINE und in IBM z/OS Unix System Services als Zeilenwechsel verwendet.

Das CSV-Format akzeptiert alle diese Control Characters (ebenso wie den Übergang zum nächsten Satz einer SAM-Datei) als Zeilenwechsel.

Syntax einer CSV-Datei

Eine syntaktische Darstellung des Formats einer CSV-Datei finden Sie in "Syntaxübersicht CSV-Datei".

Interpretation von CSV-Dateien als SQL-Tabelle

In der CSV-Funktion werden die Anzahl der zu lesenden Spalten und ihre Datentypen angegeben. Diese Spalten entsprechen den Spaltenwerten in der CSV-Datei in derselben Reihenfolge. Wenn eine Zeile der CSV-Datei weniger Spaltenwerte enthält, dann werden NULL-Werte ergänzt. Wenn eine Zeile der CSV-Datei mehr Spaltenwerte enthält, dann werden die überschüssigen Spaltenwerte ignoriert.

Eine Zeile in einer CSV-Datei muss wenigstens ein Zeichen enthalten. Mehrere aufeinanderfolgende Zeilenwechsel werden wie ein Zeilenwechsel behandelt.

Ein leerer Spaltenwert (z.B. zwischen zwei aufeinanderfolgenden DELIMITER-Zeichen) wird als NULL-Wert interpretiert.

Ein Spaltenwert, der länger ist als die (maximale) Länge des Datentyps der Spalte, wird gekürzt. Es wird eine Warnung ausgegeben.

Wenn der Datentyp der Spalte CHARACTER(n), aber der Spaltenwert kürzer als n ist, dann wird wird der Spaltenwert am Ende mit Leerzeichen (X'40') ergänzt.

Ein Spaltenwert der Länge 0 lässt sich mit QUOTE-Zeichen schreiben, z.B. als "";wenn in der CSV-Funktion DELIMITER ';' QUOTE '"' angegeben wird.

Einschränkungen bei der Verwendung von CSV-Dateien

Die BS2000-Datei wird exklusiv geöffnet. Sie kann deshalb nicht gleichzeitig von derselben oder einer anderen SQL-Transaktion in einer weiteren CSV-Funktion verwendet werden. Eine Abhilfe bietet die CACHE-Annotation, bei der die CSV-Datei temporär zwischengespeichert wird, siehe das Handbuch „ Performance“.

Wenn die Datei nicht geöffnet werden kann, dannn wird eine Fehlermeldung ausgegeben und die Bearbeitung abgebrochen.

Die Datei wird erst geschlossen, wenn die sie enthaltende Abfrage vollständig ausgewertet wurde, wenn die Abfrage nicht mehr benötigt wird (z.B. weil der Cursor, der die Datei verwendet hat, geschlossen wird) oder wenn die CSV-Datei zwischengespeichert ist.

Darüber hinaus gibt es eine Maximalzahl von CSV-Dateien (derzeit: 4), die gleichzeitig geöffnet sein dürfen. Wenn diese Maximalzahl überschritten ist, dann wird eine entsprechende Fehlermeldung ausgegeben.

Wenn für die verwendete Datenbank und für die CSV-Datei jeweils ein codierter Zeichensatz (`CODE_TABLE` ungleich `_NONE_` bzw. `CODED-CHARACTER-SET` ungleich `*NONE`) definiert ist, dann müssen die beiden angegebenen Namen gleich sein.

5.2.8.9 CURRENT_DATE - Aktuelles Datum

Funktionsgruppe: Zeitfunktion

CURRENT_DATE liefert das aktuelle Datum.

```
CURRENT_DATE
```

Ergebnis

Aktuelles Datum.

Kommen Zeitfunktionen innerhalb einer Anweisung mehrmals vor, werden diese simultan ausgeführt (siehe Abschnitt „Zeitfunktionen“).

Datentyp: DATE

5.2.8.10 CURRENT_TIME(3) - Aktuelle Uhrzeit

Funktionsgruppe: Zeitfunktion

CURRENT_TIME(3) liefert die aktuelle Uhrzeit.

```
CURRENT_TIME(3)
```

Ergebnis

Aktuelle Uhrzeit.

Kommen Zeitfunktionen innerhalb einer Anweisung mehrmals vor, werden diese simultan ausgeführt (siehe Abschnitt „Zeitfunktionen“).

Datentyp: TIME

5.2.8.11 CURRENT_TIMESTAMP(3) - Aktueller Zeitstempel

Funktionsgruppe: Zeitfunktion

CURRENT_TIMESTAMP(3) liefert den aktuellen Zeitstempel.

```
CURRENT_TIMESTAMP(3)
```

Ergebnis

Aktueller Zeitstempel.

Kommen Zeitfunktionen innerhalb einer Anweisung mehrmals vor, werden diese simultan ausgeführt (siehe Abschnitt „Zeitfunktionen“).

Datentyp: TIMESTAMP

5.2.8.12 DATE_OF_JULIAN_DAY() - Julianische Tagesnummer umwandeln

Funktionsgruppe: Zeitfunktion

DATE_OF_JULIAN_DAY() liefert zu einer Julianischen Tagesnummer das entsprechende Datum gemäß dem Gregorianischen Kalender (siehe auch die inverse Funktion „JULIAN_DAY_OF_DATE()“ in "JULIAN_DAY_OF_DATE() - Datum umwandeln").

Die Julianische Tagesnummer eines Datums ist die Anzahl Tage, die seit dem 24. November des Jahres 4714 v. Chr. (gemäß dem Gregorianischen Kalender) vergangen sind.

> ¡ DATE_OF_JULIAN_DAY() und JULIAN_DAY_OF_DATE() sind inverse Funktionen. Wenn z.B. eine Bedingung der Form JULIAN_DAY_OF_DATE(*spalte*) < *:benutzervariable* vorliegt, dann kann der SQL-Optimizer diese Bedingung intern zur Bedingung *spalte* < DATE_OF_JULIAN_DAY(*:benutzervariable*) umformen, um die Nutzung von Indizes auf *spalte* zu ermöglichen. *:benutzervariable* darf deshalb nur Werte enthalten, die als Argument von DATE_OF_JULIAN_DAY() erlaubt sind. Dies gilt auch für beliebige konstante Ausdrücke an Stelle von *:benutzervariable*.

`DATE_OF_JULIAN_DAY ( `*ausdruck*` )`

---

ausdruck

Ganzzahliger numerischer Ausdruck. Sein Wert stellt eine Anzahl von Tagen dar, die seit dem 24. November des Jahres 4714 v. Chr. vergangen sind. Er muss zwischen 1721426 und 5373484 liegen.
ausdruck darf kein multipler Wert mit Dimension > 1 sein.

Ergebnis

Ergibt *ausdruck* den NULL-Wert, so ist das Ergebnis der NULL-Wert.

Sonst:

SESAM/SQL interpretiert den Wert, der sich aus *ausdruck* ergibt, als Julianische Tagesnummer. Das Ergebnis der Funktion ist das Datum, das dieser Julianischen Tagesnummer entspricht.

Datentyp: DATE

Beispiel

```
DATE_OF_JULIAN_DAY (2451545)

2000-01-01
```

5.2.8.13 DECRYPT() - Daten entschlüsseln

Funktionsgruppe: Kryptografische Funktion

DECRYPT() entschlüsselt Zeichenketten nach dem AES-Algorithmus mit einem Schlüssel (key) von 128 Bit (16 Bytes) im Electronic Codebook Mode (ECM) in den entsprechenden Wert eines vorgegebenen Datentyps.

```
DECRYPT ( ausdruck , schlüssel , datentyp )
```

ausdruck

Gibt den Wert an, der entschlüsselt werden soll.
Der Wert muss vom alphanumerischen Datentyp CHARACTER oder CHARACTER VARYING sein.
ausdruck darf kein multipler Wert mit Dimension > 1 sein.
Die Länge von *ausdruck* muss ein ganzzahliges Vielfaches von 16 und größer als 0 sein. Auch ein NULL-Wert ist erlaubt.

schlüssel

Schlüssel, mit dem der Wert von *ausdruck* entschlüsselt werden soll.
Alphanumerische Zeichenkette mit einer Länge von 16 Zeichen, also vom Datentyp CHARACTER(16) oder CHARACTER VARYING(n) mit n >= 16.
Außerdem ist der NULL-Wert eines dieser Datentypen zulässig.
Für ein korrektes Ergebnis muss der Schlüssel derselbe sein, der auch für die Verschlüsselung mit ENCRYPT() verwendet wurde.

datentyp

Datentyp des entschlüsselten Wertes (ohne *dimension*-Angabe). Die erlaubten Datentypen hängen von der (maximalen) Länge des Datentyps von *ausdruck* ab, siehe die Tabelle auf der nächsten Seite.

Ergebnis

Wenn der Wert von *ausdruck* oder von *schlüssel* der NULL-Wert ist, dann ist das Ergebnis der NULL-Wert.

Sonst:

Der entschlüsselte Wert von *ausdruck* im angegebenen Datentyp, siehe Tabelle auf der nächsten Seite. Zu möglichen Fehlern siehe „Fehlerfälle“.

Datentyp: der angegebene *datentyp*

Datentyp von *ausdruck*	***datentyp* und Datentyp des Ergebnisses**
CHAR(m), VARCHAR(>= m) [1]	CHAR(n), falls n <= 256 [2]
CHAR(m), VARCHAR(>= m) [1]	VARCHAR(n) [2]

CHAR(m), VARCHAR(>= m) [1]	NCHAR(n) [3]
CHAR(m), VARCHAR(>= m) [1]	NVARCHAR(n) [3]
CHAR(16), VARCHAR(>= 16)	SMALLINT, INTEGER
CHAR(16), VARCHAR(>= 16)	NUMERIC (bis 14 Stellen)
CHAR(32), VARCHAR(>= 32)	NUMERIC (15 bis 30 Stellen)
CHAR(48), VARCHAR(>= 48)	NUMERIC (31 Stellen)
CHAR(16), VARCHAR(>= 16)	DECIMAL (bis 27 Stellen)
CHAR(32), VARCHAR(>= 32)	DECIMAL (28 bis 31 Stellen)
CHAR(16), VARCHAR(>= 16)	FLOAT, REAL, DOUBLE PRECISION
CHAR(16), VARCHAR(>= 16)	DATE, TIME(3), TIMESTAMP(3)

Tabelle 13: Erlaubte Kombinationen bei DECRYPT()

[1]m muss >= 16 und ein ganzzahliges Vielfaches von 16 sein
[2]Die Länge n muss >= 1 sein und zwischen (m - 17) und (m -2) liegen (inklusive)
[3]Die Länge n muss >= 1 sein und zwischen (m/2 - 1) und (m/2 - 8) liegen (inklusive)

Beispiele

Entschlüsselung in einem SELECT-Ausdruck:

```
SELECT DECRYPT(lsatz_encr,'0123456789ABCDEF',NUMERIC(5,0))
AS test_decr FROM leistung
```

Mit der Funktion VALUE_OF_REP lassen sich auch einzelne Werte einer gemeinsam verschlüsselten Zeichenkette entschlüsseln (siehe auch "ENCRYPT() - Daten verschlüsseln"):

```
VALUE_OF_REP (SUBSTRING (DECRYPT (gehaltundbonus, :schluessel, CHAR(12))
   FROM 7 FOR 6), NUMERIC(6))
   AS bonus
```

Fehlerfälle

Bei der Ausführung der DECRYPT-Funktion können folgende Fehler auftreten:

- Die Länge der verschlüsselten Zeichenkette ist 0 oder kein ganzzahliges Vielfaches von 16.
- Der Schlüssel *schlüssel* ist eine Zeichenkette mit Länge ungleich 16 oder es ist nicht derselbe Schlüssel, der zum Verschlüsseln verwendet wurde.
- Der entschlüsselte Wert passt nicht zum angegebenen Datentyp des Ergebnisses (wenn z.B. ein SMALLINT-Wert veschlüsselt wurde, aber in der DECRYPT-Funktion als Ergebnistyp INTEGER angegeben wurde (oder umgekehrt)).

Allerdings wird bei der Ausführung der DECRYPT-Funktion nicht geprüft, dass der entschlüsselte Wert genau denselben Datentyp bekommt den der verschlüsselte Wert hatte. Verschlüsselt und entschlüsselt wird nur die interne Darstellung von Werten, aber keine zusätzliche Information.

So haben zum Beispiel in SESAM/SQL die Werte ungleich NULL der Datentypen INTEGER, CHARACTER(4), NUMERIC(4,0), DECIMAL(7,2) und REAL alle eine interne Darstellung mit genau 4 Bytes. Es kann daher ein Wert vom Datentyp INTEGER verschlüsselt und zu einem Wert vom Typ CHAR(4) oder REAL entschlüsselt werden. Auch wenn zu einem Wert vom Typ NUMERIC(4,0) entschlüsselt wird, liefert die DECRYPT-Funktion keinen Fehler. Je nach entschlüsseltem Wert kann aber bei einer nachfolgenden arithmetischen Operation ein Fehler auftreten.

5.2.8.14 DEE() - Tabelle ohne Spalten

Funktionsgruppe: Tabellenfunktion

Die Tabellenfunktion DEE() liefert eine Tabelle ohne Spalte mit einer Zeile.

In SESAM/SQL gibt es sonst keine derartigen Tabellen. Sie können z.B. verwendet werden um einen Ausdruck ohne Bezug auf eine Basistabelle auszuwerten. Für das Lesen mit DEE() wird kein SQLPrivileg benötigt.

```
DEE [()]
```

Ergebnis

Die Tabelle ohne Spalte mit einer Zeile.

Beispiele

Diese Abfrage liefert Angaben über den SQL-Betrieb:

```
SELECT CURRENT_USER AS "Who am I",
       LOCALTIMESTAMP(3) AS "and what time is it, anyway"
   FROM TABLE(DEE())
```

Folgende Abfrage wird für die Datenbank K9 ausgeführt und könnte eine andere Zeit liefern:

```
SELECT LOCALTIMESTAMP(3) AS "local time on catalog K9" FROM TABLE(K9.DEE())
```

Folgende Abfrage erweitert die Tabelle T um eine Zeile mit NULL-Werten:

```
SELECT * FROM T UNION JOIN TABLE(DEE())
```

5.2.8.15 ENCRYPT() - Daten verschlüsseln

Funktionsgruppe: Kryptografische Funktion

ENCRYPT() verschlüsselt Werte eines beliebigen Datentyps mit dem AES-Algorithmus und einem Schlüssel (key) von 128 Bit (16 Bytes) im Electronic Codebook Mode (ECM).

```
ENCRYPT ( ausdruck , schlüssel )
```

ausdruck

Ausdruck, dessen Wert verschlüsselt werden soll.
Der Wert darf von einem beliebigen Datentyp sein, jedoch nicht CHARACTER VARYING mit Länge >= 31998 und nicht NATIONAL CHARACTER VARYING(16000).
ausdruck darf kein multipler Wert mit Dimension > 1 sein.

schlüssel

Schlüssel, mit dem der Wert von *ausdruck* verschlüsselt werden soll.
Alphanumerische Zeichenkette mit einer Länge von 16 Zeichen, also vom Datentyp CHARACTER(16) oder CHARACTER VARYING(n) mit n >= 16.
Außerdem ist der NULL-Wert eines dieser Datentypen zulässig.

Ergebnis

Wenn der Wert von *ausdruck* oder von *schlüssel* der NULL-Wert ist, dann ist das Ergebnis der NULL-Wert.

Sonst:

Der verschlüsselte Wert von *ausdruck*.

Datentyp:

CHARACTER VARYING mit einer Maximallänge gemäß folgender Tabelle.

Datentyp von *ausdruck*	**Datentyp des Ergebnisses**
CHAR(m)	VARCHAR(n) [1]
VARCHAR(m) mit m <= 31998	VARCHAR(n) [1]
NCHAR(m)	VARCHAR(n) [2]
NVARCHAR(m) mit m <= 15999	VARCHAR(n) [2]
SMALLINT, INTEGER	VARCHAR(16)
NUMERIC (bis 14 Stellen)	VARCHAR(16)

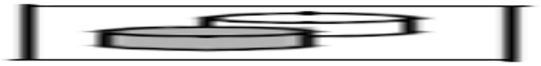
NUMERIC (15 bis 30 Stellen)	VARCHAR(32)
NUMERIC (31 Stellen)	VARCHAR(48)
DECIMAL (bis 27 Stellen)	VARCHAR(16)
DECIMAL (28 bis 31 Stellen)	VARCHAR(32)
FLOAT, REAL, DOUBLE PRECISION	VARCHAR(16)
DATE, TIME(3), TIMESTAMP(3)	VARCHAR(16)

Tabelle 14: Datentyp des Resultats von ENCRYPT()

[1]wobei n das kleinste ganzzahlige Vielfache von 16 ist, das >= m + 2 ist
[2]wobei n das kleinste ganzzahlige Vielfache von 16 ist, das >= 2*m + 2 ist

Wenn *ausdruck* einen Datentyp hat, dessen Werte unterschiedliche Längen haben können (also (NATIONAL) CHARACTER VARYING), dann können auch die verschlüsselten Werte unterschiedliche Längen haben. Die Länge des verschlüsselten Wertes ist aber immer ein Vielfaches von 16 Zeichen, siehe Tabelle oben.Hat zum Beispiel *ausdruck* den Datentyp VARCHAR(20), so hat das Resultat von ENCRYPT() den Datentyp VARCHAR(32); Zeichenketten mit 0 bis 14 Zeichen werden in Zeichenketten mit 16 Zeichen verschluesselt, Zeichenketten mit 15 bis 20 Zeichen werden in Zeichenketten mit 32 Zeichen verschluesselt. Die genaue Länge des unverschlüsselten Wertes lässt sich ohne Kenntnis des Schlüssels nicht dem verschlüsselten Wert entnehmen (sie wird zusammen mit dem Wert verschlüsselt).

Beispiele



Die Werte der Spalte LSATZ werden in die Spalte LSATZ_ENCR verschlüsselt; die unverschlüsselten Werte der Spalte LSATZ werden zu Null gemacht:

```
UPDATE leistung SET
   lsatz_encr = ENCRYPT
(lsatz,'0123456789ABCDEF'),
   lsatz = NULL WHERE lsatz IS NOT NULL
```

Mit der Funktion REP_OF_VALUE lassen sich auch mehrere Wert in einer Zeichenkette verschlüsseln (siehe auch "DECRYPT() - Daten entschlüsseln"):

```
ENCRYPT (REP_OF_VALUE(gehalt) ||
REP_OF_VALUE(bonus), :schluessel)
```

5.2.8.16 EXTRACT() - Bestandteile eines Zeitwertes extrahieren

Funktionsgruppe: Numerische Funktion

EXTRACT() selektiert den angegebenen Bestandteil aus einem Zeitwert. EXTRACT() orientiert sich dabei am Gregorianischen Kalender, auch bei Daten vor dessen Einführung am 15.10.1582.

```
EXTRACT ( bestandteil FROM ausdruck )

bestandteil ::= { YEAR | MONTH | DAY | HOUR | MINUTE | SECOND |
                  YEAR_OF_WEEK | WEEK_OF_YEAR | DAY_OF_WEEK | DAY_OF_YEAR }
```

bestandteil

Angabe des Bestandteils. Zulässige Eingaben sind:

`YEAR`	selektiert das Jahr eines Zeitstempels oder Datums, z.B. `2013`
`MONTH`	selektiert den Monat des Jahres eines Zeitstempels oder Datums, z.B. `2` für Februar
`DAY`	selektiert den Tag des Monats eines Zeitstempels oder Datums, z.B. `25`
`HOUR`	selektiert die Stunde des Tages eines Zeitstempels oder einer Uhrzeit, z.B. `23`
`MINUTE`	selektiert die Minute der Stunde eines Zeitstempels oder einer Uhrzeit, z.B. `58`
`SECOND`	selektiert die Sekunde der Minute eines Zeitstempels oder einer Uhrzeit, z.B. `35.765`
`YEAR_OF_WEEK`	bestimmt das Jahr in dem die Woche eines Zeitstempels oder Datums liegt, z.B. `2013`
`WEEK_OF_YEAR`	bestimmt die Woche des Jahres eines Zeitstempels oder Datums, z.B. `52`
`DAY_OF_WEEK`	bestimmt den Tag der Woche eines Zeitstempels oder Datums, z.B. `3` für Mittwoch
`DAY_OF_YEAR`	bestimmt den Tag des Jahres eines Zeitstempels oder Datums, z.B. `365`

ausdruck

Zeitwerte-Ausdruck. Zulässige Typen sind:

- TIMESTAMP ist bei jedem *bestandteil* zulässig
- TIME bei *bestandteil* `HOUR`, `MINUTE` oder `SECOND`
- DATE bei *bestandteil* `YEAR`, `MONTH`, `DAY`, `YEAR_OF_WEEK`, `WEEK_OF_YEAR`, `DAY_OF_WEEK` oder `DAY_OF_YEAR`

ausdruck darf kein multipler Wert mit Dimension > 1 sein.

Ergebnis

Wenn *ausdruck* den NULL-Wert ergibt, ist das Ergebnis der NULL-Wert.

Sonst:

Der entsprechende numerische Wert.

Datentyp:

DECIMAL(1,0)	bei *bestandteil* `DAY_OF_WEEK`
DECIMAL(2,0)	bei *bestandteil* `MONTH`, `DAY`, `HOUR`, `MINUTE`, `WEEK_OF_YEAR`
DECIMAL(3,0)	bei *bestandteil* `DAY_OF_YEAR`
DECIMAL(4,0)	bei *bestandteil* `YEAR` und `YEAR_OF_WEEK`
DECIMAL(5,3)	bei *bestandteil* `SECOND`

Beispiele

Bestimmen der aktuellen Jahreszahl.

```
EXTRACT (YEAR FROM CURRENT_DATE)
```

Bestimmen des Tages im Jahr.

```
EXTRACT (DAY_OF_YEAR FROM DATE '<date>')
```

Bestimmen der aktuellen Sekunde.

```
EXTRACT (SECOND FROM CURRENT_TIME(3))
```

5.2.8.17 FLOOR() - größte ganze Zahl kleiner als der Wert

Funktionsgruppe: Numerische Funktion

FLOOR() („Runden zum Boden“) bestimmt die größte ganze Zahl, die kleiner oder gleich ist als der angegebene numerische Wert. FLOOR() rundet bei nicht-ganzzahligen numerischen Werten stets ab.

```
FLOOR ( ausdruck )
```

ausdruck

Festpunktwert vom Typ NUMERIC(p,s) oder DECIMAL(p,s) wenn die Zahl der Nachkommastellen s größer als 0 ist, sonst numerischer Ausdruck.
ausdruck darf kein multipler Wert mit Dimension > 1 sein.

Ergebnis

Wenn *ausdruck* den NULL-Wert ergibt, ist das Ergebnis der NULL-Wert.

Sonst:

Die größte ganze Zahl, die kleiner ist als der angegebene numerische Wert.

Datentyp: NUMERIC(q+1,0) oder DECIMAL(q+1,0) mit q=MIN(31,p+1) wenn die Zahl der Nachkommastellen s größer als 0 war, sonst wie *ausdruck*.

Beispiele

`FLOOR (3,14)` ergibt den Wert 3.

`FLOOR (-3,14)` ergibt den Wert -4.

`FLOOR (10,54)` ergibt den Wert 10.

5.2.8.18 HEX_OF_VALUE() - Beliebigen Wert in Hexadezimalform darstellen

Funktionsgruppe: Zeichenkettenfunktion

HEX_OF_VALUE() stellt einen Wert eines beliebigen Datentyps in Hexadezimalform dar, d.h. in einer Zeichenkette bestehend aus den Sedezimalzeichen 0,1,2,...,9,a,b,...,f.

Damit können beliebige Bitmuster lesbar ausgegeben werden.

```
HEX_OF_VALUE ( ausdruck )
```

ausdruck

Ausdruck, dessen Wert in Hexadezimalform dargestellt werden soll.
Sein Datenyp darf nicht CHARACTER VARYING(n) mit einer Maximallänge n > 16000 und nicht NATIONAL CHARACTER VARYING(n) mit einer Maximallänge n > 8000 sein.
ausdruck darf kein multipler Wert mit Dimension > 1 sein.

Ergebnis

Wenn der Wert von *ausdruck* der NULL-Wert ist, dann ist das Ergebnis der NULL-Wert.

Sonst:

Die interne Darstellung des Werts von *ausdruck* in Hexadezimalform als alphanumerische Zeichenkette. Deren Länge ist in der Tabelle auf der nächsten Seite angegeben.

Datentyp: CHARACTER VARYING mit einer Maximallänge gemäß folgender Tabelle.

Datentyp von *ausdruck*	**Datentyp des Ergebnisses**	**Länge des Ergebnisses, falls nicht NULL**
CHAR(n)	VARCHAR(2*n)	2*n
VARCHAR(n) mit n <= 16000	VARCHAR(2*n)	0 bis 2*n, gerade
NCHAR(n)	VARCHAR(4*n)	4*n
NVARCHAR(n) mit n <= 8000	VARCHAR(4*n)	0 bis 4*n, durch 4 teilbar
SMALLINT	VARCHAR(4)	4
INTEGER	VARCHAR(8)	8
NUMERIC(p,s)	VARCHAR(2*p)	2*p
DECIMAL(p,s)	VARCHAR(q [1])	q [1]
REAL, FLOAT (<= 21 Stellen)	VARCHAR(8)	8
DOUBLE PRECISION,	VARCHAR(16)	16

FLOAT (>= 22 Stellen)		
DATE	VARCHAR(12)	12
TIME(3)	VARCHAR(16)	16
TIMESTAMP(3)	VARCHAR(28)	28

Tabelle 15: Datentypen und Längen bei HEX_OF_VALUE()

[1]q=p+2, falls p gerade; q=p+1, falls p ungerade.

Beispiele

```
HEX_OF_VALUE (CAST (254 AS SMALLINT))
   00fe
HEX_OF_VALUE ('ABC')
   c1c2c3
```

Interne Darstellung von Werten in SESAM/SQL

Die interne Darstellung von Werten ungleich NULL in SESAM/SQL, wie sie die Funktionen REP_OF_VALUE() und HEX_OF_VALUE() liefern, ist ähnlich der internen Darstellung entsprechender Werte in anderen Programmiersprachen (z.B. COBOL, C).

SQL-Datentyp	**Beispielwert**	**interne Darstellung (Hexadezimalform)**
CHAR, VARCHAR EBCDIC-Zeichenkette	'ABC'	`c1c2c3`
NCHAR, NVARCHAR UTF16-Zeichenkette	N'ABC'	`004100420043`
SMALLINT 2 Byte mit binärer Darstellung des Wertes (2-Excess Code)	+300 –300	`012C` `fed4`
INTEGER 4 Byte mit binärer Darstellung des Wertes (2-Excess Code)	+300 –300	`0000012c` `fffffed4`
NUMERIC(p,s) p Byte mit EBCDIC-Zeichen für Ziffern, Vorzeichen im letzten Byte	+123.5 –123.5	`f1f2f3f5` `f1f2f3d5`
DECIMAL(p,s) FLOOR(p/2)[1] Byte mit je 2 Ziffern, letztes Byte mit 1 Ziffer und Vorzeichen	+123.5 –123.5	`01235c` `01235d`
REAL, FLOAT (<= 21 Stellen) 1 Byte für Vorzeichen und Exponent, 3 Byte Mantisse	+2.550625e+2 (=255 + 1/16)	`45ff1000`

DOUBLE PRECISION, FLOAT (>= 22 Stellen) 1 Byte für Vorzeichen und Exponent zur Basis 16, 7 Byte Mantisse	+2.5506250000e+2	`c5ff100000000000`
DATE je 2 Byte mit Jahr, Monat, Tag in binärer Darstellung	DATE'2000-08-11'	`07d800008000b`
TIME(3) je 2 Byte mit Stunden, Minuten, Sekunden und Millisekunden in binärer Darstellung	TIME'12:34:56.123'	`000c00220038007b`

TIMESTAMP(3)
wie DATE und TIME(3)
	TIMESTAMP
	'2000-08-11 12:34:56.123'
		`07d800008000b000c00`
		
		`220038007b`

Tabelle 16: Überblick über die interne Darstellung von Werten in SESAM/SQL

[1]FLOOR(p/2) ist die größte ganze Zahl <= p/2

5.2.8.19 JULIAN_DAY_OF_DATE() - Datum umwandeln

Funktionsgruppe: Numerische Funktion

JULIAN_DAY_OF_DATE() liefert zu einem Datum-Zeitwert die entsprechende Julianische Tagesnummer (siehe auch die inverse Funktion „DATE_OF_JULIAN_DAY()“ auf "DATE_OF_JULIAN_DAY() - Julianische Tagesnummer umwandeln").

Die Julianische Tagesnummer für den 24. November des Jahres 4714 v. Chr. (gemäß dem Gregorianischen Kalender) ist „0“.

Die Julianische Tagesnummer für ein späteres Datum ist die Anzahl Tage, die zwischen dem 24. November des Jahres 4714 v. Chr. und dem späteren Datum vergangen sind. So entspricht dem Datum DATE '0001-01-01' die Julianische Tagesnummer „1721426“, dem Datum DATE '9999-12-31' entspricht die Julianische Tagesnummer „5373484“.

> ¡ DATE_OF_JULIAN_DAY() und JULIAN_DAY_OF_DATE() sind inverse Funktionen. Wenn z.B. eine Bedingung der Form JULIAN_DAY_OF_DATE(*spalte*) < *:benutzervariable* vorliegt, dann kann der SQL-Optimizer diese Bedingung intern zur Bedingung *spalte* < DATE_OF_JULIAN_DAY(*:benutzervariable*) umformen, um die Nutzung von Indizes auf *spalte* zu ermöglichen. *:benutzervariable* darf deshalb nur Werte enthalten, die als Argument von DATE_OF_JULIAN_DAY() erlaubt sind. Dies gilt auch für beliebige konstante Ausdrücke an Stelle von *:benutzervariable*.

```
JULIAN_DAY_OF_DATE ( ausdruck )
```

ausdruck

Zeitwerte-Ausdruck, dessen Auswertung einen Wert des Datentyps DATE ergibt;Wert zwischen 0001-01-01 und 9999-12-31.
ausdruck darf kein multipler Wert mit Dimension > 1 sein.

Ergebnis

Ergibt *ausdruck* den NULL-Wert, so ist das Ergebnis der NULL-Wert.

Sonst:

Das Ergebnis ist die Julianische Tagesnummer, die dem Datum entspricht, das sich aus *ausdruck* ergibt.

Datentyp: INTEGER

Beispiele

```
JULIAN_DAY_OF_DATE(DATE'2000-01-01')
```

```
2451545
```

Einen View definieren, der die Bestellungen der letzten zwei Wochen ausgibt:

```
CREATE VIEW bestellungen AS SELECT * FROM auftrag
WHERE adatum >= DATE_OF_JULIAN_DAY(JULIAN_DAY_OF_DATE(CURRENT_DATE)-14)
```

5.2.8.20 LOCALTIME(3) - Aktuelle Ortszeit

Funktionsgruppe: Zeitfunktion

LOCALTIME(3) liefert die aktuelle Ortszeit.

```
LOCALTIME(3)
```

Ergebnis

Aktuelle Ortszeit.

Kommen Zeitfunktionen innerhalb einer Anweisung mehrmals vor, werden diese simultan ausgeführt (siehe Abschnitt „Zeitfunktionen“).

Datentyp: TIME

5.2.8.21 LOCALTIMESTAMP(3) - Aktueller Ortszeitstempel

Funktionsgruppe: Zeitfunktion

LOCALTIMESTAMP(3) liefert den aktuellen Ortszeitstempel.

```
LOCALTIMESTAMP(3)
```

Ergebnis

Aktueller Ortszeitstempel.

Kommen Zeitfunktionen innerhalb einer Anweisung mehrmals vor, werden diese simultan ausgeführt (siehe Abschnitt „Zeitfunktionen“).

Datentyp: TIMESTAMP

5.2.8.22 LOWER() - Großbuchstaben umwandeln

Funktionsgruppe: Zeichenkettenfunktion

LOWER() wandelt Großbuchstaben einer Zeichenkette in Kleinbuchstaben um.

```
LOWER ( ausdruck )
```

ausdruck

Alphanumerischer Ausdruck oder National-Ausdruck.

Ergebnis

Wenn *ausdruck* den NULL-Wert ergibt, ist das Ergebnis der NULL-Wert.

Sonst:

- Wenn *ausdruck* ein alphanumerischer Ausdruck ist, ist das Ergebnis eine Kopie der Zeichenkette, die sich bei der Auswertung von *ausdruck* ergibt, wobei Großbuchstaben des SESAM/SQL-Zeichenvorrats (siehe "SESAM/SQL-Zeichenvorrat") durch entsprechende Kleinbuchstaben ersetzt werden (A-Z ohne Umlaute und ß).
- Wenn *ausdruck* ein National-Ausdruck ist, dann werden Großbuchstaben gemäß den Unicode-Regeln (wie mit der XHCS-Funktion `tolower`) durch entsprechende Kleinbuchstaben ersetzt.

Datentyp: wie *ausdruck*

Beispiele

```
SELECT LOWER(strasse) FROM kunde WHERE knr=100
```

```
    otto-hahn-ring 6
```

`LOWER('Ä')` liefert den Wert 'Ä'.

`LOWER(NX'00C4')` liefert den Wert NX'00E4' (was einem 'ä' entspricht), weil die Unicode-Regeln verwendet werden.

5.2.8.23 MAX() - Maximum bestimmen

Funktionsgruppe: Mengenfunktion

MAX() bestimmt den größten Wert einer Menge von Werten. NULL-Werte werden nicht berücksichtigt. Der Vergleich von alphanumerischen Werten, National-Werten, numerischen Werten und Zeitwerten ist im Abschnitt „Vergleich von zwei Zeilen“ beschrieben..

```
MAX ([ALL | DISTINCT ] ausdruck )
```

ALL / DISTINCT

Die Angabe ALL oder DISTINCT ist syntaktisch erlaubt, hat aber keine Auswirkung auf das Ergebnis.

ausdruck

Numerischer Ausdruck, alphanumerischer Ausdruck, National-Ausdruck oder Zeitwerte-Ausdruck (Einschränkungen siehe Abschnitt „Mengenfunktionen“).

Ergebnis

Ist die Menge der aus *ausdruck* berechneten Werte leer, ist das Ergebnis bzw. das Ergebnis für diese Gruppe der NULL-Wert.

Sonst:

Ohne GROUP BY-Klausel:

Größter Wert in der aus *ausdruck* berechneten Menge (siehe „Mengenfunktion berechnen“).

Mit GROUP BY-Klausel:

Pro Gruppe der größte Wert für diese Gruppe.

Datentyp: wie *ausdruck*

Beispiele

SELECT ohne GROUP BY:

Aus der Tabelle LEISTUNG den höchsten Leistungssatz für Auftrag 211 abfragen (Ergebnis: 1200):

```
SELECT MAX(lsatz) FROM leistung WHERE anr=211
```

SELECT mit GROUP BY:

Für jede Auftragsnummer den höchsten Leistungssatz bestimmen:

```
SELECT anr, MAX(lsatz) FROM leistung GROUP BY anr
   anr
   200      1500
   211      1200
   250      1200
```

5.2.8.24 MIN() - Minimum bestimmen

Funktionsgruppe: Mengenfunktion

MIN() bestimmt das kleinste Element einer Menge von Werten. NULL-Werte werden nicht berücksichtigt. Der Vergleich von alphanumerischen Werten, National-Werten, numerischen Werten und Zeitwerten ist im Abschnitt „Vergleich von zwei Zeilen“ beschrieben.

```
MIN ([ALL | DISTINCT ] ausdruck )
```

ALL / DISTINCT

Die Angabe ALL oder DISTINCT ist syntaktisch erlaubt, hat aber keine Auswirkung auf das Ergebnis.

ausdruck

Numerischer Ausdruck, alphanumerischer Ausdruck, National-Ausdruck oder Zeitwerte-Ausdruck (Einschränkungen siehe Abschnitt „Mengenfunktionen“).

Ergebnis

Ist die Menge der aus *ausdruck* berechneten Werte leer, ist das Ergebnis bzw. das Ergebnis für diese Gruppe der NULL-Wert.

Sonst:

Ohne GROUP BY-Klausel:

Kleinster Wert in der aus *ausdruck* berechneten Menge (siehe „Mengenfunktion berechnen“).

Mit GROUP BY-Klausel:

Pro Gruppe der kleinste Wert für diese Gruppe.

Datentyp: wie *ausdruck*

Beispiele

SELECT ohne GROUP BY:

Aus der Tabelle LEISTUNG den niedrigsten Leistungssatz für Auftrag 211 abfragen (Ergebnis: 50):

```
SELECT MIN(lsatz) FROM leistung WHERE anr=211
```

SELECT mit GROUP BY:

Für jede Auftragsnummer den niedrigsten Leistungssatz bestimmen:

```
SELECT anr, MIN(lsatz) FROM leistung GROUP BY anr
   anr
   200        75
   211        50
   250       125
```

5.2.8.25 MOD() - Rest einer Ganzzahl-Division (Modulo)

Funktionsgruppe: Numerische Funktion

MOD() bestimmt den Rest einer Division zweier ganzer Zahlen.

`MOD` (*dividend, divisor*)

dividend `::=` *ausdruck*
divisor `::=` *ausdruck*

---

dividend

Ganzzahliger numerischer Ausdruck (SMALLINT, INTEGER, NUMERIC(p,0), DECI-MAL(p,0)) für den Dividenden der Division.

divisor

Ganzzahliger numerischer Ausdruck (SMALLINT, INTEGER, NUMERIC(q,0), DECI-MAL(q,0)) für den Divisor der Division. *divisor* darf nicht 0 sein.

dividend und *divisor* dürfen keine multiplen Werte mit Dimension > 1 sein.

Ergebnis

Wenn *dividend* oder *divisor* den NULL-Wert ergibt, ist das Ergebnis der NULL-Wert.

Wenn *dividend* den Wert 0 ergibt, ist das Ergebnis 0.

Sonst:

Das Ergebnis ist der ganzzahlige Rest der Division *dividend* / *divisor* mit demselben Vorzeichen wie *dividend*.

Datentyp: wie *divisor*.

Beispiele

`MOD (3,2)` ergibt den Wert 1.

`MOD (-3,-2)` ergibt den Wert -1.

5.2.8.26 NORMALIZE() - National-Zeichenkette in Normalform bringen

Funktionsgruppe: Zeichenkettenfunktion

Die Codierung eines Zeichens in Unicode ist nicht eindeutig, d.h. es kann für ein Zeichen mehr als eine Codierung geben, siehe „ Basishandbuch“.

Ein typisches Beispiel sind die deutschen Umlaute. Beispielsweise hat das Zeichen Ä sowohl den Code Point `U+00C4` (composed form) als auch die Code Point-Kombination `U+0041` und `U+0308` (decomposed form). In normalisierten Darstellungsformen treten diese Unterschiede nicht auf. Wenn zwei normalisierte Zeichenketten unterschiedlich sind, dann sind es auch ihre unterschiedlichen Code Point-Darstellungen.

NORMALIZE() bringt eine National-Zeichenkette mit National-Zeichen, die Code Points im Bereich `U+0000` bis `U+2FFF` besitzen, in eine normalisierte Form. Andere Zeichen, z.B. Surrogates, bleiben unverändert.

`NORMALIZE (` *ausdruck* `[,{ NFC | NFD } [,` *länge* `]])`

länge `::=` *vorzeichenlose_ganzzahl*

---

ausdruck

National-Ausdruck. Seine Auswertung ergibt eine National-Zeichenkette (Datentyp NCHAR oder NVARCHAR) in normalisierter Form.
ausdruck darf kein multipler Wert mit Dimension > 1 sein.

NFC | NFD

Normalisierungsformen C („Canonical Decomposition followed by Canonical Composition“) bzw. D („Canonical Decomposition“) des Unicode-Standards.

NFC bildet alle Code Points, die zusammen ein Zeichen ergeben, in den entsprechenden Code Point ab. NFD zerlegt jedes „zusammengesetzte“ Zeichen in seine einzelnen Bestandteile, in das Grundzeichen und in die damit verknüpften diakritischen Zeichen. Die Reihenfolge der verknüpften diakritischen Zeichen ist dabei streng festgelegt.

länge

Maximale Länge der normalisierten Darstellung in Code Units.

Länge nicht angegeben:
Das Ergebnis kann abhängig von *ausdruck* bis zu 16000 Code Units lang werden.

Ergebnis

Wenn der Wert von *ausdruck* der NULL-Wert ist, dann ist das Ergebnis der NULL-Wert.

Sonst:

Die normalisierte Darstellung des Wertes von *ausdruck*.
Es gilt: Länge der normalisierten Darstellung (NFC) <= Länge der nicht normalisierten Darstellung <= Länge

der normalisierte Darstellung (NFD).
Wenn die Länge der normalisierten Darstellung größer ist als die angegebene *länge*, dann wird die Funktion mit SQLSTATE abgebrochen.

Datentyp: NVARCHAR(MIN(2*n,16000)),
wobei n die Länge des Argumentdatentyps NCHAR(n) bzw. NVARCHAR(n) ist.Auch für ein Argument vom Typ NCHAR ist der Datentyp NVARCHAR.

Beispiel

Folgende Suchbedingung normalisiert einen Benutzernamen, um unerwünschte Benutzer, die sich in unterschiedlichen Darstellungsformen anmelden, zu erkennen.

```
... WHERE NORMALIZE(:kunde,NFC)
          NOT IN (SELECT name FROM unerwünschte_kunden)
```

5.2.8.27 OCTET_LENGTH() - Zeichenkettenlänge bestimmen

Funktionsgruppe: Numerische Funktion

OCTET_LENGTH() bestimmt die Anzahl der Bytes in einer Zeichenkette.

```
OCTET_LENGTH ( ausdruck )
```

ausdruck

Alphanumerischer Ausdruck oder National-Ausdruck. Seine Auswertung ergibt entweder eine alphanumerische Zeichenkette (Datentyp CHAR oder VARCHAR) oder eine National-Zeichenkette (Datentyp NCHAR oder NVARCHAR). *ausdruck* darf kein multipler Wert mit Dimension > 1 sein. Siehe auch Abschnitt „Verträglichkeit von Datentypen“ .

Ergebnis

Enthält die Zeichenkette den NULL-Wert, ist das Ergebnis der NULL-Wert.

Sonst:

Das Ergebnis ist die Anzahl der Bytes in der Zeichenkette.

Datentyp: INTEGER

Beispiele

Anzahl der Bytes bestimmen, die in der alphanumerischen Zeichenkette 'fuer' enthalten sind (Ergebnis: 4).

```
OCTET_LENGTH ('fuer')
```

Anzahl der Bytes bestimmen, die in der National-Zeichenkette ’Ein kühler Abend’ enthalten sind (Ergebnis: 16).

```
OCTET_LENGTH (U&'Ein k\00FChler Abend')
```

5.2.8.28 POSITION() - Zeichenkettenposition bestimmen

Funktionsgruppe: Numerische Funktion

POSITION() bestimmt die Position einer Zeichenkette in einer anderen Zeichenkette.

```
POSITION ( ausdruck IN ausdruck [USING CODE_UNITS])
```

ausdruck

Alphanumerischer Ausdruck oder National-Ausdruck. Seine Auswertung ergibt entweder eine alphanumerische Zeichenkette (Datentyp CHAR oder VARCHAR) oder eine National-Zeichenkette (Datentyp NCHAR oder NVARCHAR). *ausdruck* darf kein multipler Wert mit Dimension > 1 sein. Siehe auch Abschnitt „Verträglichkeit von Datentypen“ .

Ergebnis

Bei der folgenden Darstellung der möglichen Ergebnisse bezeichnet *zeichenkette1* die Zeichenkette, deren Position bestimmt werden soll, *zeichenkette2* bezeichnet die andere Zeichenkette.

zeichenkette1 und/oder *zeichenkette2* enthält den NULL-Wert:

Das Ergebnis ist der NULL-Wert.

zeichenkette1 hat die Länge 0:

Das Ergebnis ist 1.

zeichenkette1 liegt innerhalb von *zeichenkette2*:

Das Ergebnis ist um 1 größer als die Anzahl der Zeichen (für CHAR/VARCHAR) oder. der Code Units (für NCHAR/NVARCHAR) von *zeichenkette2*, die dem ersten Zeichen bzw. dem ersten Code Unit von *zeichenkette1* vorangehen.

Sonst*:* Das Ergebnis ist 0.

Datentyp: INTEGER

Beispiele

Bestimmen der Position der Zeichenkette 'nett' in der Zeichenkette 'annette' (Ergebnis: 3):

```
POSITION ('nett' IN 'annette')
```

Bestimmen der Position der Zeichenkette 'Dicht' (Ergebnis: 22):

```
POSITION('Dicht' IN 'Wer dichten kann ist Dichtersmann')
```

Bestimmen der Position der Zeichenkette 'Hans' in der Zeichenkette 'Glueck' (Ergebnis: 0):

```
POSITION ('Hans' IN 'Glueck')
```

5.2.8.29 REP_OF_VALUE() - Beliebigen Wert als Zeichenkette darstellen

Funktionsgruppe: Zeichenkettenfunktion

REP_OF_VALUE() stellt einen Wert eines beliebigen Datentyps als alphanumerische Zeichenkette (Folge von Bytes) dar.

```
REP_OF_VALUE ( ausdruck )
```

ausdruck

Ausdruck, dessen Wert als Zeichenkette dargestellt werden soll.
ausdruck darf kein multipler Wert mit Dimension > 1 sein.

Ergebnis

Wenn der Wert von *ausdruck* der NULL-Wert ist, dann ist das Ergebnis der NULL-Wert.

Sonst:

Die interne Darstellung des Wertes von *ausdruck* als Folge von Bytes in einer alphanumerischen Zeichenkette. Für die interne Darstellung von Werten der verschiedenen Datentypen siehe Tabelle 16.

Datentyp: CHARACTER VARYING(n), wobei die maximale Länge n vom Datentyp von *ausdruck* gemäß folgender Tabelle abhängt.

Datentyp von *ausdruck*	**Datentyp des Ergebnisses**	**Länge des Ergebnisses, falls nicht NULL**
CHAR(n)	VARCHAR(n)	n
VARCHAR(n)	VARCHAR(n)	0 bis n
NCHAR(n)	VARCHAR(2*n)	2*n
NVARCHAR(n)	VARCHAR(2*n)	0 bis 2*n, gerade
SMALLINT	VARCHAR(2)	2
INTEGER	VARCHAR(4)	4
NUMERIC(p,s)	VARCHAR(n)	p
DECIMAL(p,s)	VARCHAR(q [1])	q [1]
REAL, FLOAT (<= 21 Stellen)	VARCHAR(4)	4
DOUBLE PRECISION, FLOAT (>= 22 Stellen)	VARCHAR(8)	8
DATE	VARCHAR(6)	6

TIME(3)	VARCHAR(8)	8
TIMESTAMP(3)	VARCHAR(14)	14

Tabelle 17: Datentypen und Längen bei REP_OF_VALUE

[1]q=(p + 2)/2 falls p gerade; q=(p + 1)/2 falls p ungerade.

Beispiele

```
REP_OF_VALUE (CAST (254 AS SMALLINT))
```

254 hat die binäre Darstellung X'00fe' (2 Bytes).
Diese 2 Bytes (nicht abdruckbar) sind auch das Ergebnis des Ausdrucks.

```
REP_OF_VALUE ('ABC')
```

Ergebnis ist die Zeichenkette 'ABC'.

5.2.8.30 SIGN() - Signum bestimmen

Funktionsgruppe: Numerische Funktion

SIGN() bestimmt das Signum (Vorzeichen) eines numerischen Wertes.

```
SIGN ( ausdruck )
```

ausdruck

Numerischer Ausdruck.
ausdruck darf kein multipler Wert mit Dimension > 1 sein.

Ergebnis

Wenn *ausdruck* den NULL-Wert ergibt, ist das Ergebnis der NULL-Wert.

Wenn *ausdruck* den Wert 0 ergibt, ist das Ergebnis 0.

Wenn *ausdruck* > 0 ist, dann ist das Ergebnis 1.

Wenn *ausdruck* < 0 ist, dann ist das Ergebnis -1.

Datentyp: DECIMAL(1,0)

Beispiele

`SIGN (3,14)` ergibt den Wert 1.

`SIGN (-3,14)` ergibt den Wert -1.

5.2.8.31 SUBSTRING() - Teilzeichenkette extrahieren

Funktionsgruppe: Zeichenkettenfunktion

SUBSTRING() extrahiert eine Teilzeichenkette aus einer Zeichenkette.

```
SUBSTRING ( ausdruck FROM startposition [FOR teilkettenlänge ][USING CODE_UNITS])
```

ausdruck

Alphanumerischer Ausdruck oder National-Ausdruck. Seine Auswertung ergibt entweder eine alphanumerische Zeichenkette (Datentyp CHAR oder VARCHAR) oder eine National-Zeichenkette (Datentyp NCHAR oder NVARCHAR). Siehe auch Abschnitt „Verträglichkeit von Datentypen“.

startposition

Numerischer Ausdruck, dessen Datentyp DECIMAL oder NUMERIC ohne Nachkommastellen (SCALE 0), SMALLINT oder INTEGER ist. Die Auswertung von *startposition* ergibt eine Ganzzahl oder eine Festpunktzahl ohne Nachkommastellen. Es darf kein multipler Wert mit Dimension > 1 sein.

startposition gibt die Position eines Zeichens inner- oder außerhalb derjenigen Zeichenkette an, die sich bei der Auswertung von *ausdruck* ergibt. *startposition* bestimmt das Zeichen, ab dem die Teilzeichenkette extrahiert werden soll.

teilkettenlänge

Numerischer Ausdruck, dessen Datentyp DECIMAL oder NUMERIC ohne Nachkommastellen (SCALE 0), SMALLINT oder INTEGER ist. Die Auswertung von *teilkettenlänge* ergibt eine Ganzzahl oder eine Festpunktzahl ohne Nachkommastellen ergibt. Der Wert von *teilkettenlänge* darf nicht < 0 sein. Es darf kein multipler Wert mit Dimension > 1 sein.

teilkettenlänge gibt die maximale Länge der Teilzeichenkette an.

Ergebnis

Bei der folgenden Darstellung der möglichen Ergebnisse bezeichnet *zeichenkette* die Zeichenkette, die sich bei der Auswertung von *ausdruck* ergibt.

Das Ergebnis ist der NULL-Wert, wenn *ausdruck, startposition* und/oder *teilkettenlänge* den NULL-Wert annimmt.

Das Ergebnis ist eine Zeichenkette der Länge 0, wenn eine der folgenden Voraussetzungenn gegeben ist:

- *startposition* ist größer als die Anzahl der Zeichen in *zeichenkette*
- *zeichenkette* hat die Länge 0
- *teilkettenlänge* ist 0
- Die Summe aus *startposition* und *teilkettenlänge* ergibt einen Wert <= 1

Sonst:

Das Ergebnis ist eine Teilzeichenkette von *zeichenkette*. Die Reihenfolge der Zeichen entspricht der Reihenfolge der Zeichen in *zeichenkette*. Die Teilzeichenkette enthält so viele Zeichen, wie durch *startposition* und *teilkettenlänge* vorgegeben ist:

teilkettenlänge ist angegeben und *startposition* >=1:

> Die Teilzeichenkette enthält *teilkettenlänge* Zeichen, höchstens jedoch bis zum letzten Zeichen von *zeichenkette*, beginnend bei dem Zeichen von *zeichenkette*, das durch *startposition* vorgegeben ist.

teilkettenlänge ist angegeben und *startposition* < 1:

> Die Teilzeichenkette enthält (*startposition* + *teilkettenlänge*-1) Zeichen, höchstens jedoch bis zum letzten Zeichen von *zeichenkette,* beginnend beim ersten Zeichen von *zeichenkette*.

teilkettenlänge ist nicht angegeben und *startposition* >= 1:

> Die Teilzeichenkette enthält ab *startposition* alle Zeichen der Zeichenkette bis zum letzten Zeichen.

teilkettenlänge ist nicht angegeben und *startposition* < 1:

> Die gesamte Zeichenkette wird extrahiert.

Datentyp:

Wenn *ausdruck* den alphanumerischen Datentyp CHAR(*n*) oder VARCHAR(*n*) hat, dann hat das Ergebnis den alphanumerischen Datentyp VARCHAR(*n*).

Wenn *ausdruck* den National-Datentyp NCHAR(*n*) oder NVARCHAR(*n*) hat, dann hat das Ergebnis den National-Datentyp NVARCHAR(*n*).

Beispiele

Aus der Zeichenkette 'Pudelshop Anke' soll eine Teilkette extrahiert werden. 'Pudelshop Anke' ist der Firmenname eines Kunden in der Tabelle KUNDE.

startposition ist > 1*, teilkettenlänge* ist angegeben:

```
SELECT SUBSTRING (firma FROM 6 FOR 4) FROM kunde WHERE knr=105
```

> Das Ergebnis ist die Zeichenkette „shop“.

startposition ist gleich 0, *teilkettenlänge* ist angegeben:

```
SELECT SUBSTRING (firma FROM 0 FOR 5) FROM kunde WHERE knr=105
```

> Das Ergebnis ist die Zeichenkette „Pude“ der Länge (0+5-1) = 4.

startposition ist <0 und (*startposition* + *teilkettenlänge* -1) ist größer als die Länge von *zeichenkette*:

```
SELECT SUBSTRING (firma FROM -2 FOR 20) FROM kunde WHERE knr=105
```

> Das Ergebnis ist die Zeichenkette „Pudelshop Anke“.

startposition ist > 1, *teilkettenlänge* ist nicht angegeben:

```
SELECT SUBSTRING (firma FROM 6) FROM kunde WHERE knr=105
```

Das Ergebnis ist die Zeichenkette „shop Anke“.

startposition ist größer als die Anzahl der Zeichen in *zeichenkette*:

```
SELECT SUBSTRING (firma FROM 15 FOR 5) FROM kunde WHERE knr=105
```

Das Ergebnis ist eine Zeichenkette der Länge 0.

5.2.8.32 SUM() - Summe berechnen

Funktionsgruppe: Mengenfunktion

SUM() berechnet die Summe aller Werte einer Menge. NULL-Werte werden nicht berücksichtigt.

```
SUM ([ALL | DISTINCT ] ausdruck )
```

ALL

Alle Werte werden berücksichtigt, auch solche, die doppelt vorkommen.

DISTINCT

Nur verschiedene Werte werden berücksichtigt. Duplikate werden ignoriert.

ausdruck

Numerischer Ausdruck (Einschränkungen siehe Abschnitt „Mengenfunktionen“).

Ergebnis

Ist die Menge der aus *ausdruck* berechneten Werte leer, ist das Ergebnis bzw. das Ergebnis für diese Gruppe der NULL-Wert.

Sonst:

Ohne GROUP BY-Klausel:

Summe der Werte in der aus *ausdruck* berechneten Menge (siehe „Mengenfunktion berechnen“).

Mit GROUP BY-Klausel:

Pro Gruppe die Summe der Werte für diese Gruppe.

Datentyp: wie *ausdruck* mit folgender Stellenzahl:

Ganzzahl oder Festpunktzahl:

Die Gesamtstellenzahl ist 31, die Nachkommastellenzahl bleibt gleich.

Gleitpunktzahl:

Die Gesamtstellenzahl entspricht 21 Binärstellen bei REAL und 53 Binärstellen bei DOUBLE PRECISION.

Ist die Summe der Werte für diesen Datentyp zu groß, erfolgt eine Fehlermeldung.

Beispiel

Aus der Tabelle VERWENDUNG für jede Artikelnummer die Summe der Bestandteile berechnen:

```
SELECT artnr, SUM(anzahl) FROM verwendung GROUP BY artnr
   artnr
     1         4
```

```
120         27
200         20
```

5.2.8.33 TRANSLATE() - Zeichenkette transliterieren bzw. transcodieren

Funktionsgruppe: Zeichenkettenfunktion

TRANSLATE() transliteriert, d.h. wandelt eine alphanumerische Zeichenkette in eine National-Zeichenkette um oder umgekehrt, siehe „ Basishandbuch“.

TRANSLATE() transcodiert, d.h. wandelt eine in eine Zeichenkette im Zeichensatz UTFE in eine National-Zeichenkette im Zeichensatz UTF-16 um oder umgekehrt, siehe „ Basishandbuch“.

```
TRANSLATE ( ausdruck
            USING [[ catalog. ]INFORMATION_SCHEMA.] transname [DEFAULT zeichen ] [ ,länge ] )

zeichen ::= ausdruck
länge ::= vorzeichenlose_ganzzahl
```

ausdruck

Alphanumerischer Ausdruck oder National-Ausdruck.
Seine Auswertung ergibt entweder eine alphanumerische Zeichenkette oder eine National-Zeichenkette. Siehe auch Abschnitt „Verträglichkeit von Datentypen“.*ausdruck* darf kein multipler Wert mit Dimension > 1 sein.

transname

Einfacher Name für eine Transliteration von EBCDIC nach Unicode (Zeichensatz UTF-16) und umgekehrt bzw. für eine Transcodierung von UTF-EBCDIC nach UTF-16 und umgekehrt.

In SESAM/SQL sind alle Transliterationsnamen vordefiniert. Es sind entweder die CCS-Namen, die im BS2000-Subsystem XHCS für die Transliteration zwischen EBCDIC und UTF-16 definiert sind, oder CATALOG_DEFAULT für die Transliteration in der voreingestellten Datenbank, falls für diese CODE_TABLE nicht auf _NONE_ gesetzt ist (siehe CREATE/ALTER CATALOG-Anweisungen im Handbuch „ SQL-Sprachbeschreibung Teil 2: Utilities“). Der CCS-Name darf maximal 8 Zeichen lang sein.

Wenn *ausdruck* ein alphanumerischer Ausdruck ist und der Transliterationsname UTFE (!) angegeben wird, dann wird *ausdruck* von UTF-EBCDIC (Zeichensatz UTFE) in den Zeichensatz UTF-16 transcodiert.

Wenn *ausdruck* ein National-Ausdruck ist (d.h. der Zeichensatz ist UTF-16) und der Transliterationsname UTFE angegeben wird, dann wird *ausdruck* von UTF-16 in den Zeichensatz UTFE transcodiert.

Die Transliteration bzw. Transcodierung kann durch einen Datenbanknamen und den Schema-Namen INFORMATION_SCHEMA qualifiziert werden, ansonsten wird das IN-FORMATION_SCHEMA der voreingestellten Datenbank angenommen.

zeichen

Mit *zeichen* können Sie ein Ersatzzeichen festlegen, das ausgegeben werden soll als Ersatz für Zeichen, die mit dem angegebenen *transname* nicht bearbeitet werden können. Wenn Sie DEFAULT *zeichen* nicht angegeben haben und *ausdruck* ein Zeichen enthält, das mit dem angegebenen *transname* nicht bearbeitet

werden kann, wird die enthaltende SQL-Anweisung mit SQLSTATE abgebrochen. Wenn *ausdruck* den alphanumerischen Datentyp CHAR oder VARCHAR hat, dann muss das Ersatzzeichen den National-Datentyp NCHAR(1) bzw. NVARCHAR(*n*) mit *n*>=1 haben. Wenn *ausdruck* den National-Datentyp NCHAR oder NVARCHAR hat, dann muss das Ersatzzeichen den alphanumerischen Datentyp CHAR(1) bzw.VARCHAR(*n*) mit *n*>=1 haben.

länge

Maximale Länge der transliterierten bzw. trancodierten Zeichenkette in Code Units.

1 <= *länge* <= 16000, wenn *ausdruck* eine alphanumerische Zeichenkette ist (Transliterationsname ist ein EBCDIC-Zeichensatz oder UTFE).

1 <= *länge* <= 32000 wenn *ausdruck* eine National-Zeichenkette ist. (Transliterationsname ist ein EBCDIC-Zeichensatz).

Länge nicht angegeben:
Das Ergebnis hat die maximal mögliche Länge (siehe oben).

Ergebnis

Wenn *ausdruck* und/oder *zeichen* den NULL-Wert ergibt, ist das Ergebnis der NULL-Wert.

Sonst:

Das Ergebnis ist die Zeichenkette in der angegebenen bzw. der maximalen Länge, die sich bei der Transliteration bzw. Transcodierung von *ausdruck* ergibt.

Wenn bei der Transliteration das Ersatzzeichen verwendet werden musste, wird die Warnung SQLSTATE `'01SBB'` ausgegeben.

Wenn die Länge der transliterierten bzw. transcodierten Zeichenkette größer ist als die angegebene oder die maximale *länge*, dann wird die Funktion mit SQLSTATE abgebrochen.

Datentyp:

Wenn *ausdruck* den alphanumerischen Datentyp CHAR(*n*) oder VARCHAR(*n*) hat, dann hat das Ergebnis den National-Datentyp NVARCHAR(*n*).

Wenn *ausdruck* den National-Datentyp NCHAR(*n*) oder NVARCHAR(*n*) hat, dann hat das Ergebnis bei Transliteration den alphanumerischen Datentyp VARCHAR(*n*) und bei Transcoding den National-Datentyp NVARCHAR(*n*) .

Beispiele

Die angegebene National-Zeichenfolge soll mit der Transliteration EDF03IRV in den Standard-Zeichensatz von BS2000 transliteriert werden. Nicht-darstellbare Zeichen werden als Fragezeichen dargestellt.

```
TRANSLATE (NX'0041004200430308' USING
```

```
WELTKUNDEN.INFORMATION_SCHEMA.EDF03IRV DEFAULT '?')
```

Das Ergebnis ist die Zeichenkette „ABC?“.

Die angegebene alphanumerische Zeichenfolge soll als Zeichenfolge mit dem Zeichensatz UTF-EBCDIC interpretiert und in den Unicode-Zeichensatz UTF-16 transcodiert werden.

```
TRANSLATE ('ABC' USING UTFE)
```

```
004100420043
```

Interpretieren einer Datei `NAMETITEL.TXT` im Zeichensatz UTFE (erstellt z.B. mit UNLOAD) als CSV-Datei.

```
CREATE VIEW MYVIEW(x,y) AS
SELECT TRANSLATE(name USING UTFE), TRANSLATE(titel USING UTFE)
FROM TABLE(CSV(FILE 'NAMETITEL.TXT' DELIMITER ';',CHAR(25),VARCHAR(16)))
AS T(name,titel)
```

5.2.8.34 TRIM() - Zeichen entfernen

Funktionsgruppe: Zeichenkettenfunktion

TRIM() entfernt führende und/oder nachgestellte Zeichen einer Zeichenkette.

`TRIM ([[LEADING | TRAILING | BOTH][` *zeichen* `] FROM]` *ausdruck)*

zeichen `::=` *ausdruck*

---

zeichen | *ausdruck*

> *zeichen* und *ausdruck* sind entweder beide alphanumerische Ausdrücke (Datentyp CHAR oder VARCHAR) oder beide National-Ausdrücke (Datentyp NCHAR oder NVARCHAR).
> Keiner der Operanden darf ein multipler Wert mit Dimension > 1 sein.
> Der Wert von *zeichen* hat die Länge 1. Wenn Sie *zeichen* nicht angeben, gilt als Voreinstellung das Leerzeichen ( ).

FROM

> FROM-Operator; Sie dürfen FROM nur angeben, wenn Sie auch LEADING, TRAILING oder BOTH und/oder *zeichen* angeben.

Ergebnis

> Wenn *zeichen* und/oder *ausdruck* den NULL-Wert ergibt, ist das Ergebnis der NULL-Wert.
>
> Sonst:
>
> Das Ergebnis ist eine Kopie der Zeichenkette, die sich bei der Auswertung von *ausdruck* ergibt, jedoch werden führende und/oder nachgestellte Zeichen entfernt, sofern sie dem Wert von *zeichen* entsprechen. Ob führende oder nachgestellte Zeichen entfernt werden, ist abhängig davon, ob Sie LEADING, TRAILING oder BOTH angeben:
>
> LEADING: Führende Zeichen werden entfernt.
>
> TRAILING: Nachgestellte Zeichen werden entfernt.
>
> BOTH: Führende und nachgestellte Zeichen werden entfernt. BOTH ist Voreinstellung.

Datentyp:

Wenn *ausdruck* den alphanumerischen Datentyp CHAR(*n*) oder VARCHAR(*n*) hat, dann hat das Ergebnis den alphanumerischen Datentyp VARCHAR(*n*).

Wenn *ausdruck* den National-Datentyp NCHAR(*n*) oder NVARCHAR(*n*) hat, dann hat das Ergebnis den National-Datentyp NVARCHAR(*n*).

Beispiele

Folgende Beispiele sind äquivalent und liefern den Wert 'ABC'.

```
TRIM(' ABC ')
```

```
TRIM (BOTH ' ' FROM ' ABC ')
```

Folgendes Beispiel liefert den Wert 'URDUGUDRU'.

```
TRIM (BOTH N'N' FROM N'NURDUGUDRUN')
```

In die Tabelle DOZENTEN wird ein Satz eingefügt. Die Spalte ANREDE der Tabelle hat den Datentyp VARCHAR (50). Sie soll den Wert 'Herr Professor' erhalten.
Die entsprechende COBOL-Benutzervariable hat den Datentyp PIC X(50). Damit in die Spalte ANREDE nur der Wert 'Herr Professor' und nicht der Wert 'Herr Professor...' mit 36 nachgestellten Leerzeichen übertragen wird, benutzt man die Zeichenkettenfunktion TRIM:

```
INSERT INTO dozenten (..., anrede, ...)
```

```
VALUES (..., TRIM (TRAILING FROM :ANREDE), ...)
```

5.2.8.35 TRUNC() - Nachkommastellen entfernen

Funktionsgruppe: Numerische Funktion

TRUNC() bestimmt den ganzzahligen Anteil eines numerischen Wertes.

TRUNC() rundet nicht bei nicht-ganzzahligen Werten.

`TRUNC ( ` *ausdruck* ` )`

---

ausdruck

Numerischer Ausdruck.
ausdruck darf kein multipler Wert mit Dimension > 1 sein.

Ergebnis

Wenn *ausdruck* den NULL-Wert ergibt, ist das Ergebnis der NULL-Wert.

Sonst:

ausdruck >= 0:	die größte ganze Zahl, die kleiner oder gleich ist als der angegebene numerische Wert, also FLOOR(*ausdruck)*.
ausdruck < 0:	die kleinste ganze Zahl, die größer oder gleich ist als der angegebene numerische Wert, also CEILING(*ausdruck)*.

Datentyp:	NUMERIC(p-s,0) DECIMAL(q-s,0)	für Datentyp von *ausdruck* NUMERIC(p,s) oder DECIMAL(q,s) mit p,q > s
	wie *ausdruck*	für Datentyp von *ausdruck* ganzzahlig numerisch (SMALLINT, INTEGER, NUMERIC(p,0), DECIMAL(q,0) oder REAL, DOUBLE PRECISION, FLOAT

Beispiele

`TRUNC (3,14)` ergibt den Wert 3.

`TRUNC (-3,14)` ergibt den Wert -3.

5.2.8.36 UPPER() - Kleinbuchstaben umwandeln

Funktionsgruppe: Zeichenkettenfunktion

UPPER() wandelt Kleinbuchstaben einer Zeichenkette in Großbuchstaben um.

```
UPPER ( ausdruck )
```

ausdruck

Alphanumerischer Ausdruck oder National-Ausdruck.

Ergebnis

Wenn *ausdruck* den NULL-Wert ergibt, ist das Ergebnis der NULL-Wert.

Sonst:

- Wenn *ausdruck* ein alphanumerischer Ausdruck ist, ist das Ergebnis eine Kopie der Zeichenkette, die sich bei der Auswertung von *ausdruck* ergibt, wobei Kleinbuchstaben des SESAM/SQL-Zeichenvorrats (siehe "SESAM/SQL-Zeichenvorrat") durch entsprechende Großbuchstaben ersetzt werden (a-z ohne Umlaute und ß).
- Wenn *ausdruck* ein National-Ausdruck ist, dann werden Kleinbuchstaben gemäß den Unicode-Regeln (wie mit der XHCS-Funktion `toupper`) durch entsprechende Großbuchstaben ersetzt.

Datentyp: wie *ausdruck*

Beispiele

```
SELECT UPPER(ort) FROM kunde WHERE knr=100
```

Liefert die Zeichenkette 'MUENCHEN'.

```
UPPER('ä')
```

Liefert den Wert 'ä'.

```
UPPER(NX'00E4')
```

Liefert den Wert NX'00C4' (was einem 'Ä' entspricht), weil die Unicode-Regeln verwendet werden.

5.2.8.37 VALUE_OF_HEX() - Hexadezimalform als Wert darstellen

Funktionsgruppe: Zeichenkettenfunktion

Die Funktion VALUE_OF_HEX() liefert einen Wert des angegebenen Datentyps aus der gegebenen internen Darstellung in Hexadezimalform.
Sie ist die Umkehrfunktion zu HEX_OF_VALUE().

```
VALUE_OF_HEX ( ausdruck , datentyp )
```

ausdruck

Die interne Darstellung des Ergebniswertes in hexadezimaler Form.
Der Wert von *ausdruck* darf nur die Zeichen '0' bis '9', 'a' bis 'f' und 'A' bis 'F' enthalten. *ausdruck* muss einen Datentyp CHARACTER(n) (n gerade) oder CHARACTER VARYING(n) haben.
Sein Wert muss entweder der NULL-Wert sein oder eine Länge haben, die zum Datentypp *datentyp* passt (siehe die Tabelle auf der nächsten Seite). Der Datentyp von *ausdruck* muss Werte dieser Länge bzw. der maximalen Länge zulassen.
ausdruck darf kein multipler Wert mit Dimension > 1 sein.

datentyp

Datentyp des Wertes (ohne *dimension*-Angabe), von dem *ausdruck* die Darstellung in Hexadezimalform ist.
Der Datenyp darf nicht CHARACTER VARYING(n) mit einer Maximallänge n > 16000 und nicht NATIONAL CHARACTER VARYING(n) mit einer Maximallänge n > 8000 sein.

Ergebnis

Wenn der Wert von *ausdruck* der NULL-Wert ist, dann ist das Ergebnis der NULL-Wert.

Sonst:

Der Wert vom angegebenen *datentyp*, dessen interne Darstellung in Hexadezimalform der Wert von *ausdruck* ist. Für die interne Darstellung von Werten der verschiedenen Datentypen siehe Tabelle 16.

Datentyp: der angegebene *datentyp*

> Bei der Ausführung dieser Funktion wird nicht geprüft, ob *datentyp* derselbe Datentyp ist, der zuvor bei der entsprechenden Darstellung in interner Form mit HEX_OF_VALUE() verwendet worden war.

Länge von *ausdruck* in Zeichen	***datentyp***
2*n	CHAR(n)
0 bis 2*n	VARCHAR(n)
4*n	NCHAR(n)
0 bis 4*n, durch 4 teilbar	NVARCHAR(n)

4	SMALLINT
8	INTEGER
2*p	NUMERIC(p,s)
q [1]	DECIMAL(p,s)
8	REAL, FLOAT (<= 21 Stellen)
16	DOUBLE PRECISION, FLOAT (>= 22 Stellen)
12	DATE
16	TIME(3)
28	TIMESTAMP(3)

Tabelle 18: Datentypen und Längen bei VALUE_OF_HEX

[1]q=p+2, falls p gerade; q=p+1, falls p ungerade.

Beispiele

```
VALUE_OF_HEX ('00fe', SMALLINT)
   254
VALUE_OF_HEX ('c1c2c3', CHAR(3))
   ABC
```

5.2.8.38 VALUE_OF_REP() - Zeichenkette als Wert darstellen

Funktionsgruppe: Zeichenkettenfunktion

Die Funktion VALUE_OF_REP() liefert einen Wert des angegebenen Datentyps aus der gegebenen internen Darstellung (Folge von Bytes).
Sie ist die Umkehrfunktion zu REP_OF_VALUE().

```
VALUE_OF_REP ( ausdruck , datentyp )
```

ausdruck

Die interne Darstellung des Ergebniswertes. Für die interne Darstellung von Werten der verschiedenen Datentypen siehe Tabelle 16.
ausdruck muss einen Datentyp CHARACTER(n) (n gerade) oder CHARACTER VARYING(n) haben.
Sein Wert muss entweder der NULL-Wert sein oder eine Länge haben, die zum Datentypp *datentyp* passt (siehe die Tabelle auf der nächsten Seite). Der Datentyp von *ausdruck* muss Werte dieser Länge bzw. der maximalen Länge zulassen.
ausdruck darf kein multipler Wert mit Dimension > 1 sein.

datentyp

Datentyp des Wertes (ohne *dimension*-Angabe), von dem *ausdruck* die interne Darstellung ist.

Ergebnis

Wenn der Wert von *ausdruck* der NULL-Wert ist, dann ist das Ergebnis der NULL-Wert.

Sonst:

Der Wert vom angegebenen *datentyp*, dessen interne Darstellung der Wert von *ausdruck* ist.

Datentyp: der angegebene *datentyp*

> Bei der Ausführung dieser Funktion wird nicht geprüft, ob *datentyp* derselbe Datentyp ist, der zuvor bei der entsprechenden Darstellung in interner Form mit REP_OF_VALUE() verwendet worden war.

Länge von *ausdruck* in Zeichen	***datentyp***
n	CHAR(n)
0 bis n	VARCHAR(n)
2*n	NCHAR(n)
0 bis 2*n, gerade	NVARCHAR(n)
2	SMALLINT

4	INTEGER
p	NUMERIC(p,s)
q [1]	DECIMAL(p,s)
4	REAL, FLOAT (<= 21 Stellen)
8	DOUBLE PRECISION, FLOAT (>= 22 Stellen)
6	DATE
8	TIME(3)
14	TIMESTAMP(3)

Tabelle 19: Datentypen und Längen bei VALUE_OF_REP

[1]q=(p + 2)/2, falls p gerade ist; q=(p + 1)/2, falls p ungerade ist

Beispiele

```
VALUE_OF_REP (X'00fe', SMALLINT)
   254
VALUE_OF_REP ('ABC', CHAR(3))
   ABC
```

5.3 Prädikat

Prädikate sind die Bestandteile von Suchbedingungen (siehe Abschnitt „Suchbedingung“).

Ein Prädikat besteht aus Operanden und Operatoren. Entsprechend den Operatoren sind Prädikate in folgende Gruppen unterteilt:

- Vergleich von zwei Zeilen
- Quantifizierter Vergleich (Vergleich mit den Zeilen einer Tabelle)
- BETWEEN-Prädikat (Bereichsabfrage)
- CASTABLE-Prädikat (Konvertierbarkeit prüfen)
- IN-Prädikat (Elementabfrage)
- LIKE-Prädikat (einfacher Mustervergleich)
- LIKE_REGEX-Prädikat (Mustervergleich mit regulären Ausdrücken)
- NULL-Prädikat (Vergleich auf NULL-Wert)
- EXISTS-Prädikat (Existenzabfrage)

Die einzelnen Gruppen sind im Folgenden in der obigen Reihenfolge beschrieben.

Ein Prädikat liefert den Wahrheitswert wahr, falsch oder unbestimmt. Der Wert eines Prädikats wird berechnet, indem die Werte der Operanden berechnet werden und die jeweiligen Operatoren auf die berechneten Werte angewendet werden. In bestimmten Fällen wird ein Operand nicht oder nur teilweise berechnet, falls das zur Bestimmung des Ergebnisses ausreicht.

In der folgenden Übersicht ist die Syntax aller Prädikate in vereinfachter Form zusammengestellt:

praedikat ::=

{
  zeile vergleichs_op zeile |

  vektor_spalte vergleichs_op ausdruck |

  zeile vergleichs_op { ALL | SOME | ANY } *unterabfrage* |

  zeile [NOT] BETWEEN *zeile* AND *zeile* |

  vektor_spalte [NOT] BETWEEN *ausdruck* AND *ausdruck* |

  ausdruck IS [NOT] CASTABLE AS *datentyp* |

  zeile [NOT] IN { *unterabfrage* | (*zeile* ,...) } |

  vektor_spalte [NOT] IN (*ausdruck* , *ausdruck* ,...) |

  operand [NOT] LIKE *muster* [ESCAPE *zeichen* ...] |

  operand [NOT] LIKE_REGEX *regulärer_ausdruck* [FLAG *modifikatoren*] |

  ausdruck IS [NOT] NULL |

```
  EXISTS unterabfrage
}
```

zeile ::= { (*ausdruck* , . . .) | *ausdruck* | *unterabfrage* }
vektor_spalte ::= [*tabelle* .]{ *spalte[min..max]* | *spalte* (*min..max*) }
vergleichs_op ::= { = | < | > | <= | >= | <> }
operand ::= *ausdruck*
muster ::= *ausdruck*
zeichen ::= *ausdruck*
regulärer_ausdruck ::= *ausdruck*
modifikatoren ::= *ausdruck*

5.3.1 Vergleich von zwei Zeilen

Zwei Zeilen werden gemäß einem Vergleichsoperator lexikografisch verglichen. Haben beide Zeilen nur eine Spalte, dann erhält man den gewöhnlichen Vergleich von zwei Werten.

{ *zeile vergleichs_op zeile* | *vektor_spalte vergleichs_op ausdruck* }

zeile ::= { (*ausdruck* , . . .) | *ausdruck* | *unterabfrage* }

vektor_spalte ::= [*tabelle* .]{ *spalte[min..max]* | *spalte* (*min..max* }

vergleichs_op ::= { = | < | > | <= | >=| <> }

zeile

Operanden für den Vergleich.

Jeder *ausdruck* in *zeile* muss einfach sein. Die Zeile besteht aus den Werten von *ausdruck* in der angegebenen Reihenfolge. Ein einzelner *ausdruck* liefert also eine Zeile mit einer Spalte.

unterabfrage muss eine Tabelle ohne multiple Spalten liefern, die höchstens eine Zeile hat. Diese Zeile ist der Vergleichsoperand. Ist die gelieferte Tabelle leer, so besteht der Vergleichsoperand aus einer Zeile, deren Spalten alle den Wert NULL haben.

Die zu vergleichenden Zeilen müssen dieselbe Anzahl Spalten haben, und die korrespondierenden Spalten der linken und rechten Zeile müssen verträgliche Datentypen haben (siehe Abschnitt „Verträglichkeit von Datentypen“).

vektor_spalte

Eine multiple Spalte, die nach besonderen Regeln verglichen wird. Die Spaltenangabe darf keine Außenreferenz sein.

> ¡ Die eckigen Klammern, die in der Syntax kursiv gedruckt sind, sind Sonderzeichen und müssen in der Anweisung angegeben werden.

ausdruck

Der Wert von *ausdruck* muss einfach sein, und sein Datentyp muss mit dem Datentyp der Ausprägungen von *vektor_spalte* verträglich sein (siehe Abschnitt „Verträglichkeit von Datentypen“).

vergleichs_op

Vergleichsoperator:

=	Vergleich auf gleich
<	Vergleich auf kleiner
>	Vergleich auf größer

<=	Vergleich auf kleiner oder gleich
>=	Vergleich auf größer oder gleich
<>	Vergleich auf ungleich

Ergebnis

zeile vergleichs_op zeile

Werden Zeilen mit mehr als einer Spalte verglichen, so gelten die lexikografischen Vergleichsregeln für Zeilen (siehe Abschnitt „Vergleichsregeln“).

Werden einspaltige Zeilen verglichen, so sind die Vergleichsregeln abhängig vom Datentyp der Spalten (siehe Abschnitt „Vergleichsregeln“).

vektor_spalte vergleichs_op ausdruck

Jede Ausprägung von *vektor_spalte* wird gemäß den Vergleichsregeln für den Datentyp mit *ausdruck* verglichen (siehe unten Abschnitt „Vergleichsregeln“). Die Vergleichsergebnisse werden mit OR verknüpft.

Beispiel

Wenn X eine multiple Spalte mit 3 Elementen ist, ist der Vergleich

```
X[1..3] >= 13
```

zu folgenden Vergleichen äquivalent:

```
X[1] >= 13 OR X[2] >= 13 OR X[3] >= 13
```

5.3.1.1 Vergleichsregeln

Wie eine Vergleichsoperation ausgeführt wird, hängt von ihren Operanden ab. Lexikografische Vergleichsregeln gelten für den Vergleich von Zeilen mit mehr als einer Spalte; beim Vergleich von einspaltigen Zeilen und von Werten richten sich die Vergleichsregeln nach den Datentypen. Diese Vergleichsregeln sind im Folgenden zusammengestellt.

Lexikografischer Vergleich

Das Vergleichsresultat wird aus dem Vergleich der Werte in korrespondierenden Spalten beider Zeilen abgeleitet. Dabei spielen die Werte von weiter rechts stehenden Spalten nur dann eine Rolle, wenn die Werte in allen vorhergehenden Spalten bei beiden Operanden gleich sind (nach diesen Vergleichsregeln wird auch im Lexikon sortiert.)

Formal ausgedrückt heißt das:
Der Vergleich zweier Zeilen mit dem Vergleichsoperator `OP` (der die Werte „<“ oder „>“ hat), den Spaltenwerten `L1, L2,...,Ln` im linken Operanden und den Spaltenwerten `R1,R2,...,Rn` im rechten Operanden liefert den Wahrheitswert wahr bzw. falsch bzw. unbestimmt, wenn es einen Index `i` zwischen `1` und `n` gibt, so dass alle Vergleiche

```
L1    =   R1
L2    =   R2
 .    .    .
 .    .    .
 .    .    .
L(i-1) = R(i-1)
```

als Ergebnis den Wahrheitswert wahr liefern, und der Vergleich

```
Li OP Ri
```

als Ergebnis den Wahrheitswert wahr bzw. falsch bzw. unbestimmt liefert.

Dabei erfolgen die einzelnen Vergleiche je nach Datentyp wie unten beschrieben.

Beachten Sie bitte Folgendes:

- Der Wert in einer Spalte kann durchaus NULL sein, ohne dass als Ergebnis der Wahrheitswert unbestimmt geliefert wird.

  Zum Beispiel ergibt der Vergleich `(1,CAST(NULL AS INT)) < (2,0)` den Wahrheitswert wahr als Ergebnis. Die zweite Spalte wird beim Vergleich ignoriert, weil sich schon die Werte der ersten Spalten unterscheiden.
- Nicht alle Spalten müssen für das Vergleichsergebnis relevant sein. Man darf sich daher nicht darauf verlassen, dass alle Spalten beider Zeilen immer ausgewertet werden.
- Der Vergleich `(L1, L2, ..., Ln) = (R1, R2, ..., Rn)` ist äquivalent ist zum Vergleich `L1 = R1 AND L2 = R2 ... AND Ln = Rn`.

  Bei den Vergleichsoperatoren „<“, „<=“, „>=“, und „>“ gibt es aber keine so einfache Entsprechung.

Vergleich zweier Werte

Ist ein Operand der NULL-Wert, oder sind es beide, so liefern alle Vergleichsoperatoren als Ergebnis den Wahrheitswert unbestimmt (siehe auch Abschnitt „NULL-Wert“).

Alphanumerische Werte

Zwei alphanumerische Werte werden von links nach rechts Zeichen für Zeichen verglichen. Sind die beiden Werte unterschiedlich lang, wird die kürzere Zeichenkette rechts mit Leerzeichen (X'40') aufgefüllt, sodass beide gleich lang sind.

Zwei Zeichenketten sind gleich, wenn sie an jeder Position das gleiche Zeichen haben.

Wenn zwei Zeichenketten nicht gleich sind, bestimmt der EBCDIC-Code der ersten beiden unterschiedlichen Zeichen, welche Zeichenkette größer bzw. kleiner ist.

National-Werte

Zwei National-Werte werden von links nach rechts Code Unit für Code Unit verglichen. Sind die beiden Werte unterschiedlich lang, wird die kürzere Zeichenkette rechts mit Leerzeichen (NX'0020') aufgefüllt, sodass beide gleich lang sind.

Zwei Zeichenketten sind gleich, wenn sie an jeder Position die gleiche Code Unit haben.

Wenn zwei Zeichenketten nicht gleich sind, bestimmt der binäre Wert der ersten beiden unterschiedlichen UTF-16 Code Units, welche Zeichenkette größer bzw. kleiner ist.

Numerische Werte

Werte von numerischen Datentypen werden nach ihrem arithmetischen Wert verglichen. Zwei numerische Werte sind gleich, wenn sie beide 0 sind oder dasselbe Vorzeichen und denselben Betrag haben.

Zeitwerte

Datum, Zeit und Zeitstempel können verglichen werden. Der Datentyp der beiden Operanden muss identisch sein.

- Ein Datum ist größer als ein anderes, wenn es jünger ist.
- Eine Uhrzeit ist größer als eine andere, wenn sie eine spätere Zeit angibt.
- Ein Zeitstempel ist größer als ein anderer, wenn entweder das Datum jünger ist oder, bei gleichem Datum, die Uhrzeit größer ist.

Beispiele

1. `1 <= 1` ist stets wahr.
2. Alphanumerische Werte vergleichen:
   Aus der Tabelle KUNDE die Kunden mit Kundeninformation heraussuchen, die aus München kommen:

```
SELECT firma, kinfo, ort FROM kunde WHERE ort = 'Muenchen'
   firma               kinfo                ort
   Siemens AG        Elektro             Muenchen
   Login GmbH        PC Netzwerke        Muenchen
   Plenzer Trading   Fruechtehandel      Muenchen
```

3. Vergleich mit Unterabfrage, die genau einen Wert liefert:
   Aus der Tabelle VERWENDUNG den Artikel heraussuchen, für den die größte Menge des Bestandteils 501 benötigt wird.

```
SELECT artnr FROM verwendung
   WHERE bestandteil = 501 AND anzahl = (SELECT MAX(anzahl)
   FROM verwendung WHERE bestandteil = 501)
```

Die Unterabfrage liefert genau eine Zeile, da das Maximum für eine einzige Gruppe bestimmt wird.

```
artnr

200
```

Das Beispiel kann man auch mit dem Vergleich zweier Zeilen mit je zwei Spalten schreiben:

```
SELECT artnr FROM verwendung
   WHERE (bestandteil, anzahl) = (SELECT 501, MAX(anzahl)
   FROM verwendung WHERE bestandteil = 501)
```

4. In diesem Beispiel wird eine Cursortabelle mit ORDER BY definiert.
   Mit der WHERE-Klausel werden diejenigen Zeilen ausgewählt, die in der mit ORDER BY festgelegten Reihenfolge hinter den Zeilen mit *knr* `012` und
   fertigsoll `DATE'<date>'` kommen:

   ```
   DECLARE cur_auftrag CURSOR FOR
                       SELECT anr, knr, atext, fertigsoll FROM auftrag
                       WHERE (knr, fertigsoll) > (012, DATE'<date>')
      ORDER BY knr, fertigsoll
   ```

   Man erhält vom Kunden mit der Kunden-Nummer 012 nur die Aufträge, die nach dem angegebenen Datum fertig sein sollen, und von Kunden mit größerer Kunden-Nummer alle Aufträge.

   Die lexikografischen Vergleichsregeln unterscheiden sich nur bei NULL-Werten von den Vergleichsregeln für ORDER BY.

5. Lexikografischer Vergleich von Zeilen

   ```
   DECLARE rest_verwendung CURSOR FOR
      SELECT artnr, bestandteil, SUM(anzahl)
      FROM verwendung
      WHERE (artnr, bestandteil) > (:letzte_artnr, :letzter_bestandteil)
      GROUP BY artnr, bestandteil
      HAVING SUM(anzahl) > 0
      ORDER BY artnr, bestandteil
   ```

   Mit diesem Cursor wird gelesen, wieviele Exemplare von jedem Bestandteil in den verschiedenen Artikeln enthalten sind. Gelesen wird in der Reihenfolge aufsteigender Artikelnummern, und, bei gleichen Artikelnummern, nach aufsteigenden Nummern der Bestandteile.

   Mit der Klausel WHERE läßt sich die Cursortabelle stückweise lesen (FETCH). Wurde z.B. schon bis zum Artikel 120 und Bestandteil 230 gelesen und wird der Cursor erneut mit den Benutzervariablen `:letzte_artnr = 120` und `:letzter_bestandteil = 230` geöffnet, so enthält die Cursortabelle für Artikel 120 nur noch Angaben zu den Bestandteilen mit Nummern > 230 sowie Angaben zu Artikeln mit Nummern > 120 (und beliebigen Bestandteilen).

5.3.2 Quantifizierter Vergleich (Vergleich mit den Zeilen einer Tabelle)

Der Wert einer Zeile wird mit den Zeilen einer Tabelle verglichen. Entweder wird bestimmt, ob der Vergleich für alle Zeilen der Tabelle zutrifft, oder ob er wenigstens für eine Zeile zutrifft.

zeile vergleichs_op `{ ALL | SOME | ANY }` *unterabfrage_1*

zeile `::= { (` *ausdruck* `,...) |` *ausdruck* `|` *unterabfrage_2* `}`

vergleichs_op `::= { = | < | > | <= | >=| <> }`

zeile

Linker Operand für den Vergleich.

Jeder *ausdruck* in *zeile* muss einfach sein. Die Zeile besteht aus den Werten von *ausdruck* in der angegebenen Reihenfolge. Ein einzelner *ausdruck* liefert also eine Zeile mit einer Spalte.

unterabfrage_2 muss eine Tabelle ohne multiple Spalten liefern, die höchstens eine Zeile hat. Diese Zeile ist der linke Vergleichsoperand. Ist die gelieferte Tabelle leer, so besteht der Vergleichsoperand aus einer Zeile, deren Spalten alle den Wert NULL haben.

vergleichs_op

Vergleichsoperator:

=	Vergleich auf gleich
<	Vergleich auf kleiner
>	Vergleich auf größer
<=	Vergleich auf kleiner oder gleich
>=	Vergleich auf größer oder gleich
<>	Vergleich auf ungleich

unterabfrage_1

Die Anzahl der Spalten muss gleich der Anzahl der Spalten von *zeile* sein, und korrespondierende Spalten von *zeile* und *unterabfrage_1* müssen verträgliche Datentypen haben (siehe Abschnitt „Verträglichkeit von Datentypen“).

Ergebnis

ALL

Wahr, wenn der rechte Operand eine leere Tabelle ist, oder wenn der Vergleich des linken Operanden *zeile* mit allen Zeilen des rechten Operanden wahr ergibt.

Falsch, wenn der Vergleich des linken Operanden *zeile* mit mindestens einer Zeile des rechten Operanden falsch ergibt.

Unbestimmt, sonst.

SOME / ANY

Wahr, wenn der Vergleich des linken Operanden *zeile* mit mindestens einer Zeile des rechten Operanden wahr ergibt.

Falsch, wenn der rechte Operand eine leere Tabelle ist, oder wenn der Vergleich des linken Operanden *zeile* mit allen Zeilen des rechten Operanden falsch ergibt.

Unbestimmt, sonst.

Alle Vergleiche erfolgen nach den Vergleichsregeln in Abschnitt „Vergleichsregeln“ auf "Vergleichsregeln".

Beispiele

Folgender Vergleich ist wahr, wenn das aktuelle Datum nach allen Terminen in der Ergebnisspalte liegt und all diese Termine ungleich NULL sind. Er ist falsch, wenn das aktuelle Datum früher als ein Termin oder gleichzeitig mit einem Termin ungleich NULL der Ergebnisspalte ist. In allen anderen Fällen ist das Ergebnis unbestimmt.

```
CURRENT_DATE > ALL (SELECT fertigsoll FROM auftrag)
```

Aus der Tabelle VERWENDUNG die Artikel heraussuchen, die einen Bestandteil enthalten, dessen Anzahl höher ist als die Anzahl aller Bestandteile des Artikels mit der Artikelnummer

```
SELECT artnr FROM verwendung WHERE anzahl > ALL (SELECT anzahl FROM verwendung
WHERE artnr = 1)
```

5.3.3 BETWEEN-Prädikat (Bereichsabfrage)

Es wird bestimmt, ob eine Zeile in einem Bereich liegt, der durch eine obere und untere Grenze angegeben wird.

```
{ zeile_1 [NOT] BETWEEN zeile_2 AND zeile_3 |
  vektor_spalte [NOT] BETWEEN ausdruck AND ausdruck }
zeile ::= { (ausdruck ,...) | ausdruck | unterabfrage_2 }
vektor_spalte ::= [tabelle.] { spalte[min..max] | spalte(min..max) }
```

zeile

Jeder *ausdruck* in *zeile* muss einfach sein. Die Zeile besteht aus den Werten von *ausdruck* in der angegebenen Reihenfolge. Ein einzelner *ausdruck* liefert also eine Zeile mit einer Spalte.

unterabfrage muss eine Tabelle liefern, deren Spalten alle einfach sind, und die höchstens eine Zeile hat. Diese Zeile ist der Operand. Ist die gelieferte Tabelle leer, so besteht der Operand aus einer Zeile, deren Spalten alle den Wert NULL haben.

Die drei Zeilen müssen dieselbe Anzahl von Spalten haben, und korrespondierende Spalten müssen verträgliche Datentypen haben (siehe Abschnitt „Verträglichkeit von Datentypen“).

vektor_spalte

Eine multiple Spalte mit besonderen Regeln für das Ergebnis. Die Spaltenangabe darf keine Außenreferenz sein.

ausdruck

Die Werte müssen einfach sein, und ihre Datentypen müssen mit dem Datentyp der Ausprägungen von *vektor_spalte* verträglich sein (siehe Abschnitt „Verträglichkeit von Datentypen“).

Ergebnis

zeile_1 BETWEEN *zeile_2* AND *zeile_3* ist identisch mit:
(*zeile_1* >= *zeile_2*) AND (*zeile_1* <= *zeile_3*)

zeile_1 NOT BETWEEN *zeile_2* AND *zeile_3* ist identisch mit:
NOT (*zeile_1* BETWEEN *zeile_2* AND *zeile_3*)

vektor_spalte [NOT] BETWEEN *ausdruck* AND *ausdruck*

- Die Bereichsabfrage wird für jede Ausprägung von *vektor_spalte* durchgeführt.
- Die Einzelergebnisse werden mit OR verknüpft.

Beispiel

Wenn X eine multiple Spalte mit 3 Elementen ist, ist die Bereichsabfrage `X[1..3] BETWEEN 13 AND 20` zu folgenden Bereichsabfragen äquivalent:

```
X[1] BETWEEN 13 AND 20 OR X[2] BETWEEN 13 AND 20 OR X[3] BETWEEN 13 AND 20
```

Beispiele

BETWEEN-Prädikat mit numerischem Bereich:

Aus der Tabelle ARTIKEL alle Artikel mit Artikelbezeichnung heraussuchen, deren Preis zwischen 0 und 10 Euro liegt.

```
SELECT artnr, artbez, preis FROM artikel
   WHERE preis BETWEEN 0.00 AND 10.00
artnr   artbez       preis
  210   V-Nabe        5.00
  220   H-Nabe        5.00
  230   Felge        10.00
  240   Speiche       1.00
  500   Schraube M5   1.10
  501   Mutter        0.75
```

BETWEEN-Prädikat mit einem Zeitintervall:

Aus der Tabelle AUFTRAG die Aufträge mit Auftragsnummer, Kundennummer, Auftragsdatum und Auftragstext heraussuchen, die im Dezember 2013 gestellt wurden:

```
SELECT anr, knr, atext, adatum FROM auftrag
   WHERE adatum BETWEEN DATE'2013-12-01' AND DATE'2013-12-31'
   anr  knr  atext                    adatum
   210  106  Kunden-Verwaltung        2013-12-13
   211  106  Datenbankentwurf KUNDEN  2013-12-30
```

BETWEEN-Prädikat mit Benutzervariable:

:MINIMUM ist eine Benutzervariable. Der Vergleich ist wahr, wenn das Produkt `LSATZ*LANZ` (Preis pro Leistungseinheit mal Anzahl der Leistungseinheiten) außerhalb des angegebenen Bereichs liegt. Er ist falsch, wenn das Produkt innerhalb des Bereichs liegt. Das Ergebnis ist unbestimmt, wenn der Wert von LSATZ oder LANZ unbestimmt ist.

```
lsatz*lanz NOT BETWEEN :MINIMUM AND 2000
```

5.3.4 CASTABLE-Prädikat (Konvertierbarkeitsprüfung)

Es wird geprüft, ob ein Ausdruck in einen bestimmten Datentyp umgewandelt werden kann.

Mit dem CASTABLE-Prädikat können Sie die Ausführbarkeit eines entsprechenden CAST-Ausdruckes (siehe Abschnitt „CAST-Ausdruck“) schon vor seiner Ausführung prüfen und geeignet reagieren.

ausdruck `IS [NOT] CASTABLE AS` *datentyp*

ausdruck

CAST-Operand. Der Wert von *ausdruck* darf kein multipler Wert mit Dimension > 1 sein.

datentyp

Zieldatentyp für das Ergebnis des entsprechenden CAST-Ausdrucks.
datentyp darf keine Dimension für eine multiple Spalte enthalten.

> ¡ Der Datentyp von *ausdruck* muss mit *datentyp* kombinierbar sein, siehe die Tabelle 23.

Ergebnis

Ohne NOT:

Wahr, wenn *ausdruck* in den angegebenen Datentyp konvertiert werden kann. Falsch, wenn *ausdruck* nicht in den angegebenen Datentyp konvertiert werden kann.

Mit NOT:

Wahr, wenn *ausdruck* nicht in den angegebenen Datentyp konvertiert werden kann. Falsch, wenn *ausdruck* in den angegebenen Datentyp konvertiert werden kann.

Beispiel

Prüfen, ob eine Eingabe in einen numerischen Datentyp bestimmter Länge konvertiert werden kann.

```
CASE WHEN :input IS CASTABLE AS NUMERIC(7,2)
     THEN CAST :input AS NUMERIC(7,2)
     ELSE -1
END
```

5.3.5 IN-Prädikat (Elementabfrage)

Es wird bestimmt, ob eine Zeile in einer Tabelle vorkommt.

```
{ zeile_1 [NOT] IN { unterabfrage_2 | ( zeile_2 , ...) } |
  vektor_spalte [NOT] IN ( ausdruck , ...) }
zeile_1 ::= { ( ausdruck ,...) | ausdruck | unterabfrage_1 }
zeile_2 ::= { ( ausdruck ,...) | ausdruck }
vektor_spalte ::= [ tabelle .] { spalte[min..max] | spalte ( min..max ) }
```

zeile_1

liefert eine Zeile.

Jeder *ausdruck* in *zeile_1* muss einfach sein. Die Zeile besteht aus den Werten von *ausdruck* in der angegebenen Reihenfolge. Ein einzelner *ausdruck* liefert also eine Zeile mit einer Spalte.

unterabfrage_1

muss eine Tabelle liefern, deren Spalten alle einfach sind, und die höchstens eine Zeile hat. Diese Zeile ist der linke Operand. Ist die gelieferte Tabelle leer, so besteht der Operand aus einer Zeile, deren Spalten alle den Wert NULL haben.

unterabfrage_2

diese Tabelle ist der rechte Operand.

zeile_2

Der rechte Operand ist die Tabelle, dessen einzelne Zeile(n) mit *zeile_2* angegeben sind. Ist *zeile_2* mehrfach angegeben, so ergibt sich der Datentyp der Spalten der Tabelle aus den Regeln, die unter „Datentyp der Ergebnisspalten bei UNION“ beschrieben sind.

zeile_1, *zeile_2*, *unterabfrage_1* und *unterabfrage_2* müssen jeweils dieselbe Anzahl Spalten haben, und die korrespondierenden Spalten müssen verträgliche Datentypen haben (siehe Abschnitt „Verträglichkeit von Datentypen“).

vektor_spalte

Eine multiple Spalte mit besonderen Regeln für das Ergebnis. Die Spaltenangabe darf keine Außenreferenz sein.

ausdruck

Die Werte müssen einfach sein, und ihre Datentypen müssen mit dem Datentyp der Ausprägungen von *vektor_spalte* verträglich sein (siehe Abschnitt „Verträglichkeit von Datentypen“).

Ergebnis

zeile_1 IN *unterabfrage_2* bzw. *zeile_1* IN (*zeile_2 ,...*):

Wahr, wenn der Vergleich von *zeile_1* auf Gleichheit mit mindestens einer Zeile des rechten Operanden wahr ergibt.

Falsch, wenn der Vergleich von *zeile_1* auf Gleichheit mit jeder Zeile des rechten Operanden falsch ergibt, oder wenn der rechte Operand eine Unterabfrage ist, die eine leere Tabelle liefert.

Unbestimmt, sonst.

zeile_1 NOT IN *unterabfrage_2* bzw. *zeile_1* NOT IN (*zeile_2 ,...*):

ist identisch mit:

NOT (*zeile_1* IN *unterabfrage_2*) bzw. NOT (*zeile_1* IN **(** *zeile_2* ,...**)**)

Es gelten die Vergleichsregeln für den Vergleichsoperator „=“ (siehe auch Abschnitt „Vergleichsregeln“).

vektor_spalte [NOT] IN (*ausdruck, ,...*)

Das IN-Prädikat wird für jede Ausprägung von *vektor_spalte* ausgewertet. Die Einzelergebnisse werden mit OR verknüpft.

Beispiel

Wenn X eine multiple Spalte mit 3 Elementen ist, ist die Elementabfrage `X[1..3] IN (13, 20, 30)` zu folgenden Elementabfragen äquivalent:

```
X[1] IN (13, 20, 30) OR X[2] IN (13, 20, 30) OR X[3] IN (13, 20, 30)
```

Beispiele

IN-Prädikat mit einzelnen Zeilen als rechtem Operanden:

Aus der Tabelle KUNDE alle Kunden mit Kundeninformation heraussuchen, die aus München oder Berlin sind.

```
SELECT firma, kinfo, ort FROM kunde
   WHERE ort IN ('Muenchen','Berlin')

   firma            kinfo           ort
   Siemens AG       Elektro         Muenchen
   Login GmbH       PC-Netzwerke    Muenchen
```

```
Plenzer Trading  Fruechtehandel  Muenchen
Jonas Fischladen Einzelhandel    Berlin
```

IN-Prädikat mit Unterabfrage als rechtem Operanden:

Aus den Tabellen AUFTRAG und LEISTUNG die Aufträge heraussuchen, bei denen keine Schulung durchgeführt wurde.

```
SELECT knr FROM auftrag
WHERE anr NOT IN (SELECT anr FROM leistung WHERE ltext = 'Schulung')
```

5.3.6 LIKE-Prädikat (einfacher Mustervergleich)

Es wird geprüft, ob ein alphanumerischer Wert oder ein National-Wert zu einem angegebenen Muster passt. Ein Muster ist eine Zeichenkette, die außer normalen Zeichen auch Platzhalter und Entwertungszeichen enthalten kann.

Ein Platzhalter steht für ein oder mehrere andere Zeichen. Platzhalter können auch als normales Zeichen im Muster erscheinen, wenn sie mit dem Entwertungszeichen entwertet werden. Das Entwertungszeichen können Sie mit der ESCAPE-Klausel definieren.

operand `[NOT] LIKE` *muster* `[ESCAPE` *zeichen* `]`

operand `::=` *ausdruck*
muster `::=` *ausdruck*
zeichen `::=` *ausdruck*

operand

Alphanumerischer Ausdruck oder National-Ausdruck, der den Operanden für den Mustervergleich darstellt.

Der Wert von *operand* muss entweder einfach sein oder der Name einer multiplen Spalte. Ist der Operand eine multiple Spalte, darf die Spaltenangabe keine Außenreferenz, also nicht Spalte eines übergeordneten Abfrage-Ausdrucks sein.

muster

Alphanumerischer Ausdruck oder National-Ausdruck, zu dessen Wert der Wert von *operand* passen soll. *muster* darf enthalten:

- normale Zeichen (d.h. ohne Platzhalter und Entwertungszeichen)
- Platzhalter

Platzhalter	Bedeutung
_ (Unterstrich)	ein beliebiges Zeichen
%	beliebige (auch leere) Folge von Zeichen

- Entwertungszeichen (gefolgt von Platzhalter oder Entwertungszeichen)

Leerzeichen in *muster*, auch am Anfang oder Ende, gehören zum Muster.

ESCAPE-Klausel

Mit der ESCAPE-Klausel können Sie ein Entwertungszeichen (ESCAPE-Zeichen) definieren. Entwertungszeichen vor Platzhaltern bewirken, dass die Platzhalter ihre Funktion als Platzhalter verlieren und statt dessen als normale Zeichen interpretiert werden. Mit dem Entwertungszeichen können Sie auch das Entwertungszeichen entwerten und als normales Zeichen verwenden.

zeichen

Alphanumerischer Ausdruck oder National-Ausdruck, dessen Wert die Länge 1 hat.*zeichen* gilt in diesem Vergleich als Entwertungszeichen.

ESCAPE-Klausel nicht angegeben:
Es ist kein Entwertungszeichen definiert.

> Die Datentypen von *operand*, *muster* und *zeichen* müssen vergleichbar sein, d.h. entweder haben alle einen der Datentypen CHAR und VARCHAR, oder alle einen der Datentypen NCHAR und NVARCHAR, siehe auch Abschnitt „Verträglichkeit von Datentypen“.

Ergebnis

operand einfach:

Unbestimmt, wenn der Wert von *operand*, *muster* oder *zeichen* der NULL-Wert ist, sonst:

Ohne NOT:

Wahr, wenn die Platzhalter für Zeichen und Zeichenfolgen in *muster* jeweils durch Zeichen und Zeichenfolgen ersetzt werden können, sodass der entstehende Wert gleich dem Wert von *operand* ist und die gleiche Länge hat.

Falsch, sonst.

Mit NOT:

Wahr, wenn die Platzhalter für Zeichen und Zeichenfolgen in *muster* nicht jeweils durch Zeichen und Zeichenfolgen ersetzt werden können, sodass der entstehende Wert gleich dem Wert von *operand* ist und die gleiche Länge hat.

Falsch, sonst.

operand multiple Spalte:

Der Mustervergleich wird für jede Ausprägung der multiplen Spalte durchgeführt. Die Einzelergebnisse werden mit OR verknüpft.

Beispiele

Aus der Tabelle KONTAKT alle Kontaktpersonen heraussuchen, deren Vorname mit Ro beginnt:

```
SELECT vorname, nachname FROM kontakt WHERE vorname LIKE 'Ro%'
vorname    nachname
Roland     Loetzerich
Robert     Heinlein
```

Die folgende Anweisung sucht aus einer alphanumerischen Spalte SP einer Tabelle TAB alle Zeichenfolgen, die mit Unterstrich beginnen und mit mindestens einem Leerzeichen enden:

```
SELECT * FROM tab WHERE sp LIKE '@_% ' ESCAPE '@'
```

Folgendes Prädikat liefert wahr für alle dreistelligen Werte von ANREDE, deren erstes Zeichen „M“ und deren drittes Zeichen „.“ ist, also auch für „Mr.“ und „Ms.“. „_“ ist ein Platzhalter, der für ein beliebiges Zeichen steht. Da der Datentyp der Spalte ANREDE CHAR(20) ist, muss die Zeichenkette mit Leerzeichen auf genau die Länge 20 aufgefüllt werden.

```
anrede LIKE 'M_.                 '
```

Durch das Escape-Zeichen „!“ wird der Platzhalter „%“ entwertet, so dass der Vergleich nur für 'Reisekosten% Rabatt' wahr ergibt.

```
ltext LIKE 'Reisekosten!%Rabatt' ESCAPE '!'
```

5.3.7 LIKE_REGEX-Prädikat (Mustervergleich mit regulären Ausdrücken)

Es wird geprüft, ob ein alphanumerischer Wert zu einem angegebenen regulären Ausdruck passt. Reguläre Ausdrücke sind genau definierte Suchmuster, die weit über die Möglichkeiten der Suchmuster im LIKE-Prädikat hinaus gehen. Reguläre Ausdrücke sind ein mächtiges Mittel, um große Datenbestände nach komplexen Suchausdrücken zu durchsuchen. Sie werden seit langem z.B. in der Programmiersprache Perl eingesetzt.

operand `[NOT] LIKE_REGEX` *regulärer_ausdruck* `[FLAG` *modifikatoren* `]`

operand `::=` *ausdruck*
regulärer_ausdruck `::=` *ausdruck*
modifikatoren `::=` *ausdruck*

operand

Alphanumerischer Ausdruck, der den Operanden für den Vergleich mit dem regulären Ausdruck darstellt. Der Wert von *operand* darf kein multipler Wert mit Dimension > 1 sein.

regulärer_ausdruck

Alphanumerischer Ausdruck, dessen Wert ein regulärer Ausdruck ist, zu dem der Wert von *operand* passen soll. Zum Aufbau reguläre Ausdrücke siehe "LIKE_REGEX-Prädikat (Mustervergleich mit regulären Ausdrücken)". Modifikatoren für *regulärer_ausdruck* geben Sie in der FLAG-Klausel an.
Der Wert von *regulärer_ausdruck* darf kein multipler Wert mit Dimension > 1 sein.

FLAG-Klausel

Alphanumerischer Ausdruck der Modifikatoren für *regulärer_ausdruck*. Folgende Modifikatoren können Sie angeben:

modifikatoren	**Bedeutung**
i (caseless)	Wenn dieser Modifikator gesetzt ist, passen Buchstaben im Suchmuster sowohl auf groß als auch auf klein geschriebene Buchstaben.
m (multiline)	Standardmäßig behandelt SESAM/SQL eine zu durchsuchende Zeichenkette wie eine einzige Zeile von Zeichen, auch wenn sie tatsächlich mehrere Zeilenumbrüche (NEWLINE-Zeichen, siehe "CSV() - BS2000-Datei als Tabelle lesen") enthält. Das Metazeichen für einen Zeilenanfang (^) passt nur auf den Anfang der Zeichenkette, das Metazeichen für ein Zeilenende ($) nur auf das Ende der Zeichenkette. Wenn dieser Modifikator gesetzt ist, passen die Zeilenanfang- und Zeilenende Konstrukte in der Zeichenkette sowohl direkt nach, bzw. vor einem Zeilenumbruch als auch auf deren Anfang und Ende. Falls die Zeichenkette kein NEWLINE-Zeichen enthält, oder im Suchmuster kein ^ oder $ vorkommt, hat dieser Modifikator keine Wirkung.
s (dotall)	Wenn dieser Modifikator gesetzt ist, passt das Metazeichen Punkt im Suchmuster auf alle Zeichen inklusive Zeilenumbrüche (NEWLINE-Zeichen, siehe "CSV() - BS2000-Datei als Tabelle lesen"). Ohne diesen Modifikator sind Zeilenumbrüche ausgeschlossen.

	Unabhängig davon, ob dieser Modifikator gesetzt ist, passt eine verneinende Zeichenklasse wie z.B. [^a] immer auf einen Zeilenumbruch.
x (extended)	Wenn dieser Modifikator gesetzt ist, werden Leerzeichen im Suchmuster ignoriert, sofern sie nicht maskiert sind oder sich innerhalb einer Zeichenklasse befinden. Außerdem werden Zeichen, die außerhalb einer Zeichenklasse zwischen nicht maskierten # stehen, einschließlich dem nächsten Zeilenumbruch ignoriert. Dies bietet die Möglichkeit, Kommentare in komplizierte Suchmuster einzufügen. Beachten Sie aber, dass dies nur für Datenzeichen gilt. Leerräume dürfen niemals innerhalb einer Folge spezieller Zeichen auftreten, zum Beispiel in der Folge (?(, die einen bedingten Teilausdruck einleitet.

modifikatoren muss aus Kleinbuchstaben bestehen. Jedes Zeichen kann mehrmals angegeben werden. Es dürfen keine Leerzeichen angegeben werden.

FLAG-Klausel nicht angegeben:
Es sind keine Modifikatoren für *regulärer_ausdruck* definiert.

Ergebnis

Unbestimmt, wenn der Wert von *operand*, *regulärer_ausdruck* oder *modifikatoren* der NULL-Wert ist, sonst:

Ohne NOT:

Wahr, wenn die Platzhalter für Zeichen und Zeichenfolgen in *regulärer_ausdruck* jeweils durch Zeichen und Zeichenfolgen ersetzt werden können, sodass der entstehende Wert gleich dem Wert von *operand* ist und die gleiche Länge hat.

Falsch, sonst.

Mit NOT:

Wahr, wenn die Platzhalter für Zeichen und Zeichenfolgen in *regulärer_ausdruck* nicht jeweils durch Zeichen und Zeichenfolgen ersetzt werden können, sodass der entstehende Wert gleich dem Wert von *operand* ist und die gleiche Länge hat.

Falsch, sonst.

Beispiele

Aus der Tabelle KONTAKT alle Kontaktpersonen heraussuchen, deren Nachname die Zeichenfolge meier „oder so ähnlich“ enthält:

```
SELECT vorname, nachname FROM kontakt
   WHERE nachname LIKE_REGEX '[a-z]* M [ae]? [iy] [a-z]* r' FLAG 'ix'
   vorname    nachname
   Albert     Gansmeier
   Berta      Hintermayr
   Thea       Mayerer
   Herbert    Meier
   Anton      Kusmir
```

In der Tabelle KONTAKT die nicht korrekten Postleitzahlen der Spalte PLZ herausfinden:

```
SELECT * FROM kontakt WHERE plz NOT LIKE_REGEX '\d{5}'
```

Aus der Tabelle KONTAKT alle E-Mail-Kontakte zu Fujitsu heraussuchen:

```
SELECT address FROM kontakt
   WHERE address LIKE_REGEX '([A-Za-z])+\.([A-Za-z]+)@fujitsu\.com'
   address
   Albert.Gansmeier@fujitsu.com
   Berta.Hintermayr@fujitsu.com
   Thea.Mayerer@fujitsu.com
```

Reguläre Ausdrücke in SESAM/SQL

Die regulären Ausdrücke im LIKE_REGEX-Prädikat entsprechen den regulären Ausdrücken der Programmiersprache Perl mit folgenden Ausnahmen:

- sie werden nicht in Begrenzer eingeschlossen
- es gibt keine „Ersetzen“-Funktion
- die Modifikatoren werden in der FLAG-Klausel angegeben

Sonderzeichen

Sonderzeichen in regulären Ausdrücken haben spezielle Funktionen:

Zeichen	Bedeutung	Beispiel
.	Der Punkt steht für ein beliebiges Zeichen ungleich Punkt.	`en.e` Treffer z.B.: Ente, Ende, denke
+	Das Pluszeichen steht für ein oder mehrmaliges Vorkommen des davor stehenden Zeichens.	`e+` Treffer z.B. Redner, Seenot, seeeeehr gut
*****	Das Sternzeichen steht für kein, ein- oder mehrmaliges Vorkommen des davorstehenden Zeichens.	`se*` Treffer z.B. Sturm, sehr gut, Seenot
?	Das Fragezeichen steht für kein oder einmaliges Vorkommen des davorstehenden Zeichens.	`se?` Treffer z.B. Sturm, sehr gut aber nicht: Seenot
^	Das Dach- oder Hütchensymbol kann eine Zeichenklasse verneinen oder bei Zeichenketten angeben, dass das nachfolgende Suchmuster am Anfang des Suchbereichs vorkommen muss.	`^Hans`

		Treffer z.B. Hans Meier, Hans Müller aber nicht: Meier Hans `^[^äöüÄÖÜ]*$` Treffer z.B. Meier aber nicht: Müller
$	Das Dollarzeichen gibt bei Zeichenketten an, dass das voranstehende Suchmuster am Ende des Suchbereichs vorkommen muss.	`Hans$` Treffer z.B. Meier Hans aber nicht: Hans Meier
|	Der Senkrechtstrich trennt alternative Ausdrücke.	`[M|m]eier` Treffer z.B. Meier, meier aber nicht: Beier, eier
****	Der Gegenschrägstrich maskiert das nachfolgende (Sonder-)Zeichen.	`Schif\?` Treffer bei Schif? aber nicht: Schiff
[ ]	Eckige Klammern begrenzen eine Zeichenklasse. .	`Me[ijy]er` Treffer z.B. Meier, Mejer, Meyer aber nicht: Meister
-	Der Bindestrich trennt Bereichsgrenzen einer Zeichenklasse.	`Me[a-z]er` Treffer z.B. Meier, Mener, Mezer aber nicht: Meister
( )	Runde Klammern gruppieren Teilausdrücke. .	`(Herr|Frau) M[a-z]+` Treffer bei Herr Meier, Frau Müller aber nicht: Freifrau Meier
{ }	Geschweifte Klammern sind eine Wiederholungsangabe für davorstehende Zeichen.	`Schif{2,5}` Treffer bei Schiff, Schifffahrt aber nicht: Schif

Zeichenwiederholungen

Einfache Zeichenwiederholungen prüfen Sie mit den Sonderzeichen +, * oder ?, siehe vorangehende Tabelle.

Mehrfache Zeichenwiederholungen können Sie auch mit der geschweiften Klammer prüfen: **{m,n}**. Dabei gibt **m** die Mindestzahl, **n** die Maximalzehl der Wiederholungen an.

Folgende Angaben sind erlaubt:

{m}	Wiederholung genau m-mal
{m,}	Wiederholung mindestens m-mal
{m,n}	Wiederholung mindestens m-mal, jedoch nicht mehr als n-mal

`f{1,3}` liefert z.B. Treffer bei Schilf, Schiff und Schifffahrt.

Gruppierungen

Gruppierungen werden durch runde Klammern gebildet. Das nachfolgende Wiederholungszeichen bezieht sich dann auf den gesamten geklammerten Ausdruck.

`h(al)+lo` liefert z.B. Treffer bei hallo, halallo, halalallo.

Auswahl von Zeichen

Eine Liste von Zeichen in eckigen Klammern bietet eine Auswahl von Zeichen an, auf die der reguläre Ausdruck passen kann. Der Ausdruck in eckigen Klammern steht nur für ein Zeichen aus der Liste.

`Me[ijy]er` liefert z.B. Treffer bei Meier, Mejerer, Hintermeyer, nicht aber bei Mayer.

Um eine Auswahl aus einem Ziffernbereich oder einem Bereich des Alphabets treffen verwenden Sie den Bindestrich „-“.

`[A-Z][a-z]+[0-9]{2}` liefert Treffer bei Wörtern, die mit einem Großbuchstaben beginnen, gefolgt von einem oder mehreren Kleinbuchstaben und abgeschlossen mit genau zwei Ziffern, z.B. Meierlein15, Huber01, nicht aber bei meier15, Huber1.

Alternativen

Mit dem Senkrechtstrich „|“ können Sie in einem regulären Ausdruck mehrere alternative Zeichenketten angeben, auf die eine Zeichenkette untersucht werden soll.

`([H|h]err|[F|f]rau] M[a-z]*` liefert Treffer bei allen Anreden von Personen, deren Name mit M beginnt, z. B. Herr Meier, Freifrau Müller.

Maskieren von Sonderzeichen

Sonderzeichen müssen Sie maskieren, wenn Sie nicht die Sonderbedeutung des Zeichens meinen, sondern seine literale, normale Bedeutung, also einen Senkrechtstrich als Senkrechtstrich oder einen Punkt als Punkt meinen. Das Maskierungszeichen ist in allen Fällen der Gegenschrägstrich „\“.

`([A-Z]|[a-z])+\.([A-Z]|[a-z])+@fujitsu\.com` liefert Treffer bei allen E-Mail-Adressen der Form: `vorname.nachname@fujitsu.com`.

`[A-Z]+\.[a-z]+@fujitsu\.com` liefert das gleiche Ergebnis, wenn sie in der Flag-Klausel `'i'` angeben, also Groß- und Kleinschreibung ignorieren möchten.

Operatoren

Buchstaben, denen ein Gegenschrägstrich „\“ vorangestellt wird, kennzeichnen Sonderzeichen oder bestimmte Zeichenklassen:

\n	eines der NEWLINE-Zeichen, siehe "CSV() - BS2000-Datei als Tabelle lesen"
\t	Tabulatorzeichen
\f	FORM FEED-Zeichen
\r	CARRIAGE RETURN-Zeichen
\s	Leerzeichen, Tabulatorzeichen, NEWLINE-Zeichen, CARRIAGE RETURN-Zeichen, FORM FEED-Zeichen
\S	alle Zeichen außer Leerzeichen, Tabulatorzeichen, NEWLINE-Zeichen, CARRIAGE RETURN-Zeichen, FORM FEED-Zeichen
\d	eine Ziffer
\D	ein beliebiges Zeichen, das keine Ziffer ist
\w	ein Wortzeichen, also A bis Z, a bis z und der Unterstrich „_“
\W	ein beliebiges Zeichen, das kein Wortzeichen ist
\A	Anfang einer Zeichenkette
\Z	Ende einer Zeichenkette
\b	Wortgrenze, d.h. bei Angabe von \b... bzw. ...\b führt ein Muster nur dann zu einem Treffer, wenn es am Wortanfang bzw. am Wortende steht.
\B	negative Wortgrenze, d.h. bei Angabe von \b... bzw. ...\b führt ein Muster nur dann zu einem Treffer, wenn es nicht am Wortanfang bzw. am Wortende steht.

Z.B. liefert `\d{3,4}` Treffer bei allen 3- oder 4-stelligen Ziffern und
`\w{5}` liefert Treffer bei allen 5-stelligen Wörtern

Rangfolge in regulären Ausdrücken

Die Sonderzeichen innerhalb von regulären Ausdrücken werden nach einer bestimmten Rangfolge bewertet.

1. Rangstufe: `(  )` (Klammerung)

2. Rangstufe: `+ * ? {m,n}` (Wiederholungsoperatoren)

3. Rangstufe: `abc ^ $ \b \B` (Zeichen/Zeichenketten, Zeilenanfang/-ende, Wortanfang/-ende)

4. Rangstufe: `|` (Alternativen)

Dadurch kann jeder reguläre Ausdruck eindeutig bewertet werden. Wenn Sie in einem Ausdruck jedoch anders bewerten möchten, als es nach der Rangfolge geschieht, können Sie innerhalb des Ausdrucks Klammern setzen, um eine andere Bewertung zu erzwingen.

Z.B. liefert `a|bc|d` Treffer bei 'a' oder 'bc' oder 'd'.
`(a|b)(c|d)` liefert Treffer bei 'ac' oder 'ad' oder 'bc' oder 'bd'.

Hinweise

- Führende oder nachfolgende Leerzeichen müssen evtl. mit `\s*` im Muster berücksichtigt werden. Inbesondere bei Angabe von $ (Ende des Suchbereichs) werden sonst mögliche Treffer nicht erkannt.

  Beispiel

  Beim Datentyp CHAR(n) wird z.B die Zeichenfolge Berta*bbbb* (*b* repräsentiert ein Leerzeichen) mit dem Muster `B.*ta$` nicht erkannt, da noch Blanks folgen.
- Beim Prädikat LIKE bedeutet ein Muster `Ber%` dass ein Trefferwert auch wirklich mit `Ber` beginnt, während das gleiche Muster beim Prädikat LIKE REGEX auch an beliegbiger Stelle im Satz beginnen darf. Mit dem Muster `^Ber.*` wird erreicht, dass das Muster am Satzanfang steht.

5.3.8 NULL-Prädikat (Vergleich auf NULL-Wert)

Es wird geprüft, ob ein Ausdruck den NULL-Wert enthält.

operand `IS [NOT] NULL`

operand `::=` *ausdruck*

operand

Operand für den Vergleich. Der Wert von *operand* muss entweder einfach sein oder der Name einer multiplen Spalte. Ist der Operand eine multiple Spalte, darf die Spaltenangabe keine Außenreferenz, also nicht Spalte eines übergeordneten Abfrage-Ausdrucks sein.

Ergebnis

operand einfach:

Ohne NOT:

Wahr, wenn der Wert von *operand* der NULL-Wert ist.

Falsch, sonst.

Mit NOT:

Wahr, wenn der Wert von *operand* nicht der NULL-Wert ist.

Falsch, sonst.

operand multiple Spalte:

Ohne NOT:

Wahr, wenn mindestens eine Ausprägung der multiplen Spalte der NULL-Wert ist.

Falsch, sonst.

Mit NOT:

Wahr, wenn mindestens eine Ausprägung der multiplen Spalte nicht der NULL-Wert ist.

Falsch, sonst.

Beispiele

```
sprache1 IS NOT NULL
```

`SPRACHE1` im Beispiel ist eine einfache Spalte. Enthält `SPRACHE1` keinen NULL-Wert, dann ergibt der Vergleich wahr. Der Vergleich `NOT sprache1 IS NULL` liefert in diesem Fall denselben Wahrheitswert.

SPRACHE2(1..5) ist eine multiple Spalte, die nur in einigen, aber nicht in allen Spalten den NULL-Wert enthält. Der Vergleich `sprache2(1..5) IS NOT NULL` ergibt in diesem Fall wahr, während `NOT (sprache(1..5) IS NULL)` den Wahrheitswert falsch ergibt.

spalte IS NOT NULL und NOT (*spalte* IS NULL) sind also nicht äquivalent, wenn *spalte* eine multiple Spalte ist. Dies wird deutlich, wenn man `sprache2(1..5) IS NOT NULL` darstellt als:

```
sprache2(1) IS NOT NULL OR sprache2(2) IS NOT NULL OR ...

sprache2(5) IS NOT NULL
```

Der Vergleich ergibt wahr, falls mindestens eine Ausprägung von SPRACHE2 ungleich NULL ist.

`NOT (sprache(1..5) IS NULL)` lässt sich dagegen darstellen als:

```
NOT (sprache(1) IS NULL OR sprache(2) IS NULL ... OR sprache(5) IS NULL)
```

Dieser Vergleich ergibt wahr, wenn die Vergleiche auf den NULL-Wert in der Klammer nach NOT falsch ergeben, d.h. falls alle Ausprägungen von SPRACHE2 ungleich NULL sind.

Aus der Tabelle AUFTRAG die Aufträge mit Auftragstext und Soll-Termin heraussuchen, die noch nicht fertiggestellt sind, d.h. für die der Ist-Termin der NULL-Wert ist.

```
SELECT anr, atext, fertigsoll FROM auftrag WHERE fertigist IS NULL
   anr  atext                  fertigsoll
   250  Serienbrief-Einweisung  <date>
   251  Kunden-Verwaltung       <date>
   300  Netzwerk-Test/Vergleich
   305  Mitarbeiterschulung     <date>
```

5.3.9 EXISTS-Prädikat (Existenzabfrage)

Es wird geprüft, ob eine Ergebnistabelle leer ist.

`EXISTS` *unterabfrage*

unterabfrage

Unterabfrage, die eine Ergebnistabelle liefert.

Ergebnis

Wahr, wenn die Ergebnistabelle nicht leer ist.

Falsch, wenn die Ergebnistabelle leer ist.

Beispiel

Aus der Tabelle KUNDE alle Kunden heraussuchen, die keinen Auftrag vergeben haben:

```
SELECT firma FROM kunde
   WHERE NOT EXISTS (SELECT anr FROM auftrag
                     WHERE auftrag.knr = kunde.knr)
   firma
   Siemens AG
   Plenzer Trading
   Jonas Fischladen
   Externa & Co Kg
```

5.4 Suchbedingung

Suchbedingungen dienen dazu, die Menge der Sätze einzuschränken, die von einer Tabellenoperation oder SQL-Anweisung einer Routine betroffen sind. Es werden nur die Sätze berücksichtigt, die die angegebene Suchbedingung erfüllen. Sie können Suchbedingungen angeben beim Löschen (DELETE), Einfügen/Ändern (MERGE), Ändern (UPDATE) und Auswählen von Sätzen (SELECT), beim Verbinden von Tabellen (Join-Ausdruck) und in einem bedingten Ausdruck (CASE-Ausdruck). In Tabellen- und Spaltenbedingungen können Sie Suchbedingungen angeben, um Integritätsbedingungen zu formulieren. Auch bei AAnweisungen in Routinen treten Suchbedingungen auf.

Sie formulieren die Suchbedingung in einer WHERE-, HAVING-, ON-, CHECK- oder WHEN-Klausel oder einer Kontrollanweisung einer Routine, die in folgenden Anweisungen und Ausdrücken bzw. Abfrage-Ausdrücken vorkommen können:

- WHERE-Klausel
  - DELETE-Anweisung
  - SELECT-Anweisung
  - SELECT-Ausdruck bei CREATE VIEW, DECLARE, INSERT
  - UPDATE-Anweisung
- HAVING-Klausel
  - SELECT-Anweisung
  - SELECT-Ausdruck bei CREATE VIEW, DECLARE, INSERT
- ON-Klausel
  - MERGE-Anweisung
  - Join-Ausdruck
- CHECK-Bedingung in der CREATE TABLE- oder ALTER TABLE-Anweisung
- WHEN-Klausel in einem CASE-Ausdruck mit Suchbedingung
- IF-, CASE-, REPEAT- oder WHILE-Anweisung in einer Routine

Eine Suchbedingung besteht aus Prädikaten und eventuell logischen Operatoren. Die Prädikate sind die Operanden der logischen Operatoren.

Zur Auswertung der Suchbedingung werden die Operatoren auf die Ergebnisse der Operanden angewendet. Das Ergebnis ist einer der Wahrheitswerte wahr, falsch oder unbestimmt.

Die Reihenfolge der Auswertung der Operanden ist nicht festgelegt. In bestimmten Fällen wird ein Operand nicht berechnet, wenn er für die Berechnung des Gesamtergebnisses nicht benötigt wird.

---

suchbedingung `::= {`

praedikat |

suchbedingung `{ AND | OR }` *suchbedingung* |

`NOT` *suchbedingung* |

(*suchbedingung* **)**

}

---

praedikat

Prädikat.

AND

Logisches UND.

Ergebnis für Op1 AND Op2

	Op1 wahr	**Op1 falsch**	**Op1 unbestimmt**
Op2 wahr	wahr	falsch	unbestimmt
Op2 falsch	falsch	falsch	falsch
Op2 unbestimmt	unbestimmt	falsch	unbestimmt

Tabelle 20: logischer Operator AND

OR

Logisches ODER.

Ergebnis für Op1 OR Op2

	Op1 wahr	**Op1 falsch**	**Op1 unbestimmt**
Op2 wahr	wahr	wahr	wahr
Op2 falsch	wahr	falsch	unbestimmt
Op2 unbestimmt	wahr	unbestimmt	unbestimmt

Tabelle 21: logischer Operator OR

NOT

Negation.

Ergebnis

	NOT Op
Op wahr	falsch
Op falsch	wahr
Op unbestimmt	unbestimmt

Tabelle 22: logischer Operator NOT

Prioritäten

- Klammerausdrücke haben die höchste Priorität.

- NOT hat Vorrang vor AND und OR.
- AND hat Vorrang vor OR.
- Operatoren gleicher Priorität werden von links nach rechts angewendet.

Beispiele

Aus den Tabellen AUFTRAG und KUNDE alle Aufträge mit zugehöriger Firma heraussuchen, die nach dem angegebenen Datum gestellt wurden.

```
SELECT a.anr, k.firma, a.atext, a.adatum
   FROM auftrag a, kunde k
   WHERE a.adatum > DATE '<date>' AND a.knr = k.knr
   anr  firma            atext                    adatum
   250  Pudelshop Anke   Serienbrief-Einweisung   <date>
   251  Pudelshop Anke   Kunden-Verwaltung        <date>
   300  Login GmbH       Netzwerk-Test/Vergleich  <date>
   305  Pudelshop Anke   Mitarbeiterschulung      <date>
```

Aus der Tabelle ARTIKEL alle Artikel löschen, deren Preis kleiner als 500.00 ist und deren Artikelbezeichnung aus dem Buchstaben H und einer beliebigen Zeichenfolge besteht.

```
DELETE FROM artikel WHERE preis < 500.00 AND artbez LIKE 'H%'
```

Aus der Tabelle LEISTUNG alle Aufträge heraussuchen, die im angegebenen Zeitraum durchgeführt wurden oder bei denen keine Schulungen durchgeführt bzw. kein Schulungsmaterial und kein Handbuch erstellt wurde.

```
SELECT lnr, ldatum, ltext FROM leistung
   WHERE ldatum BETWEEN DATE '2013-04-01' AND DATE '2013-04-30>'
   OR ltext NOT IN('Schulung','Schulungsmaterial','Handbuch')
   lnr     anr    ldatum       ltext
    1      200    2013-04-19   Schulungsmaterial
    2      200    2013-04-22   Schulung
    3      200    2013-04-23   Schulung
    4      211    2013-01-20   Systemanalyse
    5      211    2013-01-28   Datenbankentwurf
    6      211    2013-02-15   Kopien/Folien
   10      250    2013-02-21   Reisekosten
```

5.5 CASE-Ausdruck

Ein CASE-Ausdruck ist ein bedingter Ausdruck, also ein Ausdruck, der Bedingungen enthält. Jeder Bedingung ist ein bestimmter Ausdruck bzw. der NULL-Wert zugeordnet. Bei der Auswertung des CASE-Ausdrucks wird der zugeordnete Ausdrucks- bzw. NULL-Wert derjenigen Bedingung zurückgeliefert, die wahr ist.

Es gibt verschiedene Varianten des CASE-Ausdrucks:

- CASE-Ausdruck mit Suchbedingung
- Einfacher CASE-Ausdruck
- CASE-Ausdruck mit NULLIF
- CASE-Ausdruck mit COALESCE
- CASE-Ausdruck mit MIN oder MAX

Die Syntax der verschiedenen Varianten ist in der folgenden Übersicht zusammengestellt:

case_ausdruck ::=

```
{
    CASE
    WHEN suchbedingung THEN
    ...
    [ELSE { ausdruck | NULL }]
    END |

    CASE ausdruckx
    WHEN ausdruck1 [, ausdruck2 ] ... THEN { ausdruck | NULL }
    ...
    [ELSE { ausdruck | NULL }]
    END |

    NULLIF ( ausdruck1 , ausdruck2 ) |

    COALESCE ( ausdruck1 , ausdruck2, ... ausdruckn ) |

    { MIN | MAX }( ausdruck1,ausdruck2, ..., ausdruckn )
}
```

Nachfolgend sind die Varianten des CASE-Ausdrucks beschrieben.

Daneben gibt es auch die SQL-Anweisung CASE in Routinen, siehe Abschnitt „CASE - SQL-Anweisungen bedingt ausführen“.

5.5.1 CASE-Ausdruck mit Suchbedingung

Ein CASE-Ausdruck mit Suchbedingung hat folgende Syntax:

case_ausdruck ::=

CASE

WHEN *suchbedingung* THEN { *ausdruck* | NULL }

...

[ELSE { *ausdruck* | NULL }]

END

suchbedingung

Suchbedingung, deren Auswertung einen Wahrheitswert ergibt.

ausdruck

Ausdruck, dessen Auswertung einen alphanumerischen Wert, einen National-Wert, einen numerischen Wert oder einen Zeitwert ergibt. Es darf kein multipler Wert mit Dimension > 1 sein.

ausdruck muss in der THEN-Klausel oder in der ELSE-Klausel oder in beiden Klauseln enthalten sein.

Die Datentypen der Werte von *ausdruck* in den THEN-Klauseln und in der ELSE-Klausel müssen verträglich sein (siehe Abschnitt „Verträglichkeit von Datentypen“).

Ergebnis

Das Ergebnis des CASE-Ausdrucks ist der Inhalt derjenigen THEN-Klausel, deren zugehörigee *suchbedingung* als Erste den Wahrheitswert wahr ergibt. Der Inhalt der THEN-Klausel ist der Wert des der THEN-Klausel zugeordneten *ausdruck* bzw. der NULL-Wert. Die WHEN-Klauseln werden von links nach rechts abgearbeitet.

Ergibt keine *suchbedingung* den Wahrheitswert wahr, so ist das Ergebnis der Inhalt der ELSE-Klausel, also der Wert des der ELSE-Klausel zugeordneten *ausdruck* bzw. der NULL-Wert. Wenn Sie die ELSE-Klausel nicht angeben, gilt die Voreinstellung NULL.

Der Datentyp eines CASE-Ausdrucks mit Suchbedingung ergibt sich aus den Datentypen der Werte von *ausdruck*, die in den THEN-Klauseln und der ELSE-Klausel enthalten sind, wie folgt:

- Jeder *ausdruck* hat Datentyp CHAR bzw. NCHAR:Der Wert des CASE-Ausdrucks hat Datentyp CHAR bzw. NCHAR mit der größten Länge..
- Mindestens ein Wert von *ausdruck* hat Datentyp VARCHAR bzw. NVARCHAR:Der Wert des CASE-Ausdrucks hat Datentyp VARCHAR bzw. NVARCHAR mit der größten bzw. größten maximalen Länge.
- Jeder *ausdruck* hat ganzzahligen Typ oder Festpunktzahl-Typ (INT, SMALLINT, NUMERIC, DEC):Der Wert des CASE-Ausdrucks hat Datentyp Ganz- oder Festpunktzahl.

- Die Nachkommastellenzahl ist die größte der Nachkommastellenzahlen der verschiedenen Werte von *ausdruck*.
- Die Gesamtstellenzahl ist die größte der Vorkommastellenzahlen plus die größte der Nachkommastellenzahlen der verschiedenen Werte von *ausdruck*, höchstens jedoch 31.

- Mindestens ein Wert von *ausdruck* hat Gleitpunktzahl-Typ (REAL, DOUBLE PRECISION, FLOAT), die anderen haben einen beliebigen numerischen Datentyp:Der Wert des CASE-Ausdrucks hat den Datentyp DOUBLE PRECISION.
- Jeder *ausdruck* hat Zeitdatentyp: Alle Werte müssen denselben Zeitdatentyp haben und der Wert des CASE-Ausdrucks hat auch diesen Datentyp.

Beispiel

In einer Liste werden die Artikel aus der Tabelle ARTIKEL danach sortiert, wie dringend sie bestellt werden müssen.

```
SELECT artnr, artbez,
   CASE
       WHEN bestand > minbestand THEN 'O.K.'
       WHEN bestand = minbestand THEN 'bald bestellen'
       WHEN bestand > minbestand * 0.5 THEN 'bestellen'
       ELSE 'dringend bestellen'
   END
FROM artikel
```

5.5.2 Einfacher CASE-Ausdruck

Ein einfacher CASE-Ausdruck hat folgende Syntax:

```
case_ausdruck ::=

CASE ausdruckx

     WHEN ausdruck1 [, ausdruck2 ] ... THEN { ausdruck | NULL }

     ...

     [ELSE { ausdruck | NULL }]

END
```

ausdruck

Ausdruck, dessen Auswertung einen alphanumerischen Wert, einen National-Wert, einen numerischen Wert oder einen Zeitwert ergibt.
Es darf kein multipler Wert mit Dimension > 1 sein.

Die Werte von *ausdruckx* und *ausdruck1... ausdruckn* müssen verträgliche Datentypen haben (siehe Abschnitt „Verträglichkeit von Datentypen“).

ausdruck muss in der THEN- Klausel oder in der ELSE-Klausel oder in beiden Klauseln enthalten sein.

Die Datentypen der Werte von *ausdruck* in den THEN-Klauseln und in der ELSE-Klausel müssen verträglich sein (siehe Abschnitt „Verträglichkeit von Datentypen“).

Ergebnis

Der Wert von *ausdruckx* nach CASE wird (von links nach rechts) verglichen mit den Werten der in der WHEN-Klausel enthaltenen Ausdrücke *ausdruck1*, *ausdruck2*, ... Bei der ersten Gleichheit ist das Ergebnis des CASE-Ausdrucks der Inhalt der zugehörigen THEN-Klausel, also der Wert des zugeordneten *ausdruck* bzw. der NULL-Wert. Enthält der CASE-Ausdruck mehrere WHEN-Klauseln, so ist das Ergebnis der Inhalt der ersten THEN-Klausel, in deren zugehöriger WHEN-Klausel ein Ausdruck gefunden wurde, für den Gleichheit mit *ausdruckx* festgestellt wird. Die WHEN-Klauseln werden von oben nach unten abgearbeitet.

Wird für keinen der in den WHEN-Klauseln enthaltenen Ausdrücke *ausdruck1... ausdruckn* Gleichheit mit *ausdruckx* festgestellt, so ist das Ergebnis der Inhalt der ELSE-Klausel, also der Wert des der ELSE-Klausel zugeordneten *ausdruck* bzw. der NULL-Wert. Wenn Sie die ELSE-Klausel nicht angeben, gilt die Voreinstellung NULL.

Der Datentyp eines einfachen CASE-Ausdrucks ergibt sich aus den Datentypen der Werte von *ausdruck*, die in den THEN-Klauseln und der ELSE-Klausel enthalten sind. Es gelten dieselben Regeln, wie für den Datentyp eines CASE-Ausdrucks mit Suchbedingung (siehe "CASE-Ausdruck mit Suchbedingung").

Ein einfacher CASE-Ausdruck entspricht einem CASE-Ausdruck mit Suchbedingung der folgenden Form:

```
CASE
   WHEN ausdruckx=ausdruck1 THEN {ausdruck|NULL}
   WHEN ausdruckx=ausdruck2 THEN {ausdruck|NULL}
   ...
   WHEN ausdruckx=ausdruckn THEN {ausdruck|NULL}
   ELSE {ausdruck|NULL}
END
```

Beispiele

Die Firmen aus der Tabelle KUNDE werden in einer Liste danach sortiert, in welchem Land der Firmensitz ist. Dabei sollen die Länderkennzeichen durch die Namen der Länder ersetzt werden.

```
SELECT firma,
   CASE land
       WHEN '  D' THEN 'Deutschland'
       WHEN 'USA' THEN 'Amerika'
       WHEN ' CH' THEN 'Schweiz'
   END
FROM kunde
```

Für die Lohnbuchhaltung soll eine Unterscheidung nach Werktag und Wochenende getroffen werden.

```
CASE EXTRACT(DAY_OF_WEEK FROM CURRENT_DATE)
     WHEN 1,2,3,4,5 THEN 'Werktag'
     WHEN 6,7 THEN 'Wochenende'
              ELSE '?????'
END
```

5.5.3 CASE-Ausdruck mit NULLIF

Ein CASE-Ausdruck mit NULLIF hat folgende Syntax:

case_ausdruck `::= NULLIF (` *ausdruck1* `,` *ausdruck2* `)`

ausdruck

Ausdruck, dessen Auswertung einen alphanumerischen Wert, einen National-Wert, einen numerischen Wert oder einen Zeitwert ergibt.
Es darf kein multipler Wert mit Dimension > 1 sein.

Ergebnis

Das Ergebnis des CASE-Ausdrucks ist NULL, wenn *ausdruck1* und *ausdruck2* gleich sind. Bei Ungleichheit zwischen *ausdruck1* und *ausdruck2* ist das Ergebnis *ausdruck1*.

Ein CASE-Ausdruck mit NULLIF entspricht einem CASE-Ausdruck mit Suchbedingung der folgenden Form:

```
CASE
   WHEN ausdruck1=ausdruck2 THEN NULL
   ELSE ausdruck1
END
```

Beispiel

Aus der Tabelle LEISTUNG soll die berechnete Mehrwertsteuer ermittelt werden, für den Fall, dass der Mehrwertsteuersatz nicht 0.07 beträgt.

```
SELECT lsatz * NULLIF (mwsatz,0.07) AS steuer FROM leistung
```

5.5.4 CASE-Ausdruck mit COALESCE

Ein CASE-Ausdruck mit COALESCE hat folgende Syntax:

case_ausdruck `::= COALESCE (` *ausdruck1* `,` *ausdruck2* `, ...,` *ausdruckn* `)`

ausdruck

Ausdruck, dessen Auswertung einen alphanumerischen Wert, einen National-Wert, einen numerischen Wert oder einen Zeitwert ergibt.
Es darf kein multipler Wert mit Dimension > 1 sein.

Ergebnis

Das Ergebnis des CASE-Ausdrucks ist NULL, wenn alle in der Klammer enthaltenen Ausdrücke (*ausdruck1... ausdruckn*) NULL ergeben. Ergibt mindestens ein *ausdruck* einen anderen als den NULL-Wert, so ist das Ergebnis des CASE-Ausdrucks der Wert des ersten *ausdruck*, der nicht den NULL-Wert ergibt.

Der CASE-Ausdruck `COALESCE (ausdruck1,ausdruck2)` entspricht einem CASE-Ausdruck mit Suchbedingung der folgenden Form:

```
CASE
   WHEN ausdruck1 IS NOT NULL THEN ausdruck1
   ELSE ausdruck2
END
```

Der CASE-Ausdruck `COALESCE (ausdruck1,ausdruck2,...,ausdruckn)` entspricht folgendem CASE-Ausdruck mit Suchbedingung:

```
CASE
   WHEN ausdruck1 IS NOT NULL THEN ausdruck1
   ELSE COALESCE (ausdruck2 ...,ausdruckn)
END
```

Beispiele

Eine Liste der Ansprechpartner soll für gezielte Kundenkontakte erstellt werden. Neben der Anrede, dem Nachnamen, der Telefonnummer und der Funktion soll entweder die Abteilung eines Kunden, oder, wenn diese nicht bekannt ist, zumindest der Anlass eines früheren Kontakts ermittelt werden.

```
SELECT anrede, nachname, kotelefon, funktion,

COALESCE(abteilung, koinfo) AS info FROM kontakt WHERE konr < 30
```

Ergebnistabelle

anrede	nachname	kotelefon	funktion	info
Herr Dr.	Kuehne	089/6361896	Vorstand	Personal
Herr	Walkers	089/63640182	Sekretaer	Vertrieb
Herr	Loetzerich	089/4488870	Geschaeftsfuehrer	Netzwerke
Herr	Schmidt	0551/123873	Schulung	
Frau	Kredler	089/923764	Organisation	SQL-Kurs

Nach Anrede, Nachname, Telefonnummer und Funktion wird zunächst die Abteilung eines Kunden ermittelt. Fehlt diese Information (NULL-Wert), wird für INFO der Spaltenwert für die Spalte KOINFO ermittelt. Enthält sowohl die Spalte ABTEILUNG als auch die Spalte KOINFO den NULL-Wert, dann ergibt sich auch der NULL-Wert für INFO.

Aus der Tabelle AUFTRAG soll eine Liste mit Auftragsterminen erstellt werden. Die Liste soll das Datum der Auftragserteilung, die Bezeichnung des Auftrags und einen Fertigungstermin enthalten. Ist der tatsächliche Fertigungstermin nicht bekannt, soll statt dessen der Soll-Termin eingetragen werden.

```
SELECT adatum, atext,
   COALESCE (fertigist, fertigsoll) AS termin FROM auftrag
   adatum         atext                     termin
   <date>         Mitarbeiterschulung       <date>
   <date>         Kunden-Verwaltung         <date>
   <date>         Datenbank-Entwurf Kunden  <date>
   <date>         Serienbrief-Einweisung    <date>
   <date>         Kunden-Verwaltung         <date>
   <date>         Netzwerk-Test/Vergleich
   <date>         Mitarbeiterschulung       <date>
```

Um die Werte für TERMIN zu ermitteln, wird zunächst die Spalte FERTIGIST ausgewertet. Ist ein Datum als Spaltenwert vorhanden, wird dieses übernommen. Wo FERTIGIST den NULL-Wert enthält, wird der entsprechende Spaltenwert aus der Spalte FERTIGSOLL ermittelt und in die Spalte TERMIN eingetragen. Enthält sowohl FERTIGIST als auch FERTIGSOLL den NULL-Wert, dann wird in die Spalte TERMIN der NULL-Wert eingetragen.

5.5.5 CASE-Ausdruck mit MIN / MAX

Ein CASE-Ausdruck mit MIN / MAX hat folgende Syntax:

case_ausdruck `::= { MIN | MAX }(` *ausdruck1,ausdruck2,* `...,` *ausdruckn* `)`

ausdruck

Ausdruck, dessen Auswertung einen alphanumerischen Wert, einen National-Wert, einen numerischen Wert oder einen Zeitwert ergibt.
Es darf kein multipler Wert mit Dimension > 1 sein.

Die Werte von *ausdruck1,ausdruck2,...,ausdruckn* müssen verträgliche Datentypen haben (siehe Abschnitt „Verträglichkeit von Datentypen“).

> **i** Ein CASE-Ausdruck mit MIN bzw. MAX bezieht sich auf unterschiedliche Ausdrücke. Er unterscheidet sich dadurch von den Mengenfunktionen MIN() und MAX() (siehe "Mengenfunktionen"), die sich auf die Menge aller Werte einer Spalte in einer Tabelle beziehen.

Ergebnis

Das Ergebnis des CASE-Ausdrucks ist der NULL-Wert, wenn wenigstens einer der in der Klammer enthaltenen Ausdrücke (*ausdruck1,ausdruck2,...,ausdruckn*) den NULL-Wert ergibt.

Ergibt kein *ausdruck* den NULL-Wert, so ist das Ergebnis des CASE-Ausdrucks bei Angabe von MIN der Wert des kleinsten *ausdruck*, bei Angabe von MAX der Wert des größten *ausdruck*.

Der CASE-Ausdruck `MIN(ausdruck1,ausdruck2)` entspricht einem CASE-Ausdruck mit Suchbedingung der folgenden Form:

```
CASE
   WHEN ausdruck1 <= ausdruck2 THEN ausdruck1
   ELSE ausdruck2
END
```

Der CASE-Ausdruck `MIN(ausdruck1,ausdruck2,...,ausdruckn)` entspricht dem CASE-Ausdruck `MIN(MIN(ausdruck1,ausdruck2,...),ausdruckn)`.

Der CASE-Ausdruck `MAX(ausdruck1,ausdruck2)` entspricht einem CASE-Ausdruck mit Suchbedingung der folgenden Form:

```
CASE
   WHEN ausdruck1 >= ausdruck2 THEN ausdruck1
```

```
   ELSE ausdruck2
END
```

Der CASE-Ausdruck `MAX(ausdruck1,ausdruck2,...,ausdruckn)` entspricht dem CASE-Ausdruck: `MAX(MAX(ausdruck1,ausdruck2,...),ausdruckn)`.

Beispiel

Folgendes Beispiel wählt alle Einträge der Tabelle `umsatz` seit dem mit der Benutzervariablen `input_datum` eingegebenen Datum, aber höchstens für die letzten 90 Tage.

```
SELECT * FROM umsatz WHERE umsatz.datum >= MAX(:input_datum,
DATE_OF_JULIAN_DAY(JULIAN_DAY_OF_DATE(CURRENT_DATE)-90))
```

5.6 CAST-Ausdruck

Der CAST-Ausdruck wandelt einen Wert eines bestimmtem Datentyps in einen Wert eines anderen Datentyps um.

cast_ausdruck `::= CAST ({` *ausdruck* `| NULL } AS` *datentyp* `)`

ausdruck / NULL

CAST-Operand. Er enthält das Schlüsselwort NULL oder einen Ausdruck *ausdruck*. Der Wert von *ausdruck* darf kein multipler Wert mit Dimension > 1 sein.

datentyp

Zieldatentyp für das Ergebnis des CAST-Ausdrucks.
Der Zieldatentyp *datentyp* darf keine Dimension für eine multiple Spalte enthalten.

Ergebnis

Das Ergebnis des CAST-Ausdrucks ist ein atomarer Wert des Zieldatentyps *datentyp*. Welcher Wert zurückgeliefert wird, ist einerseits abhängig vom Wert des CAST-Operanden, andererseits von dessen Datentyp.

Ergibt *ausdruck* den NULL-Wert oder enthält der CAST-Operand das Schlüsselwort NULL, so ist das Ergebnis des CAST-Ausdrucks der NULL-Wert.

Sonst gelten die Regeln für die Konvertierung eines Wertes in einen anderen Datentyp, die ab "CAST-Ausdruck" beschrieben sind.

Kombinationsmöglichkeiten von Ausgangs- und Zieldatentyp

Der Datentyp von *ausdruck*, hier Ausgangsdatentyp genannt, ist nur mit bestimmten Zieldatentypen kombinierbar. Tabelle 23 zeigt, welche Ausgangsdatentypen Sie mit welchen Zieldatentypen kombinieren dürfen und welche Kombinationen nicht zugelassen sind

		Zieldaten-typ	**Zieldaten-typ**	**Zieldaten-typ**	**Ziel-daten-typ**	**Ziel-daten-typ**	**Ziel-daten-typ**	**Zieldaten-typ**
		INTEGER SMALLINT DECIMAL NUMERIC	REAL DOUBLE PRECISION FLOAT	CHAR VAR-CHAR	NCHAR NVAR-CHAR	DATE	TIME(3)	TIME-STAMP(3)
Aus-gangs-daten-typ	INTEGER SMALLINT DECIMAL	ja	ja	ja	ja	nein	nein	nein

	NUMERIC							
Aus-gangs-daten-typ	REAL DOUBLE PRECISION FLOAT	ja	ja	ja	ja	nein	nein	nein
Aus-gangs-daten-typ	CHAR VARCHAR	ja	ja	ja	nein	ja	ja	ja
Aus-gangs-daten-typ	NCHAR NVAR-CHAR	ja	ja	nein	ja	ja	ja	ja
Aus-gangs-daten-typ	DATE	nein	nein	ja	ja	ja	nein	ja
Aus-gangs-daten-typ	TIME(3)	nein	nein	ja	ja	nein	ja	ja
Aus-gangs-daten-typ	TIME-STAMP(3)	nein	nein	ja	ja	ja	ja	ja

Tabelle 23: Zulässige und verbotene Kombinationen von Ausgangs- und Zieldatentyp beim CAST-Ausdruck

Regeln für die Konvertierung eines Wertes in einen anderen Datentyp

Neben den erlaubten Kombinationsmöglichkeiten von Ausgangs- und Zieldatentyp (siehe Tabelle 23), gelten bei der Konvertierung eines Wertes in einen anderen Datentyp weitere Regeln, die im Folgenden beschrieben sind. Abhängig vom Zieldatentyp, ist die Beschreibung in drei Gruppen eingeteilt:

- Zieldatentyp ist ein Datentyp für Ganz-, Festpunkt- oder Gleitpunktzahlen
- Zieldatentyp ist ein Datentyp für Zeichenketten fester oder variabler Länge
- Zieldatentyp ist ein Zeit-Datentyp

Zieldatentyp ist ein Datentyp für Ganz-, Festpunkt- oder Gleitpunktzahlen

- Numerische Werte werden gerundet, wenn ihre Nachkommastellenzahl für den Zieldatentyp zu groß ist. Ist der Betrag des numerischen Wertes für den Zieldatentyp zu groß, erhalten Sie eine Fehlermeldung.

  Beispiele

  ```
  CAST (4502.9267 AS DECIMAL(6,2))
  ```

  Der Wert 4502.9267 wird auf den Wert 4502.93 gerundet.

  ```
  CAST (-115.05 AS DECIMAL(2,0))
  ```

  Der Wert -115.05 wird auf den Wert -115 gerundet. Da aber der Betrag des Wertes für den Zieldatentyp zu groß ist, erfolgt eine Fehlermeldung.

  ```
  CAST (2450.43 AS REAL)
  ```

  Der Wert 2450.43 wird als Gleitpunktzahl des Wertes 2.45043E3 dargestellt.

- Alphanumerische Werte und National-Werte müssen ohne Wertverlust als Wert des zugewiesenen Zieldatentyps darstellbar sein. Führende oder nachfolgende Leerzeichen werden entfernt.

  Beispiele

  ```
  CAST ('512 ' AS SMALLINT) / CAST (N'512 ' AS SMALLINT)
  ```

  Das Leerzeichen am Ende der Zeichenkette wird entfernt. Die Zeichenkette '512' wird als kleine Ganzzahl 512 dargestellt.

  ```
  CAST ('sum' AS NUMERIC)
  ```

  Das ist ein Fehler: Die Zeichenkette 'sum' ist nicht als numerischer Wert darstellbar, da numerische Literale nur Ziffern enthalten dürfen.

  ```
  CAST ('255' AS REAL) / CAST (N'255' AS REAL)
  ```

  Die Leerzeichen am Ende der Zeichenkette werden entfernt und die Zeichenkette '255' wird als Gleitpunktzahl 2.55000E2 dargestellt.

Zieldatentyp ist ein Datentyp für Zeichenketten fester oder variabler Länge

- Numerische Werte des Datentyps Ganzzahl, Festpunktzahl oder Gleitpunktzahl müssen ohne Wertverlust als Zeichenkette fester bzw. variabler Länge darstellbar sein. Werte des Datentyps Gleitpunktzahl müssen zudem in der normierten Form einer Gleitpunktzahl darstellbar sein, wenn ihr Wert ungleich 0 ist, sonst in der Form 0E ^{0}. Für alle numerischen Werte gilt: Ist die Länge des Werts kleiner als die feste Länge des Zieldatentyps CHAR oder NCHAR, wird der Wert am Ende mit Leerzeichen ergänzt; ist die Länge des Werts kleiner als die maximale Länge des Zieldatentyps VARCHAR oder NVARCHAR, wird sie beibehalten. Ist die Länge des Werts größer als die feste bzw. maximale Länge des Zieldatentyps, erhalten Sie eine Fehlermeldung.

  Beispiele

  ```
  CAST (1234 AS CHAR(5)) / CAST (1234 AS NCHAR(5))
  ```

  Der Wert der Ganzzahl 1234 ergibt die alphanumerische Zeichenkette '1234 ' bzw. die National-Zeichenkette N'1234 '.

  ```
  CAST (25.95 AS VARCHAR(5)) / CAST (25.95 AS NVARCHAR(5))
  ```

  Der Wert der Festpunktzahl 25.95 ergibt die alphanumerische Zeichenkette '25.95' bzw. die National-Zeichenkette N'25.95'.

  ```
  CAST (45.5E2 AS CHAR(7)) / CAST (45.5E2 AS NCHAR(7))
  ```

  Der Wert der Gleitpunktzahl 45.5E2 ergibt die alphanumerische Zeichenkette '4.55E3 ' bzw. die National-Zeichenkette N'4.55E3 '.

- Alphanumerische Werte und National-Werte werden am Ende mit Leerzeichen ergänzt, wenn ihre Länge kleiner als die feste Länge des Zieldatentyps CHAR oder NCHAR ist. Ist die Länge des Werts kleiner als die maximale Länge des Zieldatentyps VARCHAR oder NVARCHAR, wird sie beibehalten. Ist die Länge des Wertes größer als die feste bzw. maximale Länge des Zieldatentyps, wird der Wert am Ende auf die Länge des Zieldatentyps gekürzt. Werden Zeichen entfernt, die keine Leerzeichen sind, erhalten Sie eine Warnung.

  Beispiele

  ```
  CAST ('Wochenende' AS CHAR(5)) / CAST (N'Wochenende' AS NCHAR(5))
  ```

  Die Zeichenkette 'Wochenende' ist für den Datentyp CHAR(5) bzw. NCHAR(5) zu lang. Sie wird am Ende auf die Länge der Ziel-Zeichenkette, also 'Woche', gekürzt und SESAM/SQL gibt eine Warnung aus.

  ```
  CAST ('Woche' AS VARCHAR(15)) / CAST (N'Woche' AS NVARCHAR(15))
  ```

Ergebnis ist die alphanumerische Zeichenkette 'Woche' bzw. die National-Zeichenkette N'Woche'. Die Zeichenkette wird nicht am Ende bis zur maximalen Länge von 15 Zeichen aufgefüllt.

- Zeitwerte müssen als Zeichenkette darstellbar sein. Ist die Länge des Zeitwerts kleiner als die feste Länge des Zieldatentyps CHAR oder NCHAR, so wird der Wert am Ende mit Leerzeichen ergänzt. Ist die Länge des Zeitwerts kleiner als die maximale Länge des Zieldatentyps VARCHAR oder NVARCHAR, wird sie beibehalten. Ist sie größer als die feste bzw. variable Länge des Zieldatentyps, erhalten Sie eine Fehlermeldung.

  Beispiele

  ```
  CAST (DATE'2013-08-11' AS VARCHAR(20))
  ```

  ```
  CAST (DATE'2013-08-11' AS NVARCHAR(20))
  ```

  Ergebnis ist die alphanumerische Zeichenkette '2013-08-11' bzw. die National-Zeichenkette N'2013-08-11'.

  ```
  CAST (DATE'2013-08-11' AS VARCHAR(5))
  ```

  Der Zeitwert ist für eine Zeichenkette mit der maximalen variablen Länge 5 zu lang. Der Zeitwert wird nicht konvertiert und es erfolgt eine Fehlermeldung.

Zieldatentyp ist ein Zeit-Datentyp

- Alphanumerische Werte und National-Werte müssen ohne Wertverlust als Wert des zugewiesenen Zieldatentyps darstellbar sein. Führende oder nachfolgende Leerzeichen werden entfernt.

  Beispiele

  ```
  CAST ('2013-08-11' AS DATE)
  ```

  ```
  CAST (N'2013-08-11' AS DATE)
  ```

  Das führende Leerzeichen der Zeichenkette wird entfernt und die Zeichenkette wird in den Datentyp DATE konvertiert.

  ```
  CAST ('2013-08-11 17:57:35:000' AS TIMESTAMP(3))
  ```

  Das ist ein Fehler: Die Zeichenkette ist nicht als Zeitstempel darstellbar. Das Trennzeichen zwischen den Komponenten Sekunde und Sekundenbruchteile bei Zeitstempel-Werten muss ein Punkt „.“ sein.
- Bei der Konvertierung von Zeitwerten gelten folgende Regeln:
  - Ist der Zieldatentyp DATE und der Ausgangsdatentyp TIMESTAMP, enthält der Ergebniswert die Datumsangabe (Jahr-Monat-Tag) des Ausgangswertes.
  - Ist der Zieldatentyp DATE und der Ausgangsdatentyp TIME, erhalten Sie eine Fehlermeldung.
  - Ist der Zieldatentyp TIME und der Ausgangsdatentyp TIMESTAMP, enthält der Ergebniswert die Uhrzeit (Stunde:Minute:Sekunde) des Ausgangswertes.
  - Ist der Zieldatentyp TIME und der Ausgangsdatentyp DATE, erhalten Sie eine Fehlermeldung.
  - Ist der Zieldatentyp TIMESTAMP und der Ausgangsdatentyp DATE, enthält der Ergebniswert die Datumsangabe Jahr-Monat-Tag des Ausgangswertes und die auf 0 gesetzten Felder Stunde:Minute:Sekunde für die Uhrzeit.
  - Ist der Zieldatentyp TIMESTAMP und der Ausgangsdatentyp TIME, enthält der Ergebniswert die Datumsangabe Jahr-Monat-Tag des aktuellen Datums (CURRENT_DATE) und die Uhrzeit Stunde:Minute: Sekunde des Ausgangswertes.

  Beispiele

  ```
  CAST (TIMESTAMP '2013-08-11 17:57:35.000' AS DATE)
  ```

  Ergebniswert ist das Datum '2013-08-11' .

```
SELECT atext, CAST (fertigist AS TIMESTAMP(3))
   FROM auftrag WHERE knr=106
   atext                          fertigist
   Kunden-Verwaltung              <date>     00:00:00.000
   Datenbank-Entwurf Kunden       <date>     00:00:00.000
```

Die Ergebnistabelle enthält die Spalte FERTIGIST mit dem Datentyp TIMESTAMP. Die Zeitstempel-Felder für die Uhrzeit sind auf 0 gesetzt.

5.7 Integritätsbedingung

Eine Integritätsbedingung ist eine Regel für die erlaubten Inhalte von Sätzen einer Tabelle. Ein Satz kann nur dann in eine Tabelle aufgenommen (INSERT, MERGE) oder aus einer Tabelle gelöscht (DELETE) werden und ein Spaltenwert kann nur geändert (MERGE, UPDATE) werden, wenn danach alle Integritätsbedingungen erfüllt sind.

Integritätsbedingungen dürfen nicht für multiple Spalten definiert werden.

Integritätsbedingungen können für einzelne Spalten oder für eine Tabelle definiert werden. Eine Spaltenbedingung ist eine Integritätsbedingung, die sich auf eine Spalte bezieht. Eine Tabellenbedingung ist eine Integritätsbedingung, die sich auf mehr als eine Spalte der Basistabelle beziehen kann.

Nicht-NULL-Bedingung

Die Nicht-NULL-Bedingung (NOT NULL) fordert, dass eine Spalte keine NULL-Werte enthalten darf. Die Nicht-NULL-Bedingung kann nur als Spaltenbedingung angegeben werden.

Eindeutigkeitsbedingung

Die Eindeutigkeitsbedingung (UNIQUE) fordert, dass in der angegebenen Spalte oder Spaltenkombination kein Wert bzw. keine Wertekombination doppelt vorkommt.

Primärschlüsselbedingung

Die Primärschlüsselbedingung (PRIMARY KEY) legt eine Spalte oder eine Spaltenkombination als Primärschlüssel einer Tabelle fest. Die Primärschlüsselbedingung fordert, dass für die Spalte bzw. die Spaltenkombination die Eindeutigkeitsbedingung und die Nicht-NULL-Bedingung erfüllt sind. Eine Tabelle darf höchstens einen Primärschlüssel haben.

Check-Bedingung

Die Check-Bedingung fordert, dass für jeden Satz einer Tabelle eine angegebene Suchbedingung den Wahrheitswert wahr oder unbestimmt annimmt, nicht jedoch den Wahrheitswert falsch.

Die Suchbedingung darf sich nur auf die Tabelle beziehen, für die die Check-Bedingung definiert wurde.

Referenzbedingung

Die Referenzbedingung ([FOREIGN KEY]..REFERENCES) legt eine Spalte oder Spaltenkombination als Fremdschlüssel einer Tabelle fest. Die Spalten des Fremdschlüssels werden einer oder mehreren Spalten derselben oder einer anderen Tabelle zugeordnet. Diese Spalten werden referenzierte Spalten genannt. Für die referenzierten Spalten muss die Eindeutigkeitsbedingung gelten. Die Tabelle, die den Fremdschlüssel enthält, wird referenzierende Tabelle genannt. Die Tabelle, die zu den referenzierten Spalten gehört, heißt referenzierte Tabelle. Werden keine Spalten für die referenzierte Tabelle angegeben, wird der Primärschlüssel der referenzierten Tabelle verwendet.

SESAM/SQL weist eine Tabellenoperation nach Prüfung der Referenzbedingung ab,

- wenn beim Einfügen eines Satzes oder Ändern von Spaltenwerten in der referenzierenden Tabelle keine entsprechenden Werte in den referenzierten Spalten vorhanden wären
- wenn beim Löschen oder Ändern von Sätzen oder Spalten der referenzierten Tabelle Fremdschlüsselwerte in der referenzierenden Tabelle existierten, für die nun keine entsprechenden Werte in den referenzierten Spalten mehr vorhanden wären oder für die keine entsprechende Spalte mehr existierte.

Bei einspaltigen Fremdschlüsseln fordert die Referenzbedingung, dass jeder vom NULL-Wert verschiedene Wert des Fremdschlüssels einer Tabelle als Wert der referenzierten Spalte vorkommt.

Bei mehrspaltigen Fremdschlüsseln muss jede darin vorkommende Wertekombination, die keinen NULL-Wert enthält, in den referenzierten Spalten auftreten. In SESAM/SQL genügt somit ein Satz schon der Referenzbedingung, wenn er in mindestens einer Spalte des mehrspaltigen Fremdschlüssels einen NULL-Wert enthält.

5.7.1 Spaltenbedingung

Bei der Erzeugung oder Änderung einer Basistabelle (CREATE TABLE, ALTER TABLE) können in der Spaltendefinition für einzelne Spalten Spaltenbedingungen angegeben werden. Die Spalte darf keine multiple Spalte sein.

Eine Spaltenbedingung ist eine Integritätsbedingung, die sich auf eine Spalte bezieht. Alle Werte der Spalte müssen die Integritätsbedingung erfüllen.

Für CALL-DML-Tabellen darf nur die Integritätsbedingung PRIMARY KEY definiert werden.

spaltenbedingung ::=

```
{
   NOT NULL |
   UNIQUE |
   PRIMARY KEY |
   REFERENCES tabelle [( spalte )] |
   CHECK ( suchbedingung)
}
```

NOT NULL

Nicht-NULL-Bedingung.
Die Spalte darf keine NULL-Werte enthalten.

Die Nicht-NULL-Bedingung wird als Check-Bedingung (*spalte* IS NOT NULL) abgespeichert.

UNIQUE

Eindeutigkeitsbedingung.
Die Spaltenwerte, die ungleich dem NULL-Wert sind, müssen eindeutig sein.

Die Spaltenlänge muss die Einschränkungen erfüllen, die für einen Index gelten (siehe CREATE INDEX-Anweisung, "CREATE INDEX - Index erzeugen").

PRIMARY KEY

Primärschlüsselbedingung.
Die Spalte ist der Primärschlüssel der Tabelle. Die Werte der Spalte müssen eindeutig sein. Für jede Tabelle kann nur ein Primärschlüssel definiert werden.

Die Spalte darf nicht vom Datentyp VARCHAR oder NVARCHAR sein. Die Länge der Spalte einer CALL-DML-Tabelle muss zwischen 4 und 256 Zeichen liegen. Für SQL-Tabellen gibt es keine Minimallänge.

Für eine Primärschlüsselspalte gilt implizit auch die Nicht-NULL-Bedingung.

REFERENCES

Referenzbedingung.

Die Spalte der referenzierenden Tabelle darf einen vom NULL-Wert verschiedenen Wert nur dann enthalten, wenn derselbe Wert in der referenzierten Spalte der referenzierten Tabelle enthalten ist.

Der aktuelle Berechtigungsschlüssel muss das Privileg REFERENCES für die referenzierten Spalten besitzen.

tabelle

Name der referenzierten Basistabelle.
Die referenzierte Basistabelle muss eine SQL-Tabelle sein. Der einfache Name der referenzierten Basistabelle kann durch einen Datenbank- und Schemanamen qualifiziert werden. Der Datenbankname muss mit dem Datenbanknamen der referenzierenden Tabelle übereinstimmen.

(*spalte*)

Name der referenzierten Spalte.
Die referenzierte Spalte muss mit UNIQUE oder PRIMARY KEY definiert sein. Die referenzierte Spalte darf keine multiple Spalte sein. Referenzierende Spalte und referenzierte Spalte müssen genau denselben Datentyp besitzen.

(*spalte*) nicht angegeben:
Der Primärschlüssel der referenzierten Tabelle wird als referenzierte Spalte verwendet. Referenzierende Spalte und referenzierte Spalte müssen genau denselben Datentyp besitzen.

CHECK (*suchbedingung*)

Check-Bedingung.
Für jeden Wert der Spalte muss die Suchbedingung *suchbedingung* den Wahrheitswert wahr oder unbestimmt annehmen, nicht jedoch den Wahrheitswert falsch.

Für *suchbedingung* gelten folgende Einschränkungen:

- *suchbedingung* darf keine Benutzervariablen enthalten.
- *suchbedingung* darf keine Mengenfunktion enthalten.
- *suchbedingung* darf keine Unterabfrage enthalten, also kann sich *suchbedingung* nur auf die Spalte der Tabelle beziehen, zu der die Spaltenbedingung gehört.
- *suchbedingung* darf keine Zeitfunktion enthalten.
- *suchbedingung* darf kein Spezial-Literal enthalten.
- *suchbedingung* darf keine Transliteration zwischen EBCDIC und Unicode enthalten.
- *suchbedingung* darf keine Umwandlung von Großbuchstaben in Kleinbuchstaben oder von Kleinbuchstaben in Großbuchstaben enthalten, wenn die umzuwandelnde Zeichenkette eine Unicode-Zeichenkette ist.
- *suchbedingung* darf keine multiple Spalte sein.
- *suchbedingung* darf keine User Defined Function (UDF) enthalten.

Besonderheiten für CALL-DML-Tabellen

Bei Spaltenbedingungen müssen für CALL-DML-Tabellen folgende Einschränkungen berücksichtigt werden:

- Eine CALL-DML-Tabelle muss genau eine Primärschlüsselbedingung als Spaltenbedingung oder als Tabellenbedingung enthalten.
- Als Spaltenbedingung ist nur PRIMARY KEY erlaubt.
- Der Datentyp der Spalte mit PRIMARY KEY muss CHAR mit einer Länge von mindestens 4 Zeichen sein.

Spaltenbedingung und Index

Wenn Sie eine Eindeutigkeitsbedingung definieren, wird dafür ein Index mit der bei UNIQUE angegebenen Spalte verwendet:

- Wenn Sie mit CREATE INDEX bereits einen Index definiert haben, der dieselbe Spalte enthält, so wird dieser Index zusätzlich für die Eindeutigkeitsbedingung verwendet.
- Ansonsten wird der benötigte Index implizit erzeugt. Der Name des implizit erzeugten Index beginnt mit UI, gefolgt von einer 16-stelligen Nummer.
  Der Index wird im Space der Basistabelle gespeichert. Bei einer partitionierten Tabelle wird der Index im Space der ersten Partition der Tabelle gespeichert.

Beispiele für Spaltenbedingung

Das Beispiel zeigt einen Ausschnitt aus der CREATE TABLE-Anweisung, die die Tabelle LEISTUNG der Datenbank AUFTRAGKUNDEN erzeugt. Für die Spalte LANZ wird eine Check-Bedingung definiert.

```
CREATE TABLE leistung (...,
lanz INTEGER CONSTRAINT lanz_pos CHECK (lanz > 0))
```

Für die Spalte FIRMA wird eine Nicht-NULL-Bedingung definiert, deren Namen explizit angegeben wird. KNR wird durch die Spaltenbedingung KNR_PRIMARY als Primärschlüssel definiert.

```
CREATE TABLE kunde
(knr INTEGER CONSTRAINT knr_primary PRIMARY KEY,
firma CHAR(40) CONSTRAINT firma_notnull NOT NULL)
```

Für die Tabelle AUFTRAG wird eine Referenzbedingung FOREIGN1 definiert, in der sich der Fremdschlüssel AUFTRAG.KNR auf die Spalte KUNDE.KNR bezieht.

```
ALTER TABLE auftrag
ADD CONSTRAINT foreign1 FOREIGN KEY(knr)
REFERENCES kunde(knr)
```

5.7.2 Tabellenbedingung

Bei der Erzeugung oder Änderung einer Basistabelle (CREATE TABLE, ALTER TABLE) können Tabellenbedingungen angegeben werden. Eine Tabellenbedingung ist eine Integritätsbedingung, die sich auf mehr als eine Spalte der Basistabelle beziehen kann. Keine der Spalten darf eine multiple Spalte sein.

Für CALL-DML-Tabellen darf nur die Integritätsbedingung PRIMARY KEY definiert werden.

tabellenbedingung ::=

```
{
   UNIQUE ( spalte ,...) |
   PRIMARY KEY ( spalte ,...) |
   FOREIGN KEY ( spalte ,...) REFERENCES tabelle [( spalte ,...)] |
   CHECK ( suchbedingung )
}
```

UNIQUE (*spalte*,...)

Eindeutigkeitsbedingung.
Die Kombination der Werte der angegebenen Spalten muss innerhalb der Tabelle eindeutig sein, falls alle Werte ungleich dem NULL-Wert sind.

Die Länge der Spalten muss die Einschränkungen erfüllen, die für einen Index gelten (siehe CREATE INDEX-Anweisung auf "CREATE INDEX - Index erzeugen").

Eine Spalte darf nicht mehrfach in der Spaltenliste genannt werden.

Die Reihenfolge der Spalten in der Spaltenliste muss sich unterscheiden von der Reihenfolge der Spalten in der Spaltenliste, die ggf. für eine andere UNIQUE Bedingung bzw. für eine PRIMARY KEY Bedingung derselben Tabelle angegeben wird.

PRIMARY KEY (*spalte*,...)

Primärschlüsselbedingung.
Die angegebenen Spalten bilden zusammen den Primärschlüssel der Tabelle. Die Kombination der Spaltenwerte muss eindeutig sein. Für jede Tabelle kann nur ein Primärschlüssel definiert werden.

Keine der Spalten darf vom Datentyp VARCHAR oder NVARCHAR sein. Die Summe der Spaltenlängen darf höchstens 256 Zeichen betragen.

Eine Spalte darf nicht mehrfach in der Spaltenliste genannt werden.

Die Reihenfolge der Spalten in der Spaltenliste muss sich unterscheiden von der Reihenfolge der Spalten in der Spaltenliste, die ggf. für eine andere UNIQUE Bedingung derselben Tabelle angegeben wird.

Für Primärschlüsselspalten gilt implizit auch die Nicht-NULL-Bedingung.

FOREIGN KEY ... REFERENCES

Referenzbedingung.Die referenzierenden Spalten dürfen eine Wertekombination, die keinen NULL-Wert enthält, nur dann enthalten, wenn die Wertekombination in den referenzierten Spalten ebenfalls vorkommt.Sie müssen die gleiche Anzahl Spalten der referenzierenden und der referenzierten Tabelle angeben. Die Datentypen der korrespondierenden Spalten müssen genau gleich sein.

Der aktuelle Berechtigungsschlüssel muss das Privileg REFERENCES für die referenzierten Spalten besitzen.

FOREIGN KEY (*spalte*,...)

Spalten der referenzierenden Tabelle, deren Wertekombination in der referenzierten Basistabelle enthalten sein soll.
Eine Spalte darf nicht mehrfach in der Spaltenliste genannt werden.

REFERENCES *tabelle*

Name der referenzierten Basistabelle.
Die referenzierte Basistabelle muss eine SQL-Tabelle sein. Der einfache Name der referenzierten Basistabelle kann durch einen Datenbank- und Schemanamen qualifiziert werden. Der Katalogname muss mit dem Datenbanknamen der referenzierenden Tabelle übereinstimmen.

(*spalte*,...)

Namen der referenzierten Spalten.
Für diese Spalten muss eine Eindeutigkeits- oder Primärschlüsselbedingung definiert sein, die dieselben Spalten und dieselbe Reihenfolge verwendet. Keine der Spalten darf eine multiple Spalte sein.
Eine Spalte darf nicht mehrfach in der Spaltenliste genannt werden.

(*spalte*,...) nicht angegeben:
Der Primärschlüssel der referenzierten Tabelle wird als referenzierte Spalte verwendet.

CHECK (*suchbedingung*)

Check-Bedingung.
Für jeden Datensatz der Tabelle muss die Suchbedingung *suchbedingung* den Wahrheitswert wahr oder unbestimmt annehmen, nicht jedoch den Wahrheitswert falsch.

Für *suchbedingung* gelten folgende Einschränkungen:

- *suchbedingung* darf keine Benutzervariablen enthalten.
- *suchbedingung* darf keine Mengenfunktion enthalten.
- *suchbedingung* darf keine Unterabfrage enthalten, also kann sich *suchbedingung* nur auf Spalten der Tabelle beziehen, zu der die Spaltenbedingung gehört.
- *suchbedingung* darf keine Zeitfunktion enthalten.
- *suchbedingung* darf kein Spezial-Literal enthalten.
- *suchbedingung* darf keine Transliteration zwischen EBCDIC und Unicode enthalten.
- *suchbedingung* darf keine Umwandlung von Großbuchstaben in Kleinbuchstaben oder von Kleinbuchstaben in Großbuchstaben enthalten, wenn die umzuwandelnde Zeichenkette eine Unicode-Zeichenkette ist.
- *suchbedingung* darf keine User Defined Function (UDF) enthalten.

Besonderheiten für CALL-DML-Tabellen

Bei Tabellenbedingungen müssen für CALL-DML-Tabellen folgende Einschränkungen berücksichtigt werden:

- Eine CALL-DML-Tabelle muss genau eine Primärschlüsselbedingung als Spaltenbedingung oder als Tabellenbedingung enthalten.
- Als Tabellenbedingung ist nur PRIMARY KEY erlaubt.
- Der Datentyp einer Spalte des PRIMARY KEY muss CHAR, NUMERIC, INTEGER oder SMALLINT sein. Beim Datentyp NUMERIC sind keine Nachkommastellen erlaubt.
- Die Summe der Spaltenlängen muss zwischen 4 und 256 Zeichen liegen.
- Die Tabellenbedingung definiert einen zusammengesetzten Primärschlüssel. Der Name entspricht dem verbalen Attributnamen des zusammengesetzten Primärschlüssels in SESAM/SQL V1.x.

Tabellenbedingung und Index

Wenn Sie eine Eindeutigkeitsbedingung definieren, wird dafür ein Index mit den bei UNIQUE angegebenen Spalten verwendet:

- Wenn Sie mit CREATE INDEX bereits einen Index definiert haben, der dieselben Spalten enthält, so wird dieser Index zusätzlich für die Eindeutigkeitsbedingung verwendet.
- Sonst wird der benötigte Index implizit erzeugt. Der Name des implizit erzeugten Index beginnt mit UI, gefolgt von einer 16-stelligen Nummer.
  Der Index wird im Space der Basistabelle gespeichert. Bei einer partitionierten Tabelle wird der Index im Space der ersten Partition der Tabelle gespeichert.

Beispiel für Tabellenbedingung

Das Beispiel zeigt einen Ausschnitt aus der CREATE TABLE-Anweisung, die die Tabelle KUNDE der Datenbank AUFTRAGKUNDEN erzeugt.

```
CREATE TABLE kunde
...
CONSTRAINT plausplz
   CHECK ((land = 'D' AND plz >= 00000) OR (land <> 'D'))
...
```

5.8 Spaltendefinition

Die Spaltendefinition legt bei der Erzeugung oder Änderung einer Basistabelle (CREATE TABLE, ALTER TABLE) den Namen und die Eigenschaften einer Spalte fest.

SESAM/SQL unterscheidet zwischen einfachen und multiplen Spalten. Bei einer einfachen Spalte kann pro Satz genau ein Wert gespeichert werden. Bei einer multiplen Spalte können pro Satz mehrere Werte desselben Datentyps gespeichert werden. Eine multiple Spalte besteht aus mehreren Spaltenelementen. In jedem Spaltenelement kann pro Satz genau ein Wert gespeichert werden.

Zur Einbindung von BLOB-Objekten in Basistabellen werden REF-Spalten benötigt. Diese werden mit der FOR REF-Klausel definiert.

Eine Basistabelle kann max. 26134 Spalten eines Datentyps außer VARCHAR und NVAR-CHAR enthalten. Sie kann max. 1000 Spalten mit Datentyp VARCHAR und/oder NVAR-CHAR enthalten. Die für CALL-DML-Tabellen geltenden Einschränkungen sind auf "Spaltendefinition" beschrieben.

---

spaltendefinition ::= *spalte* { *datentyp* [*voreinstellung*] | FOR REF(*tabelle*) }

[[CONSTRAINT *integritätsbedingungsname*] *spaltenbedingung*] ...

[*call_dml_klausel*]

voreinstellung ::= DEFAULT

{

alphanumerisches_literal |
national_literal |
numerisches_literal |
zeit_literal |
CURRENT_TIME(3) |
LOCALTIME(3) |
CURRENT_TIMESTAMP(3) |
LOCALTIMESTAMP(3) |
USER |
CURRENT_USER |
SYSTEM_USER |
NULL |
REF(*tabelle*)

}

call_dml_klausel ::= CALL DML *call_dml_voreinst* [*call_dml_symb_name*]

---

spalte

Name der Spalte. Der Spaltenname muss innerhalb der Basistabelle eindeutig sein.

datentyp

Datentyp für die Spalte.

FOR REF(*tabelle*)

Definiert eine Spalte, die Referenzen auf BLOB-Werte enthält. Mit dieser Klausel lassen sich somit BLOB-Objekte in „normale“ Basistabellen einbinden. BLOB-Werte werden in BLOB-Tabellen gespeichert. Die Definition einer BLOB-Tabelle wird im Abschnitt „CREATE TABLE - Basistabelle erzeugen“ beschrieben. BLOB-Objekte, -Tabellen und REF-Werte werden kurz im Abschnitt „Konzept des SESAM-CLI“ erklärt. Ausführlich wird deren Aufbau im „ Basishandbuch“ beschrieben.

- Die Spalte erhält den Datentyp CHAR(237).
- Die Spalte erhält als Defaultwert den REF-Wert der Klasse. Die Struktur der REF-Werte wird im nächsten Abschnitt beschrieben.
- *tabelle* darf den Datenbanknamen (*catalog*) nicht enthalten.

REF(*tabelle*)

REF-Wert der Klasse, der die gesamte Klasse der BLOB-Werte einer BLOB-Tabelle bezeichnet. Wird eine REF-Spalte erzeugt, so erhält die Spalte den REF-Wert der Klasse als Defaultwert. Dieser ist durch die Angabe des Namens der BLOB-Tabelle bestimmt. Deshalb ist es an dieser Stelle nicht sinnvoll und auch aufgrund der Syntax der Spaltendefinition nicht möglich, eine Voreinstellung für die REF-Spalte anzugeben. Ein REF-Wert hat grundsätzlich folgende Struktur:

ssltt?UID=*uuuu*&OID=*nn*

- *ss* ist der einfache Schemaname der BLOB-Tabelle ohne Datenbankname.
- *tt* ist der einfache Tabellenname der BLOB-Tabelle ohne Schema- und Datenbankname..
- *uuu* ist die eindeutige aus 32 Hexadezimalziffern bestehende Identifikationsnummer des BLOB-Objekts. Beim REF-Wert der Klasse sind alle Ziffern 0.
- *nn* ist die Nummer des BLOB-Objekts in der BLOB-Tabelle. Beim REF-Wert der Klasse ist diese Nummer 0.

voreinstellung

Legt einen SQL-Defaultwert fest, der in die Spalte eingetragen wird, wenn ein Satz eingefügt oder geändert wird und für die Spalte kein Wert oder der Defaultwert angegeben ist.

- *spalte* darf keine multiple Spalte sein.
- *spalte* darf keine CALL-DML-Spalte sein.
- *voreinstellung* muss den Zuweisungsregeln für Defaultwerte genügen (siehe Abschnitt „Defaultwerte für Tabellenspalten“).

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

voreinstellung nicht angegeben:
Es gibt keinen SQL-Defaultwert.
Bei Spalten ohne Nicht-NULL-Bedingung wird der NULL-Wert eingetragen.

[CONSTRAINT *integritätsbedingungsname*] *spaltenbedingung*

Definiert eine Integritätsbedingung für die Spalte. Sie dürfen Integritätsbedingungen nicht für multiple Spalten angeben.

[CONSTRAINT *integritätsbedingungsname*] *spaltenbedingung* nicht angegeben:Keine Spaltenbedingung definiert.

CONSTRAINT *integritätsbedingungsname*

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

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

UN *integritätsbedingungsnummer*
PK *integritätsbedingungsnummer*
FK *integritätsbedingungsnummer*
CH *integritätsbedingungsnummer*
wobei UN für UNIQUE, PK für PRIMARY KEY, FK für FOREIGN KEY und CH für CHECK steht. *integritätsbedingungsnummer* ist eine 16-stellige Nummer. Die NichtNULL-Bedingung wird als Check-Bedingung abgespeichert.

spaltenbedingung

Gibt eine Integritätsbedingung an, die die Spalte erfüllen muss.

call_dml_klausel

Die CALL-DML-Klausel dient der Kompatibilität mit der Version SESAM/SQL V1.x. Die CALL-DML-Klausel darf nur für CALL-DML-Tabellen angegeben werden, aber nicht für Spalten, die den Primärschlüssel bilden. In diesem Fall vergibt SESAM/SQL sowohl die *call_dml_voreinst* als auch den *call_dml_symb_name*.

call_dml_klausel nicht angegeben:
Die Spaltendefinition gilt entweder für eine SQL-Tabelle oder für den Primärschlüssel einer CALL-DML-Tabelle. Bei einer SQL-Tabelle darf die CREATE TABLE- bzw. ALTER TABLE-Anweisung, in der die Spaltendefinition auftritt, ebenfalls keine CALL-DML-Klausel enthalten.

call_dml_voreinst

Gibt den nicht signifikanten Wert einer Spalte als alphanumerisches Literal an.

call_dml_voreinst entspricht dem nicht signifikanten Attributwert der SESAM/SQL-Version 1.x.

call_dml_symb_name

Gibt den symbolischen Namen der Spalte an.

call_dml_symb_name entspricht dem symbolischen Attributnamen der SESAM/SQL-Version 1.x .

call_dml_symb_name nicht angegeben:
Der *call_dml_symb_name* wird vom System vergeben.

Besonderheiten für CALL-DML-Tabellen

Bei Spaltendefinitionen müssen für CALL-DML-Tabellen folgende Einschränkungen berücksichtigt werden:

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

Beispiele für Spaltendefinition

Das Beispiel zeigt einen Ausschnitt aus der CREATE TABLE-Anweisung für die Tabelle AUFTRAG der Datenbank AUFTRAGKUNDEN.

```
CREATE TABLE auftrag
(anr                          INTEGER,
knr                           INTEGER NOT NULL,
konr                          INTEGER,
adatum                        DATE DEFAULT CURRENT_DATE,
atext                         CHARACTER (30),
fertigist                     DATE,
fertigsoll                    DATE,
astnr                         INTEGER DEFAULT 1 NOT NULL,
...)
```

Das Beispiel zeigt die CREATE TABLE-Anweisung für die Tabelle KATART der Datenbank AUFTRAGKUNDEN. Diese Tabelle beinhaltet zwei REF-Spalten.

```
CREATE TABLE katart
```

```
(artnr                  INTEGER NOT NULL,
abb                     FOR REF(zusaetze.bilder),
beschr                  FOR REF(zusaetze.beschreibung))
```

6 Abfrage-Ausdruck

Abfrage-Ausdrücke sind in SESAM/SQL das zentrale Mittel für die Datenabfrage.

Dieses Kapitel beschreibt die Syntax von Abfrage-Ausdrücken und erklärt die unterschiedlichen Join-Möglichkeiten. Im Einzelnen behandelt es folgende Themen:

- Tabellenangabe
- SELECT-Ausdruck
- Tabellenabfrage
- Join
- Unterabfrage
- Verbindung von Abfrage-Ausdrücken mit UNION
- Verbindung von Abfrageausdrücken mit EXCEPT DISTINCT
- Änderbarkeit von Abfrage-Ausdrücken

Überblick

Mit einem Abfrage-Ausdruck können Sätze und Spalten aus Basistabellen und Views ausgewählt werden. Die gefundenen Sätze bilden die Ergebnistabelle.

Ein Abfrage-Ausdruck ist ein Teil einer SQL-Anweisung. Ein Abfrage-Ausdruck kann in Unterabfragen und in einer der folgenden SQL-Anweisungen vorkommen:

CREATE VIEW	View definieren
DECLARE CURSOR	Cursor vereinbaren
INSERT	Sätze in Tabelle einfügen

Die in diesem Kapitel dargestellten Beispiele stellen lediglich den jeweiligen Abfrageausdruck dar. Ohne die umgebende Unterabfrage bzw. SQL-Anweisung sind die Beispiele selbstverständlich nicht ablauffähig.

Um einen Abfrage-Ausdruck in einer SQL-Anweisung zu verwenden, müssen Sie Eigentümer der mit dem Abfrage-Ausdruck angesprochenen Tabellen sein oder das SELECT-Privileg für diese Tabellen besitzen.

---

abfrageausdruck ::= [*abfrageausdruck* { UNION [ALL | DISTINCT] | EXCEPT [DISTINCT] }]

{ *select_ausdruck* | TABLE *tabelle* | *join_ausdruck* | (*abfrageausdruck*) }

---

select_ausdruck

SELECT-Ausdruck, siehe Abschnitt „SELECT-Ausdruck“.

TABLE *tabelle*

Tabellenabfrage, siehe Abschnitt „TABLE - Tabellenabfrage“.

join_ausdruck

Join-Ausdruck, siehe Abschnitt „Join-Ausdruck“.

(*abfrageausdruck*)

Unterabfrage, siehe Abschnitt „Unterabfrage“.

UNION

Verbindung von zwei Abfrage-Ausdrücken mit UNION, siehe Abschnitt „Verbindung von Abfrage-Ausdrücken mit UNION“.

EXCEPT DISTINCT

Verbindung von zwei Abfrage-Ausdrücken mit EXCEPT, siehe Abschnitt „Verbindung von Abfrage-Ausdrücken mit EXCEPT“.

6.1 Tabellenangabe

tabellenangabe ::= { *tabelle* [[AS] *korrelationsname* [(*spalte* , ...)]] |
unterabfrage [AS] *korrelationsname* [(*spalte* , ...)] |
TABLE([*catalog* .] *tabellenfunktion*) [WITH ORDINALITY]
[[AS] *korrelationsname* [(*spalte* , ...)]] |
join_ausdruck }

tabelle

Name einer Basistabelle oder eines View.

Dieselbe Tabelle kann mehrmals in einer Tabellenangabe im Abfrage-Ausdruck vorkommen. Um unterschiedliche Vorkommen derselben Tabelle unterscheiden zu können, werden Korrelationsnamen verwendet.

unterabfrage

Die Tabelle ist die Ergebnistabelle der Unterabfrage *unterabfrage*.

[*catalog*.]*tabellenfunktion*

Die („read-only“) Tabelle (siehe „ Basishandbuch“) ist das Ergebnis der Tabellenfunktion *tabellenfunktion*.

Wenn die Tabellenfunktion DEE() angegeben ist, dann dürfen keine Spaltennamen angegeben werden.

Der Datenbankname *catalog* muss angegeben werden, wenn die enthaltende Anweisung nicht auf der implizit eingestellten Datenbank (siehe "Qualifizierte Namen") ausgeführt werden soll (und daher ggf. auch mit einem anderen SQL-Server).

WITH ORDINALITY

Definition einer Zählspalte in der Ergebnistabelle. Diese Angabe darf nur für die Tabellenfunktion CSV(), nicht aber für DEE() angegeben werden.

Die Ergebnistabelle muss „am Ende“ eine Spalte mehr besitzen als die Spaltenangaben in jeder Zeile der CSV-Datei. Der Datentyp der letzen Spalte der Ergebnistabelle muss DECIMAL(31,0) sein. Diese Spalte dient als Zählspalte. Sie erhält, mit 1 beginnend, aufsteigend die Ordnungszahl der eingelesenen Zeile der CSV-Datei. Mit der WHERE-Klausel können in einem SELECT-Ausdruck auch Ergebnissätze bestimmter Ordnungszahlen ignoriert werden, siehe das Beispiel auf der nächsten Seite.

Die Datentypen jeder Spalte der Ergebnistabelle (mit Ausnahme der letzen Spalte) müssen mit den Datentypen der Spaltenangaben in der CSV-Datei übereinstimmen.

WITH ORDINALITY nicht angegeben:

Die Anzahl der Spalten der Ergebnistabelle muss der Anzahl der Spaltenangaben in der CSV-Datei entsprechen und die Datentypen jeder Spalte müssen übereinstimmen.

Beispiel

Es sei `mit.3.kopfzeilen` eine CSV-Datei mit genau 3 Überschriftszeilen, die nicht ausgewertet bzw. überlesen werden sollen:

```
SELECT c1, c2,...,cn
       FROM TABLE(CSV('mit.3.kopfzeilen' DELIMITER ',' QUOTE '?'
                 ESCAPE '-', CHAR(20), CHAR(20),..., CHAR(20)))
       WITH ORDINALITY
       AS T(c1, c2,....,cn, counter)
WHERE counter > 3
```

korrelationsname

Tabellenname, der innerhalb des Abfrage-Ausdrucks eine Umbenennung für eine Tabelle ist.

Bei jeder Spaltenangabe, die sich auf diese Angabe der Tabelle bezieht, müssen Sie den Spaltennamen mit *korrelationsname* qualifizieren, wenn der Spaltenname nicht eindeutig ist.

Der neue Name muss eindeutig sein, d.h. *korrelationsname* darf nur einmal in einer Tabellenangabe dieses Abfrage-Ausdrucks vorkommen.

Sie müssen eine Tabelle umbenennen, wenn im Abfrage-Ausdruck die Spalten der Tabelle ohne Umbenennung nicht eindeutig angegeben werden können.
Bei *tabellenfunktion* muss *korrelationsname* angegeben werden (Ausnahme: DEE()).

Außerdem können Sie eine Tabelle umbenennen, um durch entsprechende Namen einen Abfrage-Ausdruck verständlicher zu formulieren oder um lange Namen abzukürzen..

Beispiel

Tabelle mit sich selbst verknüpfen:

```
SELECT a.firma, b.firma           -- Kunden abfragen,
   FROM kunde AS a, kunde AS b
   WHERE a.ort = b.ort            -- die in demselben Ort wohnen
   AND a.knr < b.knr              -- aber gleiche Paare vermeiden
```

spalte

Spaltenname, der innerhalb des Abfrage-Ausdrucks eine Umbenennung für die Spalte der dazugehörigen Tabelle ist.

Wenn Sie eine Spalte umbenennen, müssen Sie allen Spalten der Tabelle einen neuen Namen zuweisen.

spalte ist der neue Name der Spalte, der innerhalb der mit *korrelationsname* angegebenen Tabelle eindeutig sein muss. Die Spalte darf in diesem Abfrage-Ausdruck nur mit dem neuen Namen angesprochen werden.

Die Spalten einer Ergebnistabelle müssen umbenannt werden, wenn die Spaltennamen der zugrunde liegenden Tabellen nicht eindeutig sind oder wenn Ergebnisspalten mit intern vergebenen Namen angesprochen werden sollen.

Beispiel

Die Spalten der Tabelle LAGER sollen mit neuen, aussagekräftigeren Namen versehen werden:

```
SELECT * FROM lager l (artikelnummer, aktueller_bestand, lagerort)
WHERE lagerort = 'Teilelager'
```

spalte,... nicht angegeben:

Es gelten die Spaltennamen der zugehörigen Tabelle. Dies können intern vergebene Namen sein, die im Abfrage-Ausdruck nicht angesprochen werden können.

join_ausdruck

Join-Ausdruck, der die Tabellen bestimmt, aus denen Daten ausgewählt werden. Join-Ausdrücke sind im Abschnitt „Join-Ausdruck“ beschrieben.

Zugrunde liegende Basistabellen

Abhängig von der Angabe in der Tabellenangabe ist sind die zugrunde liegenden Basistabellen wie folgt definiert:

Angabe in Tabellenangabe	**zugrunde liegende Basistabellen**
Basistabelle	die Basistabelle
View	alle Basistabellen, auf die sich der View direkt oder indirekt bezieht
Unterabfrage	die der Unterabfrage zugrunde liegende Basistabelle
TABLE([*catalog*.]*tabellenfunktion*)	keine Basistabelle

Tabelle 24: Zugrunde liegende Basistabellen

6.2 SELECT-Ausdruck

select_ausdruck ::=

```
SELECT [ALL | DISTINCT] select_liste
FROM tabellenangabe,...
[WHERE suchbedingung]
[GROUP BY spalte,...]
[HAVING suchbedingung]
```

select_liste ::= { * | { *tabelle*.* | *ausdruck* [[AS] *spalte*] } }

Für alle Klauseln gilt:

- Die angegebene Reihenfolge der Klauseln muss eingehalten werden.
- Spaltennamen müssen eindeutig sein. Kommt ein Spaltenname in mehreren Tabellen vor, so müssen Sie den Spaltennamen mit dem Tabellennamen qualifizieren. Wenn Sie eine Tabelle mit einem Korrelationsnamen für die Dauer der SELECT-Anweisung umbenennen (siehe Abschnitt „Tabellenangabe“), dürfen Sie nur noch den Korrelationsnamen verwenden.

  Beispiel

  ```
  SELECT a.knr, l.lsatz
  FROM auftrag a, leistung l WHERE a.anr=l.anr
  ```

Auswertung von SELECT-Ausdrücken

SELECT-Ausdrücke werden in folgender Reihenfolge ausgewertet:

1. Aus allen Tabellenangaben in der FROM-Klausel wird das Kartesische Produkt gebildet..
2. Ist eine WHERE-Klausel angegeben, wird die WHERE-Suchbedingung auf alle Sätze des Kartesischen Produkts angewendet. Es werden die Sätze ausgewählt, für die die Suchbedingung wahr ergibt.
3. Ist eine GROUP BY-Klausel angegeben, werden die in Punkt 2 bestimmten Sätze zu Gruppen zusammengefasst.
4. Ist eine HAVING-Klausel angegeben, wird die HAVING-Suchbedingung auf alle Gruppen angewendet. Es werden die Gruppen ausgewählt, die die Suchbedingung erfüllen.
5. Enthält die SELECT-Liste eine Mengenfunktion und ist die Ergebnistabelle noch nicht gruppiert, werden alle Sätze der Ergebnistabelle zu einer Gruppe zusammengefasst.
6. Ist die Ergebnistabelle gruppiert (eine oder mehrere Gruppen), wird die SELECT-Liste für jede Gruppe ausgewertet.

   Ist die Ergebnistabelle nicht gruppiert, wird die SELECT-Liste für jeden Ergebnissatz ausgewertet.

   Die resultierenden Sätze bilden die Ergebnistabelle des SELECT-Ausdrucks.

6.2.1 SELECT-Liste - Ergebnisspalten auswählen

Mit der SELECT-Liste legen Sie die Spalten der Ergebnistabelle fest.

```
SELECT [ALL | DISTINCT] select_liste ...

select_liste ::= { * | { tabelle .* | ausdruck [[AS] spalte ] } } ...
```

ALL

Doppelte Sätze in der Ergebnistabelle bleiben erhalten.

DISTINCT

Doppelte Sätze werden entfernt.

*

Alle Spalten auswählen. Die Reihenfolge und die Namen der Spalten der in der FROM-Klausel angegebenen Tabellen werden übernommen. Bei mehreren Tabellen gilt die Reihenfolge der Tabellen in der FROM-Klausel. Es muss mindestens eine Spalte geben.

tabelle.*

Alle Spalten der Tabelle *tabelle* auswählen. Die Tabelle *tabelle* muss in der FROM-Klausel enthalten sein. Die Reihenfolge und die Namen der Spalten von *tabelle* werden übernommen. *tabelle* darf nicht der Korrelationsname für eine Tabellenfunktion DEE() sein.

ausdruck

Ausdruck, der eine Ergebnisspalte bezeichnet. Enthält *ausdruck* eine Spaltenangabe, muss die Tabelle, zu der die Spalte gehört, in der FROM-Klausel dieses SELECT-Ausdrucks enthalten sein.

Die Namen der Spalten in der SELECT-Liste müssen eindeutig sein. Wenn Sie Tabellen verbinden und diese Basistabellen Spalten mit identischen Namen haben, müssen Sie zur eindeutigen Identifikation die jeweiligen Namen der Tabellen bzw. deren Korrelationsnamen voranstellen.

Ist SELECT DISTINCT angegeben, darf *ausdruck* nicht aus einer multiplen Spaltenangabe bestehen.

Wenn in einer Spaltenauswahl eine Mengenfunktion (AVG, COUNT, MAX, MIN, SUM) vorkommt, gelten folgende Einschränkungen:

- In der SELECT-Liste dürfen nur Spaltennamen vorkommen, die in der GROUP BY-Klausel aufgeführt sind oder Argument einer Mengenfunktion sind.
- In **derselben** Ebene einer SELECT-Abfrage darf nur eine Mengenfunktion mit DISTINCT verwendet werden. Sie dürfen zum Beispiel nicht angeben:

SELECT COUNT(DISTINCT ...) ...SUM(DISTINCT ...) ...

[AS] *spalte*

Name für die Ergebnisspalte, die mit *ausdruck* angegeben ist.

Beispiel

```
SELECT anr AS auftrags_nr, COUNT(*) AS anzahl FROM auftrag GROUP BY anr
```

```
auftrags_nr         anzahl
...                 ...
```

spalte nicht angegeben:

Wenn *ausdruck* ein Spaltenname ist, erhält die Ergebnisspalte diesen Namen, andernfalls ist der Spaltenname nicht definiert.

Beispiel

```
SELECT anr, COUNT(*) FROM auftrag GROUP BY anr
```

```
anr         ...
...         ...
```

Spalten der Ergebnistabelle

Die Reihenfolge der Spalten in der Ergebnistabelle entspricht der Reihenfolge der Spalten in der SELECT-Liste.

Die Attribute einer Ergebnisspalte (Datentyp, Länge, Genauigkeit, Nachkommastellen) werden entweder von der zugrunde liegenden Spalte übernommen oder ergeben sich aus dem angegebenen Ausdruck.

Eine Ergebnisspalte kann den NULL-Wert liefern, wenn eine der folgenden Bedingungen gilt:

- eine der verwendeten Spalten kann den NULL-Wert liefern.
  Für Spalten von Tabellenfunktionen ist das immer der Fall. Für Spalten von Basistabellen ist das nur dann nicht der Fall, wenn für die Spalte eine NOT NULL-Bedingung gilt.
- Der Ausdruck, der die Ergebnisspalte bezeichnet, enthält mindestens einen der folgenden Operanden bzw. Elemente:
  - eine Indikatorvariable
  - eine Unterabfrage
  - die Mengenfunktion AVG, MAX, MIN oder SUM
  - einen CAST-Ausdruck der Form CAST (NULL AS *datentyp*)
  - einen CASE-Ausdruck, der in mindestens einer THEN- bzw. ELSE-Klausel den NULL-Wert enthält
  - einen CASE-Ausdruck mit NULLIF
  - einen CASE-Ausdruck mit COALESCE, wobei mindestens ein Operand des COA-LESCE (*ausdruck1 ... ausdruckn*) einen der oben aufgelisteten Operanden bzw. eines der oben aufgelisteten Elemente enthält

Beispiele

„*“ wählt alle Spalten der in der FROM-Klausel angegebenen Tabellen aus. Die Reihenfolge der Spalten in der Ergebnistabelle wird durch die Reihenfolge der Tabellen in der FROM-Klausel und innerhalb einer Tabelle durch die definierte Reihenfolge bestimmt.

```
SELECT * FROM auftrag, kunde
```

KUNDE.* wählt alle Spalten der Tabelle KUNDE aus. Durch DISTINCT werden doppelte Sätze nicht in die Ergebnistabelle übernommen.

```
SELECT DISTINCT auftrag.anr, kunde.* FROM auftrag, kunde
```

Folgender SELECT-Ausdruck wählt die Auftragsnummern aus den Tabellen LEISTUNG und AUFTRAG aus. Die Spaltennamen müssen eindeutig sein. Werden Tabellen mit identischen Spaltennamen verbunden, dann müssen die Spaltennamen mit den Tabellen- bzw. Korrelationsnamen qualifiziert werden. Durch ALL (Voreinstellung) werden doppelte Sätze in die Ergebnistabelle übernommen.

```
SELECT ALL L.anr, A.anr FROM leistung L, auftrag A
```

Folgender SELECT-Ausdruck wählt die Bezeichnung der Leistung und den Preis pro Leistungseinheit einschließlich Mehrwertsteuer aus. Ist *ausdruck* ohne die Angabe [AS] *spalte* ein Spaltenname, dann erhält die Spalte der Ergebnistabelle diesen Spaltennamen (im Beispiel LTEXT).
Mit [AS] *spalte* kann ein Name für die Ergebnisspalte vergeben werden, auf die sich *ausdruck* bezieht (im Beispiel BRUTTOPREIS). Die Eigenschaften einer Spalte der Ergebnistabelle (Datentyp, Länge, Stellen, Nachkommastellen) werden entweder von der zu Grunde liegenden Spalte übernommen (LTEXT) oder ergeben sich aus dem angegebenen *ausdruck* `(lsatz*(1.0+mwsatz)).`

```
SELECT ltext, lsatz*(1.0+mwsatz) AS bruttopreis
```

Die Ergebnistabelle enthält einen einzigen Satz. Dessen einzige Spalte die Summe der Werte ungleich NULL von LEISTUNG.LSATZ ist, bzw. NULL, wenn es keinen solchen Satz gibt. Kommt in der SELECT-Liste eine Mengenfunktion vor, dann dürfen in der SELECT-Liste nur Spaltennamen vorkommen, die innerhalb des Arguments einer Mengenfunktion aufgeführt sind.

```
SELECT SUM(lsatz) FROM leistung
```

Die Ergebnistabelle enthält einen Satz, dessen einzige Spalte die Anzahl der Sätze von KONTAKT ist. Bezeichnet *ausdruck* ohne AS-Klausel keine Spalte, dann ist der Spaltenname nicht definiert.

```
SELECT COUNT(*) FROM kontakt
```

6.2.2 SELECT ... FROM - Tabellen angeben

In der FROM-Klausel geben Sie die Tabellen an, aus denen Daten ausgewählt werden sollen.

Um Sätze in den angegebenen Tabellen lesen zu können, müssen Sie entweder Eigentümer dieser Tabellen sein oder das SELECT-Zugriffsrecht besitzen.

`SELECT ...`

`FROM` *tabellenangabe*`,...`

---

tabellenangabe

Angabe einer Tabelle, aus der Daten gelesen werden. Sie dürfen nur Tabellen derselben Datenbank angeben.

Beispiele

Aus dem Kartesischen Produkt der Tabellen KUNDE und AUFTRAG werden die Spalten KNR der Tabelle KUNDE und ANR der Tabelle AUFTRAG ausgewählt. Dabei werden die Tabellen KUNDE und AUFTRAG innerhalb des SELECT-Ausdrucks durch Vergabe von Korrelationsnamen umbenannt. Jede Spaltenangabe, die sich innerhalb des SELECT-Ausdrucks auf KUNDE oder AUFTRAG bezieht, muss dann mit den Korrelationsnamen qualifiziert werden. Korrelationsnamen können verwendet werden, um Spalten eindeutig qualifizieren zu können, um lange Tabellennamen abzukürzen oder in SELECT-Ausdrücken jeweils zum Zusammenhang passende Tabellennamen anzugeben. Aus dem Kartesischen Produkt der Tabellen KUNDE und AUFTRAG werden die Spalten A.KNR und B. ANR ausgewählt:

```
SELECT A.knr, B.anr FROM kunde A, auftrag B
```

Ergebnistabelle

knr	anr
100	200
100	210
100	211
usw.	usw.
107	300
107	305

Die Tabelle AUFSTAT wird in AUFTRAGSSTATUS umbenannt und die Spalten ASTNR und ASTXT werden unter den neuen Namen AUFTRAGSSTATUSNUMMER und AUFTRAGSSTATUSTEXT ausgewählt. Werden in der SELECT-Liste mit „*“ alle Spalten ausgewählt, dann können mit „(*spalte*, ...)“ in *tabellenangabe* neue Spaltennamen vergeben werden. Im Gegensatz zur AS-Klausel in der SELECT-Liste können jedoch nicht einzelne Spaltennamen umbenannt werden. Es können nur alle Spaltennamen umbenannt werden. Die neuen Namen müssen statt der alten in der SELECT-Liste und in der WHERE-, GROUP BY- und HAVING-Klausel verwendet werden.

```
SELECT * FROM aufstat

AS auftragsstatus (auftragsstatusnummer, auftragsstatustext)
```

Wird eine Tabelle in der FROM-Klausel mehrfach angegeben, wie beim Join einer Tabelle mit sich selbst, dann müssen Korrelationsnamen definiert werden, um Spalten eindeutig angeben zu können. Referenzen in der SELECT-Liste und den WHERE-, GROUP BY- und HAVING-Klauseln müssen diese Korrelationsnamen statt der ursprünglichen Tabellennamen verwenden.

```
SELECT A.knr, B.knr FROM kunde A, kunde B
```

6.2.3 SELECT ... WHERE - Ergebnissätze auswählen

In der WHERE-Klausel geben Sie eine Suchbedingung an, um Sätze für die Ergebnistabelle auszuwählen. Die Ergebnistabelle enthält nur die Sätze, die die angegebene Bedingung erfüllen (d.h. die Suchbedingung ist wahr). Sätze, für die die Suchbedingung falsch oder unbestimmt ergibt, werden nicht in die Ergebnistabelle aufgenommen.

```
SELECT ...
WHERE suchbedingung
```

suchbedingung

Bedingung, die die auszuwählenden Sätze erfüllen müssen.

Beispiele

Die Prädikate sind ausführlich im Kapitel „Zusammengesetzte Sprachelemente“ beschrieben. Hier sind wesentliche Arten von Suchbedingungen an Hand von einfachen Beispielen zusammengestellt.

Vergleich mit Konstante: =, <, <=, >, >=, <>

```
SELECT knr, firma FROM kunde WHERE plz = 81739
```

Vergleich mit Zeichenketten-Muster: [NOT] LIKE

```
SELECT * FROM kunde WHERE firma LIKE 'Sie%'
```

Bereichsabfrage: [NOT] BETWEEN

```
SELECT knr, firma FROM kunde WHERE plz BETWEEN 80000 AND 89999
```

Vergleich auf NULL-Wert: IS [NOT] NULL

```
SELECT lnr, anr, ltext FROM leistung WHERE rnr IS NULL
```

Vergleich auf mehrere Werte: [NOT] IN

```
SELECT knr, firma FROM kunde WHERE plz IN (81739, 80469)
```

Innere SELECT-Anweisung: [NOT] EXISTS

```
SELECT firma FROM kunde
WHERE NOT EXISTS (SELECT * FROM auftrag WHERE kunde.knr = auftrag.knr)
```

Unterabfrage (siehe Abschnitt „Unterabfrage“):

Unterabfrage, die eine Ergebnisspalte liefert: ALL, ANY, SOME, [NOT] IN

```
SELECT firma FROM kunde WHERE kunde.knr =
SOME (SELECT knr FROM auftrag WHERE adatum = DATE '<date>')
```

Korrelierte Unterabfrage:

Für jeden Auftrag die Leistung heraussuchen, die mindestens doppelt so groß ist wie die Durchschnittsleistung für diesen Auftrag:

```
SELECT l1.lnr, l1.anr, l1.ltext FROM leistung l1
   WHERE l1.lanz * l1.lsatz > 2 *
   (SELECT AVG(l2.lanz * l2.lsatz)
    FROM leistung l2 WHERE l2.anr = l1.anr)
```

Bedingung: AND, OR, NOT

```
SELECT lnr, anr, ldatum, ltext FROM leistung
WHERE ltext = 'Schulung' AND ldatum >= DATE '<date>'
```

6.2.4 SELECT ... GROUP BY - Ergebnissätze gruppieren

Mit Hilfe der GROUP BY-Klausel werden Tabellensätze zu Gruppen zusammengefasst. Zwei Sätze gehören zu derselben Gruppe, wenn für jede Gruppierungsspalte die Werte in beiden Sätzen nach den Vergleichsregeln (siehe Abschnitt „Vergleich von zwei Zeilen“) gleich sind oder beide der NULL-Wert sind.

Die Ergebnistabelle enthält für jede Gruppe einen Satz.

```
SELECT ...
GROUP BY spalte,...
```

spalte

Gruppierungsspalte. *spalte* muss Teil einer Tabelle sein, die in der FROM-Klausel angegeben wurde. Nicht eindeutige Spaltennamen müssen mit dem Tabellennamen qualifiziert werden. Haben Sie in der FROM-Klausel einen Korrelationsnamen für die betroffene Tabelle vereinbart, muss dieser Name zur Qualifizierung verwendet werden.

Multiple Spalten dürfen nicht als Gruppierungsspalte verwendet werden.

Auswirkung der GROUP BY-Klausel

Wenn Sie die GROUP BY-Klausel angeben, dürfen in der SELECT-Liste nur noch Spaltennamen vorkommen, die bei GROUP BY aufgeführt oder Argument einer Mengenfunktion sind.

Mengenfunktionen für Spalten einer gruppierten Tabelle werden für jede Gruppe ausgewertet..

Wie werden die Gruppen gebildet?

- Die Sätze, die in allen angegebenen Gruppierungsspalten einen nach den Vergleichsregeln gleichen Wert enthalten, bilden eine Gruppe.
- Sätze, die in den gleichen Gruppierungsspalten den NULL-Wert und in den restlichen Gruppierungsspalten gleiche Werte enthalten, werden zu einer Gruppe zusammengefasst..

Beispiele

Für jede Auftragsnummer den durchschnittlichen Mehrwertsteuersatz bilden:

```
SELECT anr, AVG(mwsatz) FROM leistung GROUP BY anr
   anr
   200         0.14
   211         0.06
   250         0.07
```

Für alle Kunden außerhalb der USA wird die Anzahl der Kontaktpersonen, gruppiert nach der Kundennummer, ermittelt. Bei Angabe einer GROUP BY-Klausel dürfen in der SELECT-Liste nur noch Spaltennamen vorkommen, die in der GROUP BY-Klausel angegeben sind oder Argument einer Mengenfunktion sind. Die Ergebnistabelle des SELECT-Ausdrucks erhält für jede Gruppe einen Satz.

```
SELECT kontakt.knr, COUNT(*) AS anzahl FROM kontakt, kunde
   WHERE kontakt.knr = kunde.knr AND kunde.land <>'USA'
   GROUP BY kontakt.knr
```

Ergebnistabelle

knr	anzahl
100	2
101	1
102	1
103	1
104	1
105	1

Wenn der SELECT-Ausdruck um die folgende HAVING-Klausel (siehe folgenden Abschnitt) ergänzt wird, dann enthält die Ergebnistabelle nur noch den ersten Satz.

```
HAVING COUNT(*) > 1
```

6.2.5 SELECT ... HAVING - Gruppen auswählen

In der HAVING-Klausel geben Sie Suchbedingungen an, um Gruppen auszuwählen. Die Ergebnistabelle enthält den Satz für eine Gruppe, wenn die Gruppe die angegebene Suchbedingung erfüllt. Ist keine GROUP BY-Klausel angegeben, gelten alle Sätze als eine Gruppe.

```
SELECT ...
HAVING suchbedingung
```

suchbedingung

Suchbedingung, die eine Gruppe erfüllen muss.

Im Unterschied zur WHERE-Suchbedingung, die für jeden Satz einer Tabelle ausgewertet wird, wird die HAVING-Suchbedingung einmal pro Gruppe ausgewertet.

Für einen Spaltennamen in *suchbedingung* muss eine der folgenden Bedingungen gelten:

- Der Spaltenname ist in der GROUP BY-Klausel aufgeführt.
- Der Spaltenname ist Argument einer Mengenfunktion (AVG(), SUM(), ...). Wenn der Spaltenname auch in der SELECT-Liste erscheint, darf er dort auch nur als Argument einer Mengenfunktion vorkommen.
- Der Spaltenname kommt in einer Unterabfrage vor. Wenn der Spaltenname sich auf die Tabellen in der FROM-Klausel bezieht, muss er in der GROUP BY-Klausel aufgeführt oder Argument einer Mengenfunktion sein.
- Der Spaltenname ist Teil einer Tabelle aus einem übergeordneten SELECT-Ausdruck..

Beispiel

Für jeden Auftrag soll die zuletzt erbrachte Leistung angezeigt werden, sofern sie nach dem angegebenen Datum erfolgt ist:

```
SELECT anr, MAX(ldatum) FROM leistung GROUP BY anr
HAVING MAX(ldatum) > DATE'<date>'
```

6.3 TABLE - Tabellenabfrage

Mit der Tabellenabfrage wählen Sie alle Spalten einer Tabelle aus.

Um die Sätze der angegebenen Tabelle lesen zu können, müssen Sie entweder Eigentümer dieser Tabelle sein oder das SELECT-Zugriffsrecht besitzen.

```
TABLE tabelle
```

tabelle

Name der Tabelle (Basistabelle oder View), aus der alle Spalten ausgewählt werden. Reihenfolge, Namen und Attribute (Datentyp, Länge, Genauigkeit, Nachkommastellen) der Spalten von *tabelle* werden übernommen.

Der Abfrage-Ausdruck TABLE *tabelle* entspricht dem SELECT-Ausdruck (SELECT * FROM *tabelle*) (siehe Abschnitt „SELECT-Ausdruck“).

Beispiel

Alle Spalten der Tabelle LEISTUNG anzeigen:

```
TABLE leistung
```

6.4 Join

Ein Join ist die Verknüpfung von Daten aus zwei oder mehreren Tabellen. Eine Tabelle kann auch mit sich selbst verknüpft werden.

Welche Sätze der beteiligten Tabellen in die Ergebnistabelle aufgenommen werden, ist abhängig vom Join-Typ und von gegebenenfalls vorhandenen Join-Bedingungen.

Es gibt zwei Möglichkeiten, einen Join zu formulieren:

- mit einem Join-Ausdruck
- ohne Join-Ausdruck: in einem SELECT-Ausdruck bzw. in einer SELECT-Anweisung über die FROM-Klausel und gegebenenfalls WHERE-Klausel

6.4.1 Join-Ausdruck

Ein Join-Ausdruck enthält die zu verknüpfenden Tabellen, die gewünschte Join-Operation und gegebenenfalls eine Join-Bedingung.

Ein Join-Ausdruck kann angegeben werden:

- als Abfrage-Ausdruck in einer SQL-Anweisung
- in der FROM-Klausel eines SELECT-Ausdrucks oder einer SELECT-Anweisung
- in einer Unterabfrage in Select-Liste und HAVING-Klausel

Die Ergebnistabelle eines Join-Ausdrucks ist nicht änderbar.

---

join_ausdruck `::={` *tabellenangabe* `CROSS JOIN` *tabellenangabe* |

tabellenangabe `[` INNER | `{ LEFT | RIGHT | FULL } [OUTER] ] JOIN`

tabellenangabe `ON` *suchbedingung* |

tabellenangabe `UNION JOIN` *tabellenangabe* |

(*join_ausdruck*) }

---

tabellenangabe

Angabe einer Tabelle, aus der Daten gelesen werden (siehe Abschnitt „Tabellenangabe“).

CROSS

CROSS-Operator zum Bilden eines Cross Join. Ein Cross Join entspricht dem Kartesischen Produkt aus den beteiligten Tabellen (siehe Abschnitt „Cross Join“).

INNER

INNER-Operator zum Bilden eines Inner Join. Bei einem Inner Join enthält die Ergebnistabelle nur die Sätze, die die Join-Bedingung erfüllen (siehe Abschnitt „Inner Join“).

LEFT, RIGHT, FULL

Operatoren zum Bilden eines Outer Join. Eine Tabelle, die Teil eines Outer Join ist, darf keine multiplen Spalten enthalten.

Bei einem Outer Join legen Sie mit der Art des Outer Join die dominante(n) Tabelle(n) fest (siehe Abschnitt „Outer Join“).

Erfüllt ein Satz der dominanten Tabelle nicht die Join-Bedingung, wird der Satz trotzdem in die Ergebnistabelle übernommen. Die Ergebnisspalten, die sich auf die andere Tabelle beziehen, werden auf NULL-Werte gesetzt.

LEFT Die Tabelle links vom LEFT-Operator ist die dominante Tabelle.

RIGHT	Die Tabelle rechts vom RIGHT-Operator ist die dominante Tabelle.
FULL	Die Tabellen links und rechts vom FULL-Operator sind beide dominante Tabellen. FULL erzeugt die Vereinigung der mit LEFT und RIGHT erzeugten Tabellen.

suchbedingung

Suchbedingung, die als Join-Bedingung zum Verknüpfen der angegebenen Tabellen verwendet wird.

Für eine in *suchbedingung* angegebene Spalte gilt:

Die Spalte muss entweder Teil einer der zu verknüpfenden Tabellen sein oder, im Fall von Unterabfragen, Teil einer Tabelle aus einem übergeordneten SELECT-Ausdruck.

Wenn in *suchbedingung* eine Mengenfunktion vorkommt, muss eine der beiden Bedingungen gelten:

- Die Mengenfunktion ist in einer Unterabfrage enthalten.
- Der Join-Ausdruck ist in einer Select-Liste oder HAVING-Klausel enthalten und die Spaltenangabe im Argument der Mengenfunktion ist eine Außenreferenz.

UNION

UNION-Operator zum Bilden eines Union Join. Eine Tabelle, die Teil eines Union Join ist, darf keine multiplen Spalten enthalten.

Die Ergebnistabelle eines Union Join enthält sowohl die Sätze der Tabelle links vom UNION-Operator als auch die Sätze der Tabelle rechts vom UNION-Operator, jeweils ergänzt um die auf NULL-Werte gesetzten Spalten der anderen Tabelle (siehe Abschnitt „Union Join“).

join_ausdruck

Geschachtelter Join-Ausdruck, um einen Join aus mehr als zwei Tabellen zu bilden.

6.4.2 Join ohne Join-Ausdruck

Ein Inner- bzw. ein Cross Join kann in SESAM/SQL auch ohne Join-Ausdruck gebildet werden. Die zu verknüpfenden Tabellen werden in der FROM-Klausel eines SELECT-Ausdrucks aufgelistet, und die Join-Suchbedingung wird in der zugehörigen WHERE-Klausel formuliert.

```
SELECT ... FROM tabellenangabe , tabellenangabe [,...] WHERE suchbedingung_mit_joinspalten
```

Beispiel

Kundenname und zugehörige Auftragsnummer aus den Tabellen KUNDE und AUFTRAG heraussuchen:

```
SELECT firma, anr FROM kunde,
auftrag
WHERE kunde.knr=auftrag.knr
```

6.4.3 Join-Typen

SESAM/SQL unterstützt den Cross Join, den Inner Join, den Outer Join und den Union Join. Die Join-Typen sind im Folgenden erklärt und anhand von Beispielen erläutert.

6.4.3.1 Cross Join

Die Ergebnistabelle eines Cross Join ist das Kartesische Produkt aus den beteiligten Tabellen. Jeder Satz aus der Tabelle links vom CROSS-Operator wird mit jedem Satz der Tabelle rechts vom CROSS-Operator verkettet.

Beispiel

Das Kartesische Produkt der Tabellen KUNDE und AUFTRAG bilden:

`SELECT *`	oder	`SELECT *`
`FROM kunde, auftrag`		`FROM kunde CROSS JOIN auftrag`

Tabelle KUNDE

knr	**firma**	**...**
100	Siemens AG	
101	Login GmbH	
102	JIKO GmbH	
...	...	
106	Foreign Ltd.	
107	Externa & Co KG	

Tabelle AUFTRAG

anr	**knr**	**...**
200	102	
210	106	
211	106	
...	...	
300	101	
305	105	

Ergebnistabelle

knr	**firma**	**...**	**anr**	**knr**	**...**
100	Siemens AG		200	102	

101	Login GmbH		200	102	
102	JIKO GmbH		200	102	
...	...		...	...	
106	Foreign Ltd.		200	102	
107	Externa & Co KG		200	102	
100	Siemens AG		210	106	
101	Login GmbH		210	106	
102	JIKO GmbH		210	106	
...	...		...	...	
106	Foreign Ltd.		210	106	
107	Externa & Co KG		210	106	
...	...		...	...	
100	Siemens AG		305	105	
101	Login GmbH		305	105	
102	JIKO GmbH		305	105	
...	...		...	...	
106	Foreign Ltd.		305	105	
107	Externa & Co KG		305	105	

6.4.3.2 Inner Join

Bei einem Inner Join enthält die Ergebnistabelle nur die Sätze, die die Join-Bedingung erfüllen.

Einfacher Inner Join

Ein einfacher Inner Join wählt Sätze aus dem Kartesischen Produkt von zwei Tabellen aus.

Beispiel

Kundenname und zugehörige Auftragsnummer aus den Tabellen KUNDE und AUFTRAG heraussuchen:

`SELECT firma, anr`	oder	`SELECT firma, anr`
`FROM kunde, auftrag`		`FROM kunde JOIN auftrag`
`WHERE kunde.knr=auftrag.knr`		`ON kunde.knr=auftrag.knr`

Kunden, die keinen Auftrag erteilt haben, zum Beispiel die Firma Jonas Fischladen mit der Kundennummer 104, sind in der Ergebnistabelle nicht enthalten.

```
firma           anr
Login GmbH      300
JIKO GmbH       200
Pudelshop Anke  250
Pudelshop Anke  251
Pudelshop Anke  305
Foreign Ltd.    210
Foreign Ltd.    211
```

Beispiel

Für jeden Auftrag wird die zugehörige Leistung ausgewählt.

```
SELECT a.anr, a.atext, a.astnr, l.lnr, l.ltext
FROM auftrag a INNER JOIN leistung l ON a.anr = l.anr

anr atext                   astnr lnr ltext

200 Mitarbeiterschulung         5   1 Schulungsmaterial
200 Mitarbeiterschulung         5   2 Schulung
200 Mitarbeiterschulung         5   3 Schulung
211 Datenbank-Entwurf Kunden    4   4 Systemanalyse
211 Datenbank-Entwurf Kunden    4   5 Datenbankentwurf
211 Datenbank-Entwurf Kunden    4   6 Kopien/Folien
211 Datenbank-Entwurf Kunden    4   7 Handbuch
250 Serienbrief-Einweisung      2  10 Reisekosten
250 Serienbrief-Einweisung      2  11 Schulung
```

Mehrfacher Inner Join

Ein mehrfacher Inner Join wählt Spalten aus dem Kartesischen Produkt von mehr als zwei Tabellen aus.

Beispiel

Aus den Tabellen KUNDE, AUFTRAG und LEISTUNG wird für jeden Kunden, der einen Auftrag gestellt hat, die zugehörige Leistung herausgesucht.

`SELECT k.firma, a.anr, l.lnr`	oder	`SELECT k.firma, a.anr, l.lnr`
`FROM kunde k,auftrag a,leistung l`		`FROM kunde k JOIN auftrag a`
`WHERE k.knr=a.knr`		`ON k.knr=a.knr`
`AND a.anr=l.anr`		`JOIN leistung l ON a.anr=l.anr`

```
firma            anr    lnr
JIKO GmbH        200      1
JIKO GmbH        200      2
JIKO GmbH        200      3
Foreign Ltd.     211      4
Foreign Ltd.     211      5
Foreign Ltd.     211      6
Foreign Ltd.     211      7
Pudelshop Anke   250     10
Pudelshop Anke   250     11
```

6.4.3.3 Outer Join

Eine weitere Form des Join ist der Outer Join. Er wird gebildet, indem Sie im Join-Ausdruck das Schlüsselwort LEFT, RIGHT oder FULL verwenden. Im Gegensatz zum Inner Join gilt bei einem Outer Join:

Es gibt eine (LEFT, RIGHT) oder zwei (FULL) **dominante** Tabellen. Erfüllt ein Satz einer dominanten Tabelle nicht die Join-Bedingung, wird der Satz trotzdem in die Ergebnistabelle übernommen. Die Ergebnisspalten, die sich auf die andere Tabelle beziehen, werden auf NULL-Werte gesetzt.

Beispiel

Wie im Beispiel für den Inner Join möchten Sie aus den Tabellen KUNDE und AUFTRAG Kundenname und zugehörige Auftragsnummer heraussuchen. Es sollen aber alle Kunden aufgelistet werden, auch solche, die noch keinen Auftrag gestellt haben. Dazu formulieren Sie folgenden Outer Join:

```
SELECT firma, anr FROM kunde
LEFT OUTER JOIN auftrag ON kunde.knr=auftrag.knr
```

Kunden, die keinen Auftrag erteilt haben, zum Beispiel die Firma Jonas Fischladen mit der Kundennummer 104, sind jetzt in der Ergebnistabelle enthalten. Für die fehlende Auftragsnummer ist der NULL-Wert eingetragen.

```
firma               anr
Siemens AG
Login GmbH          300
JIKO GmbH           200
Plenzer Trading
Jonas Fischladen
Pudelshop Anke      250
Pudelshop Anke      251
Pudelshop Anke      305
Foreign Ltd.        210
Foreign Ltd.        211
Externa & Co KG
```

6.4.3.4 Union Join

Ein weiterer Join-Typ ist der Union Join. Die Ergebnistabelle eines Union Join wird folgendermaßen gebildet:

- Die Tabelle links vom UNION-Operator wird rechts ergänzt um die Spalten der anderen Tabelle. Die ergänzten Spalten werden auf den NULL-Wert gesetzt.
- Die Tabelle rechts vom UNION-Operator wird links ergänzt um die Spalten der anderen Tabelle. Die ergänzten Spalten werden auf den NULL-Wert gesetzt.
- Die Vereinigungsmenge aus beiden ergänzten Tabellen bildet schließlich die Ergebnistabelle.

Beispiel

Die Tabellen ARTIKEL und VERWENDUNG sollen über Union Join verknüpft werden.

```
SELECT artikel.artnr, artikel.artbez, verwendung. *
   FROM artikel UNION JOIN verwendung

   artnr        artbez      artnr   bestandteil   anzahl

   1            Fahrrad
   2            Fahrrad
   10           Rahmen
   11           Rahmen
   120          Vorderrad
   130          Hinterrad
   200          Lenkstange
   ...
   501          Mutter M5
                               1     10              1
                               1    120              1
                               1    130              1
                               1    200              1
                             120    210              1
                             ...
                             200    501             10
```

6.4.3.5 Zusammengesetzter Join

Wenn Sie mehr als zwei Tabellen über Joins verbinden, können Sie mehrere Join-Ausdrücke schachteln.

Auf diese Weise ist es auch möglich, Inner- und Outer Joins innerhalb einer SQL-Anweisung zu kombinieren.

Beispiele

Die folgenden Beispiele suchen aus den Tabellen KUNDE, AUFTRAG und LEISTUNG Kunden-, Auftrags- und Leistungsnummern heraus. Die Ergebnisse sind abhängig von den verwendeten Joins.

- Es werden nur Kunden berücksichtigt, für die bereits Aufträge mit zugeordneten Leistungen existieren.

```
SELECT k.knr, a.anr, l.lnr
   FROM (kunde k INNER JOIN auftrag a ON k.knr = a.knr)
                 INNER JOIN leistung l ON a.anr = l.anr
   WHERE k.knr BETWEEN 100 AND 107
   knr          anr          lnr
   102          200            1
   102          200            2
   102          200            3
   105          250           10
   105          250           11
   106          211            4
   106          211            5
   106          211            6
   106          211            7
```

- Es werden alle Kunden aus der Tabelle KUNDE berücksichtigt, für die Aufträge existieren, unabhängig davon, ob diesen Aufträgen Leistungen zugeordnet sind oder nicht. Der geklammerte Join-Ausdruck ist die dominante Tabelle für den Outer Join. Für fehlende Leistungsnummern wird der NULL-Wert eingetragen.

```
SELECT k.knr, a.anr, l.lnr
   FROM (kunde k INNER JOIN auftrag a ON k.knr = a.knr)
                 LEFT OUTER JOIN leistung l ON a.anr = l.anr
   WHERE k.knr BETWEEN 100 AND 107
   knr          anr          lnr
   101          300
   102          200            1
   102          200            2
   102          200            3
   105          250           10
   105          250           11
   105          251
   105          305
   106          210
   106          211            4
   106          211            5
   106          211            6
   106          211            7
```

- Es werden alle Kunden aus der Tabelle KUNDE berücksichtigt, unabhängig davon, ob Aufträge existieren oder nicht. Aufträge werden auch dann in die Ergebnistabelle aufgenommen, wenn ihnen noch keine Leistungen zugeordnet sind.

```
SELECT k.knr, a.anr, l.lnr
   FROM (kunde k LEFT OUTER JOIN auftrag a ON k.knr = a.knr)
                 LEFT OUTER JOIN leistung l ON a.anr = l.anr
   WHERE k.knr BETWEEN 100 AND 107
```

  KUNDE ist die dominante Tabelle des geklammerten Outer Join. Der geklammerte Ausdruck ist die dominante Tabelle des äußeren Outer Join. Für fehlende Artikel- und Leistungsnummern wird der NULL-Wert eingetragen.

```
knr          anr          lnr
100
101          300
102          200           1
102          200           2
102          200           3
103
104
105          250          10
105          250          11
105          251
105          305
106          211           4
106          211           5
106          211           6
106          211           7
106          210
107
```

Die folgenden Beispiele beziehen sich auf die Tabellen KUNDE und AUFTRAG. Um die Möglichkeiten eines Outer Join besser darstellen zu können, sollen auch Aufträge ohne Kunden erlaubt sein. Dies bedeutet, dass die Fremdschlüssel-Definitionen für die Tabelle AUFTRAG hier ignoriert werden. Wir nehmen an, dass ein Auftrag mit der Nummer 400 in der Tabelle AUFTRAG enthalten ist, der noch keinem Kunden zugeordnet ist.

- Aus den Tabellen KUNDE und AUFTRAG werden Kundenname und zugehörige Auftragsnummer herausgesucht, und zwar für alle Kunden, auch solche, die zurzeit keinen Auftrag gestellt haben.

```
SELECT kunde.firma, auftrag.anr FROM kunde
       LEFT OUTER JOIN auftrag ON kunde.knr=auftrag.knr
```

  Kunden, die keinen Auftrag erteilt haben, zum Beispiel der Kunde Jonas Fischladen mit Nummer 104, sind in der Ergebnistabelle enthalten. Für die fehlende Auftragsnummer ist der NULL-Wert eingetragen.

```
firma                    anr
Siemens AG
Login GmbH               300
JIKO Gmbh                200
Plenzer Trading
Jonas Fischladen
Pudelshop Anke           250
Pudelshop Anke           251
Pudelshop Anke           305
Foreign Ltd.             210
```

```
 Foreign Ltd.              211
 Externa & Co KG
```

- Aus den Tabellen KUNDE und AUFTRAG werden Kundenname und Auftragsnummer herausgesucht, und zwar auch für Aufträge, denen kein Kunde zugeordnet ist.

  ```
  SELECT kunde.firma, auftrag.anr FROM kunde
         RIGHT OUTER JOIN auftrag ON kunde.knr=auftrag.knr
  ```

  Die Auftragsnummer 400 ist ebenfalls in der Ergebnistabelle enthalten. Für den fehlenden Kunden ist der NULL-Wert eingetragen.

```
firma              anr
 JIKO Gmbh          200
 Foreign Ltd.       210
 Foreign Ltd.       211
 Pudelshop Anke     250
 Pudelshop Anke     251
 Login GmbH         300
 Pudelshop Anke     305
                    400
```

- Aus den Tabellen KUNDE und AUFTRAG werden Kundenname und zugehörige Auftragsnummer herausgesucht, wobei sowohl Kunden ohne Auftrag als auch Aufträge ohne Kunden berücksichtigt werden.

  ```
  SELECT kunde.firma, auftrag.anr FROM kunde
         FULL OUTER JOIN auftrag ON kunde.knr=auftrag.knr
  ```

  Ein fiktiver Auftrag mit der Nummer 400, der noch keinem Kunden zugeordnet ist, ist ebenso in der Ergebnistabelle enthalten wie der Kunde Jonas Fischladen, der zurzeit keinen Auftrag gestellt hat. Für die fehlenden Spaltenwerte werden NULL-Werte eingetragen.

```
firma                    anr
 Siemens AG
 Login GmbH               300
 JIKO Gmbh                200
 Plenzer Trading
 Jonas Fischladen
 Pudelshop Anke           250
 Pudelshop Anke           251
 Pudelshop Anke           305
 Foreign Ltd.             210
 Foreign Ltd.             211
 Externa & Co KG
                          400
```

6.5 Unterabfrage

Eine Unterabfrage ist ein Abfrage-Ausdruck, der in folgenden Fällen verwendet werden kann:

- Als Ausdruck:
  Die Unterabfrage muss eine einspaltige Ergebnistabelle mit höchstens einem Satz liefern. Der Wert der Unterabfrage ist dann der Wert in der Ergebnistabelle bzw. der NULL-Wert, wenn die Ergebnistabelle leer ist.
- In Prädikaten:
  In den Prädikaten ANY, SOME, ALL, IN und EXISTS liefert die Unterabfrage eine Ergebnistabelle.
- In der FROM-Klausel von SELECT-Ausdrücken:
  Die Unterabfrage liefert eine Ergebnistabelle.
- In Join-Ausdrücken:
  Die Unterabfrage liefert eine Ergebnistabelle.

Eine Unterabfrage wird immer in runde Klammern eingeschlossen.

---

unterabfrage `::=` (*abfrageausdruck*)

---

abfrageausdruck

Abfrage-Ausdruck, der die Ergebnistabelle liefert.

Bei Unterabfragen, die nicht im Prädikat EXISTS oder in einer FROM-Klausel angegeben sind, darf die Ergebnistabelle nur einfache Spalten oder multiple Spalten mit Dimension 1 enthalten.

6.5.1 Korrelierte Unterabfragen

Bei einem geschachtelten Abfrage-Ausdruck heißt eine innere Unterabfrage **korrelierte Unterabfrage**, wenn sie sich auf Spalten einer äußeren Tabelle bezieht, das heißt einer Tabelle, die in einem der äußeren Abfrage-Ausdrücke verwendet wird.

Mit Hilfe von korrelierten Unterabfragen können Sie Beziehungen zwischen den Spaltenwerten einer Spalte bestimmen.

Beispiel

> Bei einer Personentabelle mit einer Spalte, die für jede Person das Alter enthält, können Sie feststellen, welche Personen genau das Durchschnittsalter haben (siehe auch Beispiel unten).

Nicht korrelierte Unterabfragen brauchen nur einmal ausgewertet zu werden. Korrelierte Unterabfragen müssen mehrfach für unterschiedliche Sätze der äußeren Tabelle ausgewertet werden. Ist die Unterabfrage geschachtelt, erfolgt die Auswertung von innen nach außen.

Beispiele

Bei der folgenden Abfrage handelt es sich um eine korrelierte Unterabfrage:

```
SELECT DISTINCT atext FROM auftrag WHERE EXISTS
(SELECT * FROM leistung WHERE leistung.anr = auftrag.anr)
```

Die innere Unterabfrage in der WHERE-Klausel bezieht sich auf die Spalte ANR der Tabelle AUFTRAG in der äußeren Abfrage. AUFTRAG.ANR wird auch Außenreferenz genannt, da die Spalte eine Tabelle in der äußeren Abfrage referenziert. Die Abfrage wird ausgewertet, indem für den ersten Satz der Tabelle AUFTRAG der Wert von AUFTRAG.ANR bestimmt wird und die Unterabfrage mit diesem Wert ausgewertet und das Ergebnis in die äußere Abfrage eingesetzt wird. Anschließend wird dies für den zweiten Wert von AUFTRAG.ANR wiederholt, usw. Die Abfrage liefert folgende Ergebnistabelle:

atext
Mitarbeiterschulung
Datenbank-Entwurf Kunden
Serienbrief-Einweisung

Für jeden Auftrag aus der Tabelle LEISTUNG die Leistungen heraussuchen, die über der durchschnittlichen Leistung für diesen Auftrag liegen:

```
SELECT l1.lnr, l1.anr, l1.lanz*l1.lsatz
   FROM leistung l1
   WHERE l1.lanz*l1.lsatz >
   (SELECT AVG (l2.lanz*l2.lsatz) FROM leistung l2 WHERE l1.anr=l2.anr)
```

Abfrage-Ausdrücke können beliebig geschachtelt werden:

```
SELECT firma, knr FROM kunde WHERE knr IN
   (SELECT knr FROM auftrag WHERE anr IN
       (SELECT anr FROM leistung WHERE (lsatz*lanz) IN
           (SELECT MAX(lsatz*lanz) FROM leistung)))
```

Da es sich um nicht korrelierte Unterabfragen handelt, wird jede Unterabfrage einmal ausgewertet, das Ergebnis wird jeweils in die äußere Abfrage eingesetzt.

Ergebnistabelle

firma	knr
Foreign Ltd.	106

6.6 Verbindung von Abfrage-Ausdrücken mit UNION

abfrageausdruck ::= { *select_ausdruck* | TABLE *tabelle* | *join_ausdruck* | *(abfrageausdruck)* }
[UNION [ALL | DISTINCT] *abfrageausdruck*]

select_ausdruck

SELECT-Ausdruck, siehe Abschnitt „SELECT-Ausdruck“.

TABLE *tabelle*

Tabellenabfrage, siehe Abschnitt „TABLE - Tabellenabfrage“.

join_ausdruck

Join-Ausdruck, siehe Abschnitt „Join-Ausdruck“.

(*abfrageausdruck*)

Unterabfrage, siehe Abschnitt „Unterabfrage“.

UNION

Die UNION-Klausel verbindet zwei Abfrage-Ausdrücke. Die Ergebnistabelle enthält alle Sätze, die in der ersten oder zweiten Ergebnistabelle vorkommen. Sie können mehr als zwei Ergebnistabellen verbinden, wenn Sie die UNION-Klausel mehrmals verwenden.

Für die Verknüpfung von Abfrage-Ausdrücken mittels UNION müssen folgende Bedingungen erfüllt sein:

- Die Ergebnistabellen der beiden UNION-Operanden müssen dieselbe Anzahl von Spalten haben und die Datentypen entsprechender Spalten müssen verträglich sein (siehe Abschnitt „Verträglichkeit von Datentypen“). Der Datentyp einer Ergebnisspalte ergibt sich aus den Regeln, die im Abschnitt „Datentyp der Ergebnisspalten bei UNION“ beschrieben sind.
- Wenn die entsprechenden Spalten der beiden Ausgangstabellen den gleichen Namen haben, erhält die Ergebnisspalte diesen Namen. Ansonsten ist der Name der Ergebnisspalte undefiniert.
- Es dürfen nur einfache Spalten ausgewählt werden.

Mit der UNION-Klausel gebildete Abfrage-Ausdrücke sind nicht änderbar.

ALL

Doppelte Sätze in der Ergebnistabelle bleiben erhalten.

DISTINCT

Doppelte Sätze werden entfernt. Erfolgt keine Angabe von ALL oder DISTINCT, ist DIS-TINCT voreingestellt.

> Im Gegensatz zum SELECT-Ausdruck ist bei UNION die Voreinstellung DISTINCT. Da eine Entfernung von doppelten Sätzen aufwändig sein kann, wird bei UNION die Einstellung ALL empfohlen, wenn die Anwendung auf das Entfernen doppelter Sätze verzichten kann.

Datentyp der Ergebnisspalten bei UNION

Bei der Verknüpfung von zwei Abfrage-Ausdrücken mit UNION ist der Datentyp der Ergebnisspalten durch folgende Regeln festgelegt.

- Beide Ausgangsspalten haben den Datentyp CHAR:
  Ergebnisspalte hat Datentyp CHAR mit der größeren Länge.
- Eine Ausgangsspalte hat Datentyp VARCHAR und die andere Ausgangsspalte CHAR oder VARCHAR:
  Die Ergebnisspalte hat Datentyp VARCHAR mit der größeren bzw. größeren maximalen Länge.
- Beide Ausgangsspalten haben den Datentyp NCHAR:
  Ergebnisspalte hat Datentyp NCHAR mit der größeren Länge.
- Eine Ausgangsspalte hat Datentyp NVARCHAR und die andere Ausgangsspalte NCHAR oder NVARCHAR:
  Die Ergebnisspalte hat Datentyp NVARCHAR mit der größeren bzw. größeren maximalen Länge.
- Beide Ausgangsspalten haben ganzzahligen Typ oder Festpunktzahl-Typ (INT, SMALLINT, NUMERIC, DEC):
  Die Ergebnisspalte hat den Datentyp Ganz- oder Festpunktzahl.
  - Die Nachkommastellenzahl ist die größere der beiden Nachkommastellenzahlen der Ausgangsspalten.
  - Die Gesamtstellenzahl ist die größere der beiden Vorkommastellenzahlen plus die größere der beiden Nachkommastellenzahlen der beiden Ausgangsspalten, höchstens jedoch 31.
- Eine Ausgangsspalte hat Gleitpunktzahl-Typ (REAL, DOUBLE, FLOAT), die andere hat einen beliebigen numerischen Datentyp:
  Die Ergebnisspalte hat den Datentyp DOUBLE PRECISION.
- Beide Ausgangsspalten haben Zeitdatentyp:
  Beide Spalten müssen denselben Zeitdatentyp haben und die Ergebnisspalte hat auch diesen Datentyp.

Beispiele

Alle Auftragsnummern ermitteln, deren zugehöriger Auftragswert wenigstens 10.000 Euro beträgt oder deren Solldatum vor dem angegebenen Datum liegt.

```
SELECT anr FROM leistung GROUP BY anr
   HAVING SUM(lanz * lsatz * (1 + mwsatz)) > = 10000.00
   UNION DISTINCT
   SELECT anr FROM auftrag WHERE fertigsoll <= DATE '<date>'
```

Die Namen der Firmen sollen ermittelt werden, für die die Auftragsunterlagen bereits in der Ablage sind oder bei denen Leistungen vor dem angegebenen Datum erbracht wurden.

```
SELECT k.firma FROM kunde k, auftrag a WHERE k.knr = a.knr
   AND a.anr IN
   (SELECT a.anr FROM auftrag a WHERE a.astnr > 4
                 UNION
                 SELECT DISTINCT l.anr FROM leistung l
                        WHERE l.ldatum < DATE'<date>')
```

Der UNION-Ausdruck in der Unterabfrage liefert als Ergebnistabelle die Auftragsnummern 200 und 211.

Als Ergebnistabelle ergibt sich damit

firma

JIKO GmbH

Foreign Ltd

6.7 Verbindung von Abfrage-Ausdrücken mit EXCEPT

abfrageausdruck ::= { *select_ausdruck* | TABLE *tabelle* | *join_ausdruck* | (*abfrageausdruck*) }
[EXCEPT [DISTINCT] *abfrageausdruck*]

select_ausdruck

SELECT-Ausdruck, siehe Abschnitt „SELECT-Ausdruck“.

TABLE *tabelle*

Tabellenabfrage, siehe Abschnitt „TABLE - Tabellenabfrage“.

join_ausdruck

Join-Ausdruck, siehe Abschnitt „Join-Ausdruck“.

(*abfrageausdruck*)

Unterabfrage, siehe Abschnitt „Unterabfrage“.

EXCEPT

Die EXCEPT-Operation ist vergleichbar mit der Differenz zweier Mengen in der Mengenlehre. Die Ergebnistabelle enthält alle Sätze der ersten Tabelle, die nicht in der zweiten Tabelle vorkommen.

Für die Verknüpfung von Abfrage-Ausdrücken mittels EXCEPT müssen folgende Bedingungen erfüllt sein:

- Die Ergebnistabellen der beiden EXCEPT-Operanden müssen dieselbe Anzahl von Spalten haben.
- Die Datentypen entsprechender Spalten müssen verträglich sein (siehe Abschnitt „Verträglichkeit von Datentypen“).
  Der Datentyp einer Ergebnisspalte ergibt sich aus den Regeln, die im Abschnitt „Datentyp der Ergebnisspalten bei EXCEPT“ beschrieben sind.

DISTINCT

Doppelte Sätze werden aus der Resultattabelle entfernt. Erfolgt keine Angabe, ist DISTINCT voreingestellt.

Datentyp der Ergebnisspalten bei EXCEPT

Bei der Verknüpfung von zwei Abfrage-Ausdrücken mit EXCEPT ist der Datentyp der Ergebnisspalten ebenso wie bei UNION durch folgende Regeln festgelegt.

- Beide Ausgangsspalten haben den Datentyp CHAR:
  Ergebnisspalte hat Datentyp CHAR mit der größeren Länge.

- Eine Ausgangsspalte hat Datentyp VARCHAR und die andere Ausgangsspalte CHAR oder VARCHAR:
  Die Ergebnisspalte hat Datentyp VARCHAR mit der größeren bzw. größeren maximalen Länge.
- Beide Ausgangsspalten haben den Datentyp NCHAR:
  Ergebnisspalte hat Datentyp NCHAR mit der größeren Länge.
- Eine Ausgangsspalte hat Datentyp NVARCHAR und die andere Ausgangsspalte NCHAR oder NVARCHAR:
  Die Ergebnisspalte hat Datentyp NVARCHAR mit der größeren bzw. größeren maximalen Länge.
- Beide Ausgangsspalten haben ganzzahligen Typ oder Festpunktzahl-Typ (INT, SMALLINT, NUMERIC, DEC):
  Die Ergebnisspalte hat den Datentyp Ganz- oder Festpunktzahl.
  - Die Nachkommastellenzahl ist die größere der beiden Nachkommastellenzahlen der Ausgangsspalten.
  - Die Gesamtstellenzahl ist die größere der beiden Vorkommastellenzahlen plus die größere der beiden Nachkommastellenzahlen der beiden Ausgangsspalten, höchstens jedoch 31.
- Eine Ausgangsspalte hat Gleitpunktzahl-Typ (REAL, DOUBLE, FLOAT), die andere hat einen beliebigen numerischen Datentyp:
  Die Ergebnisspalte hat den Datentyp DOUBLE PRECISION.
- Beide Ausgangsspalten haben Zeitdatentyp:
  Beide Spalten müssen denselben Zeitdatentyp haben und die Ergebnisspalte hat auch diesen Datentyp.

Beispiel

Alle Kundennummern ermitteln, für die keine geplanten oder vertraglich vereinbarten Aufträge existieren.

```
SELECT knr FROM kunde
   EXCEPT DISTINCT
   SELECT knr FROM auftrag WHERE astnr < 3
```

6.8 Änderbarkeit von Abfrage-Ausdrücken

Über die Änderbarkeit von Abfrage-Ausdrücken ist Folgendes festgelegt:

- ob ein View änderbar ist
- ob eine Basistabelle oder ein änderbarer View über einen Cursor geändert werden kann

Eine Basistabelle ist änderbar.
Eine Tabellenfunktion liefert eine nicht-änderbare („read only“) Tabelle.

6.8.1 Regeln für änderbare Abfrage-Ausdrücke

Ein Abfrage-Ausdruck ist änderbar, wenn folgende Bedingungen erfüllt sind:

- Der Abfrage-Ausdruck enthält keinen Join-Ausdruck.
- Der Abfrage-Ausdruck enthält keine UNION- oder EXCEPT-Operation.
- In der SELECT-Liste dürfen nur Spaltennamen angegeben werden. Andere Elemente eines Ausdrucks, beispielsweise Unterabfragen, Funktionsaufrufe oder Literale, sind nicht erlaubt. Einfache Spalten dürfen nicht mehrfach angegeben werden. Teilbereiche von multiplen Spalten dürfen sich nicht überlappen.
- In der FROM-Klausel darf nur eine Tabelle angegeben sein oder eine änderbare Unterabfrage. Ist eine Tabelle angegeben, muss sie eine Basistabelle oder ein änderbarer View sein.
- In der WHERE-Klausel darf keine Unterabfrage vorkommen.
- Das Schlüsselwort DISTINCT darf nicht angegeben werden.
- Der SELECT-Ausdruck darf keine GROUP BY- oder HAVING-Klausel enthalten.

6.8.2 Änderbarer View

Ein View ist änderbar, wenn der Abfrage-Ausdruck, mit dem der View definiert wurde, änderbar ist. Bei INSERT, MERGE, UPDATE und DELETE kann ein änderbarer View angegeben werden.

6.8.3 Ändern über Cursor

Eine Tabelle ist über einen Cursor änderbar, wenn die Cursorbeschreibung änderbar ist, d.h. der zugrunde liegende Abfrage-Ausdruck ist änderbar und es ist keine ORDER-BY-Klausel angegeben. Zusätzlich dürfen in der Cursorvereinbarung keine SCROLL-Klausel und keine FOR READ ONLY-Klausel angegeben werden.

Mit DELETE...WHERE CURRENT OF können Sätze in der änderbaren Tabelle über den Cursor gelöscht werden.

Mit UPDATE...WHERE CURRENT OF können Sätze in der änderbaren Tabelle über den Cursor geändert werden.

7 Routinen

SESAM/SQL unterscheidet folgende Routinen:

- **Prozeduren (Stored Procedures)**
- **User Defined Functions (UDF)**.

> ¡ In SESAM/SQL wird der Oberbegriff **Routine** für Prozeduren und User Defined Functions (UDFs) verwendet, wenn die Information sowohl für Prozeduren als auch für UDFs gilt.
>
> Der Oberbegriff „SQL-invoked routine“ der SQL-Norm wird in SESAM/SQL nicht verwendet.

Dieses Kapitel beschreibt zunächst Gemeinsamkeiten und Unterschiede zwischen Prozeduren und UDFs.

Es folgen in eigenen Abschnitten die detaillierte Beschreibung von Prozeduren (Stored Procedures) und von User Defined Functions (UDFs).

Im Anschluss daran folgen Informationen zu den Themen, in denen sich Prozeduren und UDFs nicht oder nur geringfügig unterscheiden:

- EXECUTE-Privileg für Routinen
- Informationen über Routinen
- Pragmas in Routinen
- Kontrollanweisungen in Routinen
- COMPOUND-Anweisung in Routinen
- Diagnoseinformationen in Routinen

Gemeinsamkeiten von Routinen

In einer Routine werden Abläufe von SQL-Anweisungen in der Datenbank gespeichert und verwaltet, die später mit einem einzigen Aufruf ausgeführt werden können. Eine Routine ist vergleichbar mit einem Unterprogramm, das vollständig, also ohne Datenaustausch mit dem Anwendungsprogramm, im DBH abläuft.

Im Gegensatz zu einem Unterprogramm (in ESQL-COBOL) kann eine Routine auf verschiedenen Clients mit unterschiedlichen Programmiersprachen eingesetzt werden (z.B. über JDBC).

Über Routinen kann eine Zentralisierung und Kontrolle sämtlicher Datenbankzugriffe erreicht werden. Auch einzelne SQL-Anweisungen können auf diese Weise eingeschalt werden. Sie können dann nach dem „Baukastenprinzip“ in andere Routinen und SQL-Anweisungen eingebaut werden.

Routinen können auch zur Schreib-Erleichterung eingesetzt werden.

Der Anwendungsprogrammierer benötigt keine Kenntnis über die die Struktur der Datenbank. Die Routine kann von einem Datenbankspezialisten erstellt werden, der seinerseits (außer SQL) keine Programmierkenntnisse benötigt.

Änderungen an der Datenbankstruktur wirken sich nicht notwendigerweise auf die Anwendungsprogramme aus. Ggf. genügt eine Modifikation von Routinen. Das erneute Übersetzen und Binden von Programmen erübrigt sich in solchen Fällen.

Zur Sicherheit wird für die Ausführung nur das EXECUTE-Privileg für die jeweilige Routine benötigt. Pauschale Tabellen- und Spalten-Privilegien sind nicht mehr nötig.

Routinen werden (revisionssicher) direkt in der Datenbank abgelegt. Ein separates Management zur Verwaltung von Routinen außerhalb der Datenbank ist nicht erforderlich.

Unterschiede von Prozeduren und User Defined Functions

Prozeduren und UDFs haben einen fast identischen Funktionsumfang. In UDFs von SESAM/SQL sind aber SQL-Anweisungen zum Ändern von Daten nicht erlaubt.

Prozeduren und UDFs unterscheiden sich weiter durch die Art ihres Aufrufes und ihrer Rückkehr-Information:

- Prozeduren werden mit der SQL-Anweisung CALL aufgerufen.
  Sie haben beliebig viele Ausgabeparameter, aber keinen Rückgabewert.
- UDFs werden durch ihren Funktionsaufruf in einem Ausdruck aufgerufen.Sie haben genau einen Rückgabewert.

UDFs können in Views aufgerufen werden. Prozeduren können dies nicht.

7.1 Prozeduren (Stored Procedures)

In SESAM/SQL wird für „Stored Procedure“ der Begriff **Prozedur** verwendet.

7.1.1 Erzeugen einer Prozedur

Eine Prozedur wird mit der SQL-Anweisung CREATE PROCEDURE erzeugt, siehe "CREATE PROCEDURE - Prozedur erzeugen". Eine Prozedur kann auch im Rahmen der SQL-Anweisung CREATE SCHEMA erzeugt werden, siehe "CREATE SCHEMA - Schema erzeugen".

Prozeduren können mit Eingabe-, Ein-/Ausgabe- und Ausgabeparametern definiert werden.

> ¡ *Empfehlung* Parameternamen sollten sich (z.B. durch Vergabe eines Präfixes wie `par_`) von Spaltennamen unterscheiden.

Beim Erzeugen der Prozedur muss der aktuelle Berechtigungsschlüssel das EXECUTE-Privileg für die in der Prozedur direkt aufgerufenen Routinen besitzen. Zusätzlich muss er für alle Tabellen und Spalten, die in der Prozedur angesprochen werden, diejenigen Privilegien besitzen, die benötigt werden, um die in der Prozedur enthaltenen DML-Anweisungen ausführen zu können.

Der Prozedurtext ist in SESAM/SQL vollständig in der Programmiersprache SQL geschrieben. Folgende SQL-Anweisungen zur Datensuche und -manipulation sind in Prozeduren zulässig, siehe Abschnitt „CREATE PROCEDURE - Prozedur erzeugen“:

SQL-Anweisung ohne Cursor	**Funktion in der Prozedur**	siehe
SELECT	Liest einen einzelnen Satz	"SELECT - Einzelnen Satz lesen"
INSERT	Fügt Sätze in eine bestehende Tabelle ein	"INSERT - Sätze in Tabelle einfügen"
UPDATE	Ändert in einer Tabelle die Spalten der Sätze, die eine bestimmte Suchbedingung erfüllen	"UPDATE - Spaltenwerte ändern"
DELETE	Löscht in einer Tabelle die Sätze, die eine bestimmte Suchbedingung erfüllen	"DELETE - Sätze löschen"
MERGE	Ändert in Abhängigkeit von einer bestimmten Bedingung Sätze in einer Tabelle oder fügt Sätze in eine Tabelle ein	"MERGE - Sätze in Tabelle einfügen oder Spaltenwerte ändern"
SQL-Anweisung mit Cursor	**Funktion in der Prozedur**	siehe
OPEN	Öffnet einen lokalen Cursor	"OPEN - Cursor öffnen"
FETCH	Positioniert einen lokalen Cursor und liest ggf. den aktuellen Satz	"FETCH - Cursor positionieren und Satz lesen"
UPDATE	Ändert in einer Tabelle die Spalten des Satzes, auf den der Cursor positioniert ist	"UPDATE - Spaltenwerte ändern"
DELETE		"DELETE - Sätze löschen"

	Löscht in einer Tabelle den Satz, auf den der Cursor positioniert ist	
CLOSE	Schließt einen lokalen Cursor	"CLOSE - Cursor schließen"

Tabelle 25: SQL-Anweisungen zur Datenmanipulation in Prozeduren

Neben obigen SQL-Anweisungen kann eine Prozedur auch Kontrollanweisungen (siehe Abschnitt „Kontrollanweisungen in Routinen“) und Diagnoseanweisungen(siehee Abschnitt „Diagnoseinformationen in Routinen“) enthalten.

Eine Prozedur darf keine dynamisch formulierten SQL-Anweisungen oder Cursorbeschreibungen enthalten, siehe Abschnitt „Dynamische SQL“.

Der aktuelle Berechtigungsschlüssel erhält automatisch das EXECUTE-Privileg für die erzeugte Prozedur. Hat er für die betreffenden Privilegien sogar die Berechtigung, diese weitergeben zu dürfen, dann darf er auch das EXECUTE-Privileg an andere Berechtigungsschlüssel weitergegeben.

Eine SQL-Anweisung in einer Prozedur darf auf die Parameter der Prozedur und (wenn die Anweisung Teil einer COMPOUND-Anweisung ist) auf lokale Variablen, jedoch nicht auf Benutzervariablen zugreifen.

Kommentare

Beschreibende Kommentare (siehe "Kommentar") können beliebig in eine Prozedur eingefügt werden.

7.1.2 Ausführen einer Prozedur

Eine Prozedur wird mit der SQL-Anweisung CALL ausgeführt, siehe "CALL - Prozedur ausführen". Eine Prozedur kann auch über eine dynamisch formulierte CALL-Anweisung aufgerufen werden.

Wenn eine Prozedur Eingabeparameter erwartet, dann müssen die entsprechenden Werte (die Argumente) in der CALL-Anweisung an die Prozedur übergeben werden.

Ausgabewerte von Prozeduren, die außerhalb einer Routine aufgerufen werden, werden in entsprechenden Benutzervariablen oder im SQL-Deskriptorbereich abgelegt. Ausgabewerte von Prozeduren, die in einer übergeordneten Routine aufgerufen werden, werden in Ausgabeparameter oder in lokale Variablen der übergeordneten Prozedur eingetragen.

Zur Ausführung einer Prozedur benötigt der aktuelle Berechtigungsschlüssel das EXECUTE-Privileg für die auszuführende Prozedur, nicht aber diejenigen Privilegien, die benötigt werden, um die in der Prozedur enthaltenen DML-Anweisungen ausführen zu können. Zusätzlich werden die SELECT-Privilegien für die Tabellen benötigt, die in den Aufrufparametern der Routine über Unterabfragen angesprochen werden.

7.1.3 Löschen einer Prozedur

Eine Prozedur wird mit der SQL-Anweisung DROP PROCEDURE gelöscht, siehe "DROP PROCEDURE - Prozedur löschen".

7.1.4 Beispiele für Prozeduren

Beispiel 1: Zugangsprüfung

Die nachfolgende Prozedur KUNDEN_LOGIN realisiert eine einfache Form der Zugangsprüfung für Kunden. Sie ist Bestandteil der Prozedurenbeispiele in der Beispieldatenbank von SESAM/SQL (siehe „ Basishandbuch“).

> In der Beispieldatenbank finden Sie weitere, ausführliche Beispielprozeduren, eingebettet in ein Bestellsystem.

Die Prozedur KUNDEN_LOGIN benutzt nur die Tabelle KONTAKT aus der Beispieldatenbank. Es wird geprüft, ob der Kunde bereits in der Tabelle gespeichert ist.

```
************************************************************************
* Prozedur KUNDEN_LOGIN definieren
************************************************************************
SQL CREATE PROCEDURE KUNDEN_LOGIN                                    1,
( -
   IN  PAR_KUNDENNR        INTEGER,                                  2.
   IN  PAR_KONTAKTNR       INTEGER,
   OUT PAR_STATUS          CHAR(40),
   OUT PAR_ANREDE          CHAR(20),
   OUT PAR_NACHNAME        CHAR(25)
)
READS SQL DATA                                                       3.
BEGIN                                                                4.
   /* Variablen Definition     */                                   5.
   DECLARE VAR_EOD SMALLINT DEFAULT 0;
   /* Handler Definition       */                                   6.
   DECLARE CONTINUE HANDLER FOR NOT FOUND
      SET VAR_EOD = 1;                                               7.
   /* Anweisungen              */                                   8.
   SET PAR_ANREDE   = ' ';
   SET PAR_NACHNAME = ' ';
   /* Pruefen, ob Kunde schon bekannt ist      */
   SELECT ANREDE, NACHNAME INTO PAR_ANREDE, PAR_NACHNAME
      FROM KONTAKT
      WHERE KONR = PAR_KONTAKTNR
      AND   KNR  = PAR_KUNDENNR;
   IF VAR_EOD = 1 THEN                                               9.
      SET PAR_STATUS = 'Kunde unbekannt';
   ELSE
      SET PAR_STATUS = 'Login erfolgreich';
   END IF;
END                                                                 10.
```

1. Prozedurkopf mit der Angabe des Prozedurnamens (Datenbank- und Schemaname sind voreingestellt).
2. Liste der Prozedurparameter.
3. Die Prozedur kann SQL-Anweisungen zum Lesen von Daten enthalten, jedoch keine SQL-Anweisungen zum Ändern von Daten. Die (einzige) Prozeduranweisung ist eine (nicht-atomare) COMPOUND-Anweisung. Sie führt weitere Prozeduranweisungen in einem gemeinsamen Kontext aus.
4. Definition lokaler Prozedurvariablen.
5. Definition der Fehlerbehandlung in Abhängigkeit vom SQLSTATE.

6. In diesem Fall wird die Prozedur fortgesetzt wenn ein SQLSTATE der Klasse 02xxx (keine Daten) auftritt.
7. Im Fehlerfall wird die lokale Variable `VAR_EOD` gesetzt.
8. Es folgen die Prozeduranweisungen.
9. In Abhängigkeit vom Ergebnis der Abfrageanweisung werden die Ausgabefelder der Prozedur versorgt.
10. Ende von COMPOUND-Anweisung und Prozedur.

Beispiel 2: komplexe COMPOUND-Anweisung

Die nachfolgende Prozedur `MyTables` besteht aus einer komplexen Compound-Anweisung und zeigt verschiedene Möglichkeiten der Fehlerbehandlung. Sie speichert in der zentralen Basistabelle `mySchema.myTabs` die Namen derjenigen Tabellen ab, auf die der aktuelle Berechtigungsschlüssel zugreifen darf.

Der Eingabeparameter `par_type` gibt vor, ob Basistabellen oder Views zu berücksichtigen sind. Bei `par_type='B'` werden die Namen der Basistabellen, bei `par_type='V'` die Namen der Views gespeichert. Geliefert werden folgende Ausgabeparameter:

`par_nbr_tables`

Anzahl der für den aktuellen Benutzer insgesamt gespeicherten Tabellennamen des jeweiligen Tabellentyps (Basistabelle oder View)

`par_nbr_new_tables`

Anzahl der für den aktuellen Benutzer durch den Prozeduraufruf zusätzlich abgespeicherten Tabellennamen

`par_message`

Meldungstext (OK oder Fehlerhinweis)

```
-- Procedure header
CREATE PROCEDURE ProcSchema.MyTables
   ( IN par_type CHAR(1), OUT par_message CHAR(80),
     OUT par_nbr_tables INTEGER, OUT par_nbr_new_tables INTEGER )
   MODIFIES SQL DATA
-- Procedure body, COMPOUND statement, declaration section
myTab: BEGIN ATOMIC
   DECLARE var_table_type CHAR(18);
   DECLARE var_schema_name,var_table_name CHAR(31);
   DECLARE var_eot SMALLINT DEFAULT 0;
   DECLARE var_nbr_old_tables INTEGER DEFAULT 0;
   DECLARE myCursor CURSOR FOR
           SELECT table_schema, table_name
           FROM information_schema.tables
           WHERE table_type = var_table_type;
-- Error routines
   DECLARE EXIT HANDLER FOR SQLSTATE '42SND'
           SET par_message = 'catalog ' || CURRENT_REFERENCED_CATALOG
                                        || ' not accessible';
   DECLARE CONTINUE HANDLER FOR SQLSTATE '23SA5'
   -- Primary key not unique
           SET var_nbr_old_tables = var_nbr_old_tables + 1;
   DECLARE EXIT HANDLER FOR SQLSTATE '42SQK'
           SET par_message = 'table MyTabs not accessible';
   DECLARE UNDO HANDLER FOR SQLEXCEPTION
           BEGIN -- COMPOUND statement
           SET par_message = 'unexpected error';
           SET par_nbr_tables = 0;
```

```
            SET par_nbr_new_tables = 0;
            END;
   DECLARE CONTINUE HANDLER FOR SQLWARNING
            SET par_message = 'warning ignored';
   DECLARE CONTINUE HANDLER FOR NOT FOUND
            SET var_eot = 1;
-- Set initial values
   SET par_message = 'OK';
   SET par_nbr_tables = 0;
   SET par_nbr_new_tables = 0;
   IF par_type = 'V' THEN SET var_table_type = 'VIEW';
      ELSEIF par_type = 'B' THEN SET var_table_type = 'BASE TABLE';
      ELSE SET par_message = 'wrong input parameter par_type';
            LEAVE myTab;
   END IF;
-- Procedure statements
   OPEN myCursor;
   loop1: LOOP
          FETCH myCursor INTO var_schema_name, var_table_name;
          IF var_eot = 1 -- Set by error handler for error class 'not found'
             THEN LEAVE loop1; -- End of tables reached
          END IF;
          INSERT INTO mySchema.myTabs VALUES
                 (var_schema_name, var_table_name, var_table_type,
                  current_user, current_date);
          SET par_nbr_tables = par_nbr_tables + 1;
   END LOOP loop1;
   CLOSE myCursor;
   SET par_nbr_new_tables = par_nbr_tables - var_nbr_old_tables;
   -- var_nbr_old_tables set by error handler for SQLSTATE '23SA5'
END myTab
```

Beispiel 3: Unterschiedliche CALL-Aufrufe

Die Prozedur `min_leistungssatz` liefert anhand der übergebenen Auftragsnummer den niedrigsten Leistungssatz für diesen Auftrag zurück.
Wenn als Auftragsnummer der NULL-Wert übergeben wurde, dann wird als Leistungssatz der Wert `-999` zurückgeliefert.
Wenn die Auftragsnummer existiert, aber in allen betroffenen Sätzen der Leistungssatz nicht signifikant ist, dann wird der NULL-Wert zurückgeliefert.
Wenn die Auftragsnummer nicht existiert, dann wird die CALL-Anweisung mit SQLSTATE („keine Daten“) beendet.

```
-- Procedure header
   CREATE PROCEDURE min_leistungssatz
   ( IN in_anr CHAR(8), OUT out_lsatz NUMERIC(6) )
   READS SQL DATA
-- Procedure body
   IF in_anr IS NULL THEN out_lsatz = -999;
                     ELSE SELECT MIN(lsatz) INTO out_lsatz FROM leistung
                                            WHERE anr = in_anr;
   END IF
```

Anhand dieser Prozedur sollen die Reaktionen auf verschiedene CALL-Aufrufe der Prozedur dargestellt werden.

Zu beachten ist, dass die Parameter `in_anr` und `out_lsatz` keine Indikatoren haben (nicht erlaubt). Die Signifikanz von `in_anr` wird direkt über `IS NULL` geprüft. Dem Ausgabeparameterr `out_lsatz` kann in der INTO-Klausel direkt der NULL-Wert zugewiesen werden.

Betrachtet werden nun verschiedene statische CALL-Anweisungen. Das Argument für den Eingabewert kann die unterschiedlichsten Darstellungen haben. Als Argument für den Ausgabewert muss dagegen immer eine Benutzervariable angegeben werden. Sie muss einen numerischen Datentyp haben (kompatibel mit NUMERIC(6)). Sinnvollerweise sollte auch eine Indikatorvariable verwendet werden, die vor dem CALL-Aufruf mit -1 zu initialisieren ist. Andernfalls muss die Benutzervariable selbst mit einem (gemäß ihrem Datentyp) korrekten Wert initialisiert worden sein.

```
CALL min_leistungssatz(:anr, :lsatz INDICATOR :ind-lsatz)
```

Der Eingabewert wird als Benutzervariable übergeben. Da der NULL-Wert zurückgeliefert werden kann, ist die Angabe einer Indikatorvariablen für den Ausgabewert sinnvoll.

```
CALL min_leistungssatz(:anr :ind-anr, :lsatz :ind-lsatz)
```

Wie oben, aber durch das Setzen von `:ind-anr` auf -1 kann auch der NULL-Wert übergeben werden.

```
CALL min_leistungssatz('A#123456', :lsatz)
```

Der konkrete Eingabewert lautet `A#123456`. Wenn dafür der NULL-Wert zurückgeliefert werden sollte, dann fehlt die Angabe einer Indikatorvariablen, was zu einem SQLSTA-TE SEW2202 führt.

```
CALL min_leistungssatz(CAST(NULL AS CHAR (8)), :lsatz)
```

Da der Eingabewert NULL ist, wird der Wert `-999` zurückgeliefert. Da die Benutzervariablee `:lsatz` keinen Indikator hat, muss sie vor dem Aufruf mit einem korrekten Wert (abhängig vom Datentyp) initialisiert worden sein.

```
CALL min_leistungssatz((SELECT MAX(anr) FROM leistung),:lsatz :ind-lsatz)
```

Der Eingabewert ist die höchste Auftragsnummer. Da der NULL-Wert zurückgeliefert werden kann, ist die Angabe einer Indikatorvariablen für den Ausgabewert sinnvoll. Wenn die Tabelle `leistung` leer ist, dann wird damit der NULL-Wert zurückgegeben.

7.2 User Defined Functions (UDFs)

In SESAM/SQL wird für „User Defined Function“ die Abkürzung **UDF** verwendet.

> UDFs können in fast allen Ausdrücken durch ihren Funktionsaufruf verwendet werden. Sie können in den DML-Anweisungen und in den Utility-Anweisungen EXPORT ... WHERE und UNLOAD ONLINE vorkommen.

7.2.1 Erzeugen einer UDF

Eine UDF wird mit der SQL-Anweisung CREATE FUNCTION erzeugt, siehe "CREATE FUNCTION - User Defined Function (UDF) erzeugen". Eine UDF kann auch im Rahmen der SQL-Anweisung CREATE SCHEMA erzeugt werden, siehe "CREATE SCHEMA - Schema erzeugen".

UDFs können mit Eingabeparametern definiert werden.

> *Empfehlung* Parameternamen sollten sich (z.B. durch Vergabe eines Präfixes wie `par_`) von Spaltennamen unterscheiden.

Beim Erzeugen der UDF muss der aktuelle Berechtigungsschlüssel das EXECUTE-Privileg für die in der UDF direkt aufgerufenen Routinen besitzen. Zusätzlich muss er für alle Tabellen und Spalten, die in der UDF angesprochen werden, diejenigen (SELECT-)Privilegien besitzen, die benötigt werden, um die in der Routine enthaltenen DML-Anweisungen ausführen zu können.

Der Text der UDF ist in SESAM/SQL vollständig in der Programmiersprache SQL geschrieben. Folgende SQL-Anweisungen zur Datensuche sind in UDFs zulässig, siehe Abschnitt „CREATE FUNCTION - User Defined Function (UDF) erzeugen“:

SQL-Anweisung ohne Cursor	**Funktion in der UDF**	**siehe**
SELECT	Liest einen einzelnen Satz	"SELECT - Einzelnen Satz lesen"
SQL-Anweisung mit Cursor		
OPEN	Öffnet einen lokalen Cursor	"OPEN - Cursor öffnen"
FETCH	Positioniert einen lokalen Cursor und liest ggf. den aktuellen Satz	"FETCH - Cursor positionieren und Satz lesen"
CLOSE	Schließt einen lokalen Cursor	"CLOSE - Cursor schließen"

Tabelle 26: SQL-Anweisungen zur Datenmanipulation in UDFs

SQL-Anweisungen zum Ändern von Daten (INSERT, UPDATE, DELETE, MERGE) sind in den UDFs von SESAM /SQL **nicht** zulässig.

Neben obigen SQL-Anweisungen kann eine Prozedur auch Kontrollanweisungen (siehe Abschnitt „Kontrollanweisungen in Routinen“) und Diagnoseanweisungen(siehee Abschnitt „Diagnoseinformationen in Routinen“) enthalten.

Eine UDF darf keine dynamisch formulierten SQL-Anweisungen oder Cursorbeschreibungen enthalten, siehe Abschnitt „Dynamische SQL“.

Der aktuelle Berechtigungsschlüssel erhält automatisch das EXECUTE-Privileg für die erzeugte UDF. Hat er für die betreffenden Privilegien sogar die Berechtigung, diese weitergeben zu dürfen, dann darf er auch das EXECUTE-Privileg an andere Berechtigungsschlüssel weitergegeben.

Eine SQL-Anweisung in einer UDF darf auf die Parameter der UDF und (wenn die Anweisung Teil einer COMPOUND-Anweisung ist) auf lokale Variablen, jedoch nicht auf Benutzervariablen zugreifen.

Kommentare

Beschreibende Kommentare (siehe "Kommentar") können beliebig in eine UDF eingefügt werden.

7.2.2 Ausführen einer UDF

Eine UDF wird durch ihren Funktionsaufruf in einem Ausdruck aufgerufen, siehe "User Defined Functions (UDF)".

Wenn eine UDF Eingabeparameter erwartet, dann müssen die entsprechenden Werte (die Argumente) im Funktionsaufruf an die UDF übergeben werden.

Der (einzige) Rückgabewert einer UDF wird durch die RETURN-Anweisung bestimmt, siehe "RETURN - Rückgabewert einer User Defined Function (UDF) liefern".

Zur Ausführung einer UDF wird das EXECUTE-Privileg für die auszuführende UDF benötigt, nicht aber diejenigen Privilegien, die benötigt werden, um die in der UDF enthaltenen DML-Anweisungen ausführen zu können. Zusätzlich werden die SELECT-Privilegien für die Tabellen benötigt, die in den Aufrufparametern der Routine über Unterabfragen angesprochen werden.

Bei Auswertung eines Ausdrucks wird eine darin enthaltene Funktion ausgeführt und durch den berechneten Rückgabewert ersetzt.

UDFs können in Views aufgerufen werden.

7.2.3 Löschen einer UDF

Eine UDF wird mit der SQL-Anweisung DROP FUNCTION gelöscht, siehe "DROP FUNCTION - User Defined Function (UDF) löschen".

7.2.4 Unkorrelierte Funktionsaufrufe

Funktionsaufrufe einer UDF mit konstanten Eingabewerten werden als **unkorrelierte Funktionsaufrufe** bezeichnet. Konstante Eingabewerte beziehen sich nicht auf die SQL-Anweisung, die den Funktionsaufruf enthält,

Unkorrelierte Funktionsaufrufe werden bei der Ausführung der Anweisung von SESAM/SQL so behandelt:

- Für die Auswertung von Bedingungen werden Funktionswerte von unkorrelierten Funktionsaufrufen nur einmal berechnet.
- Für folgende Ausgabewerte dagegen werden Sie jedesmal neu berechnet:
  - in SELECT-Listen
  - für ORDER BY-Werte
  - für Werte in INSERT-Zeilen
  - für UPDATE... SET ...-Werte
  - für die INSERT- / UPDATE-Werte innerhalb einer MERGE-Anweisung

Beispiel

```
SELECT f(1,2) FROM t WHERE col < g(5+4,8,9)
```

Die Funktionsaufrufe `f(1,2)` und `g(5+4,8,9)` dieser SQL-Anweisung sind unkorreliert.

Es wird, um die Bedingung für die Sätze von `t` auszuwerten, die Funktion `g` nur einmal berechnet. Mit diesem konstanten Ergebnis wird dann die Treffermenge der Abfrage bestimmt. So können in der Bedingungsauswertung auch Indizes genutzt werden.

In der SELECT-Liste dagegen wird fuer jeden Treffersatz der Funktionswert der Funktionn `f` neu berechnet.

Annotationen VOLATILE / IMMUTABLE

Die Annotationen `/*% VOLATILE %*/` und `/*% IMMUTABLE %*/` steuern die Ausführung von unkorrelierten Funktionsaufrufen. Sie werden nur in einem Funktionsaufruf zwischen dem Namen der Funktion und der öffnenden Klammer für die Funktionsparameter akzeptiert. An anderen Stellen führen diese Annotationen zu einem Syntaxfehler für die Anweisung..

Bei Angabe von `/*% VOLATILE %*/` wird der Funktionswert stets neu berechnet.

Bei Angabe von `/*% IMMUTABLE %*/` in einem unkorrelierten Funktionsaufruf wird der Funktionswert **nicht** nochmals neu berechnet. Es wird der zuvor berechnete Funktionswert verwendet. Beim ersten Funktionsaufruf wird der Funktionswert neu berechnet.

Ohne Angabe einer dieser Annotationen wird die bisherige, zuvor beschriebene Vorgehensweise von SESAM/SQL angewendet.

Beispiel

```
SELECT f /*% VOLATILE %*/ (1,2)
FROM t WHERE col < g /*% IMMUTABLE %*/ (5+4,8,9)
```

Diese Funktionsaufrufe bilden die bisherige Vorgehensweise von SESAM/SQL mit Annotationen nach.

```
SELECT f /*% IMMUTABLE %*/ (1,2)
FROM t WHERE col < g /*% VOLATILE %*/ (5+4,8,9)
```

Durch die Angabe der Annotationen wird die Funktion `g` stets neu berechnet. Die Funktion `f` wird nur einmal berechnet.

7.2.5 Beispiele für UDFs

Beispiel 1: Jahreszahl ermitteln

Die folgende UDF `GetCurrentYear` liefert das aktuelle Jahr als Zahl. Sie enthält keine SQL-Anweisungen zum Lesen oder Ändern von Daten.

```
CREATE FUNCTION GetCurrentYear (IN time TIMESTAMP(3))
   RETURNS DECIMAL(4)
   CONTAINS SQL
   RETURN EXTRACT (YEAR FROM time)
```

Die UDF `GetCurrentYear` im Schema `FktSchema` wird verwendet:

- Alle Aufträge des Jahres 2014 ermitteln:

  ```
  DECLARE cursor_1 CURSOR FOR
  SELECT Auftragsnummer, Kundenname FROM Auftraege
  WHERE FktSchema.GetCurrentYear(Auftragstermin) = 2014
  ```

- Auslaufjahr auf das übernächste Jahr setzen (Schema `FktSchema` ist voreingestellt):

  ```
  UPDATE Modell.Exemplar
  SET Auslaufjahr = GetCurrentYear(CURRENT_TIMESTAMP(3)) + 2
  ```

Beispiel 2: Preis eines Artikels ermitteln

```
CREATE FUNCTION ARTIKEL_PREIS (IN P_ARTNR INTEGER)
   RETURNS NUMERIC(8,2)
   READS SQL DATA
   BEGIN
      RETURN (SELECT PREIS FROM TEILE.ARTIKEL WHERE ARTNR = P_ARTNR);
   END
```

Beispiel 3: Anonymisieren einer Kreditkarten-Nummer

Die folgende UDF `mask_credit_card_number` anonymisiert eine Kreditkarten-Nummer durch Maskieren der letzten 4 Stellen:

```
CREATE FUNCTION mask_credit_card_number(IN card_no CHAR(16))
   RETURNS CHAR(16)
   CONTAINS SQL
   RETURN SUBSTRING(card_no FROM 1 FOR 12) || '****'
```

Eine Mitteilung könnte damit so gestaltet werden:

```
Select surname, first_name, mask_credit_card_number(credit_card_number)
from ...
```

7.3 EXECUTE-Privileg für Routinen

SESAM/SQL stellt das EXECUTE-Privileg für Routinen bereit. Es wird mit der SQL-Anweisung GRANT vergeben und mit der SQL-Anweisung REVOKE entzogen.

Beim Erzeugen einer Routine muss der aktuelle Berechtigungsschlüssel das EXECUTE-Privileg für die in der Routine direkt aufgerufenen Routinen besitzen. Zusätzlich muss er für alle Tabellen und Spalten, die in der Routine angesprochen werden, diejenigen Privilegien besitzen, die benötigt werden, um die in der Routine enthaltenen DML-Anweisungen ausführen zu können.

Beim Erzeugen eines Views muss der aktuelle Berechtigungsschlüssel das EXECUTE-Privileg für die in dem View direkt aufgerufenen UDFs besitzen.

Zur Ausführung einer Routine (mit der SQL-Anweisung CALL oder durch Funktionsaufruf) wird das EXECUTE-Privileg für die auszuführende Routine benötigt, nicht aber diejenigen Privilegien, die benötigt werden, um die in der Routine enthaltenen DML-Anweisungen ausführen zu können. Zusätzlich werden die SELECT-Privilegien für die Tabellen benötigt, die in den Aufrufparametern der Routine über Unterabfragen angesprochen werden.

7.4 Informationen über Routinen

Informationen über Routinen werden in den Informationsschemata bereitgestellt, siehe Kapitel „Informationsschemata“.

Information Schema	View	Information über
INFORMATION_SCHEMA	PARAMETERS	Parameter von Routinen
INFORMATION_SCHEMA	ROUTINES	Routinen
INFORMATION_SCHEMA	ROUTINE_PRIVILEGES	Privilegien für Routinen
INFORMATION_SCHEMA	ROUTINE_TABLE_USAGE	Tabellen in Routinen
INFORMATION_SCHEMA	ROUTINE_COLUMN_USAGE	Spalten in Routinen
INFORMATION_SCHEMA	ROUTINE_ROUTINE_USAGE	Routinen in anderen Routinen
INFORMATION_SCHEMA	VIEW_ROUTINE_USAGE	Routinen in Views
SYS_INFO_SCHEMA	SYS_PARAMETERS	Parameter von Routinen
SYS_INFO_SCHEMA	SYS_ROUTINES	Routinen
SYS_INFO_SCHEMA	SYS_ROUTINE_PRIVILEGES	Privilegien für Routinen
SYS_INFO_SCHEMA	SYS_ROUTINE_USAGE	Tabellen und Spalten in Routinen
SYS_INFO_SCHEMA	SYS_ROUTINE_ERRORS	Fehler-Ereignisse in Routinen
SYS_INFO_SCHEMA	SYS_ROUTINE_ROUTINE_USAGE	Routinen in anderen Routinen
SYS_INFO_SCHEMA	SYS_VIEW_ROUTINE_USAGE	Routinen in Views

Tabelle 27: Routinen in den Informationsschemata

7.5 Pragmas in Routinen

Folgende Pragmas gibt es speziell für Routinen:

- DEBUG ROUTINE zur Ausgabe von zusätzlichen Hinweisen oder Fehlerinformationen
- DEBUG VALUE zur Ausgabe zusätzlicher Informationen für die SQL-Anweisungen SET in Routinen und RETURN in UDFs
- LOOP LIMIT zur Begrenzung der Anzahl Schleifendurchläufe

Siehe Abschnitt „Pragmas und Annotationen“.

Die Pragmas DEBUG ROUTINE und LOOP LIMIT wirken vor der SQL-Anweisung CALL und vor den DML-Anweisungen DECLARE CURSOR, DELETE, INSERT, MERGE, SELECT und UPDATE. Bei Angabe **vor** DML-Anweisungen wirken diese Pragmas auf alle UDFs und die darin enthaltenen Routinen der DML-Anweisung. Diese Pragmas haben vor SQL-Anweisungen **in** einer Routine keine Wirkung.

Daneben können auch andere Pragmas in der CALL-Anweisung und in Routinen verwendet werden.

Pragmas EXPLAIN, CHECK, LIMIT ABORT_EXECUTION

Diese Pragmas wirken vor der SQL-Anweisung CALL und vor den DML-Anweisungen DECLARE CURSOR, DELETE, INSERT, MERGE, SELECT und UPDATE. Bei Angabe vor DML-Anweisungen wirken sie auf alle UDFs und die darin enthaltenen Routinen der DML-Anweisung. Wenn eines dieser Pragmas vor einer SQL-Anweisung in einer Routine steht, dann wird es ignoriert.

Pragmas ISOLATION LEVEL, LOCK MODE

Wenn diese Pragmas vor einer CALL-Anweisung stehen, dann wirken sie sich nur auf die möglicherweise komplexen Aufrufwerte der CALL-Anweisung aus.

Diese Pragmas können auch vor SQL-Anweisungen in Routinen stehen. Sie haben dann bei DECLARE CURSOR, DELETE, INSERT, MERGE, SELECT und UPDATE die beschriebene Wirkung.

Wenn diese Pragmas vor einer IF-Anweisung stehen, dann wirken sie nur auf die Bedingungen der IF-Anweisung. Vor den in der IF-Anweisung enthaltenen Anweisungen können diese Pragmas ebenfalls angegeben werden.

Bei der SET-Anweisung wirken sich diese Pragmas auf die Auswertung des Ausdrucks auf der rechten Seite der Zuweisung aus.

Wenn diese Pragmas vor einer LOOP-, LEAVE- oder ITERATE-Anweisung stehen, dann werden sie ignoriert.

Wenn diese Pragmas vor einer FOR-Anweisung stehen, dann wirken sie nur auf die Cursordefinition der FOR Anweisung. Vor den in der FOR-Anweisung enthaltenen SQL-Anweisungen können diese Pragmas ebenfalls angegeben werden.

Wenn diese Pragmas vor einer WHILE-Anweisung stehen, dann wirken sie nur auf die Bedingung der WHILE-Schleife. Vor den in der WHILE-Anweisung enthaltenen SQL-Anweisungen können diese Pragmas ebenfalls angegeben werden.

Wenn diese Pragmas die UNTIL-Bedingung einer REPEAT-Anweisung beeinflussen sollen, dann müssen sie direkt vor UNTIL angegeben werden (nicht vor REPEAT). Vor den in der REPEAT-Anweisung enthaltenen SQL-Anweisungen können diese Pragmas ebenfalls angegeben werden.

Wenn diese Pragmas vor einer CASE-Anweisung stehen, dann wirken sie nur auf die Ausdrücke außerhalb der THEN- und ELSE-Anweisungsblöcke. Vor den in der CASE-Anweisung enthaltenen SQL-Anweisungen können diese Pragmas ebenfalls angegeben werden.

Bei der RETURN Anweisung wirken sich diese Pragmas auf die Auswertung des RETURN-Wertes aus.

Bei allen anderen Anweisungen innerhalb von Routinen haben diese Pragmas keine Wirkung.

Pragmas IGNORE, JOIN, KEEP JOIN ORDER, OPTIMIZATION, SIMPLIFICATION, USE

Wenn eines dieser Optimierungs-Pragmas vor einer CALL-Anweisung steht, dann wirkt es sich nur auf die Optimierung der möglicherweise komplexen Aufrufwerte der CALL-Anweisung aus.

Diese Pragmas können auch vor SQL-Anweisungen einer Routine stehen. Sie bewirken dann bei DECLARE CURSOR, DELETE, INSERT, MERGE, SELECT und UPDATE die beschriebene Optimierung.

Wenn diese Pragmas vor einer IF-Anweisung stehen, dann wirken sie nur bei der Optimierung der Bedingungen der IF-Anweisung. Vor den in der IF-Anweisung enthaltenen Anweisungen können diese Pragmas ebenfalls angegeben werden.

Diese Pragmas wirken sich bei der SET-Anweisung auf die Optimierung des Ausdrucks auf der rechten Seite der Zuweisung aus.

Wenn diese Pragmas vor einer LOOP-, LEAVE- oder ITERATE-Anweisung stehen, dann werden sie ignoriert.

Wenn diese Pragmas vor einer FOR-Anweisung stehen, dann wirken sie nur auf die Cursordefinition der FOR Anweisung. Vor den in der FOR-Anweisung enthaltenen SQL-Anweisungen können diese Pragmas ebenfalls angegeben werden.

Wenn diese Pragmas vor einer WHILE-Anweisung stehen, dann wirken sie nur auf die Bedingung der WHILE-Schleife. Vor den in der WHILE-Anweisung enthaltenen SQL-Anweisungen können diese Pragmas ebenfalls angegeben werden.

Wenn diese Pragmas die UNTIL-Bedingung einer REPEAT-Anweisung beeinflussen sollen, dann müssen sie direkt vor UNTIL angegeben werden (nicht vor REPEAT). Vor den in der REPEAT-Anweisung enthaltenen SQL-Anweisungen können diese Pragmas ebenfalls angegeben werden.

Wenn diese Pragmas vor einer CASE-Anweisung stehen, dann wirken sie nur auf die Ausdrücke außerhalb der THEN- und ELSE-Anweisungsblöcke. Vor den in der CASE-Anweisung enthaltenen SQL-Anweisungen können diese Pragmas ebenfalls angegeben werden.

Bei der RETURN Anweisung wirken sich diese Pragmas auf die Auswertung des RETURN-Wertes aus.

Bei allen anderen Anweisungen innerhalb von Routinen haben diese Pragmas keine Wirkung.

Pragma DATA TYPE, PREFETCH, UTILITY MODE

Diese Pragmas werden sowohl vor einer CALL-Anweisung als auch vor einer SQL-Anweisung einer Routine ignoriert.

7.6 Kontrollanweisungen in Routinen

Kontrollanweisungen dürfen nur in Routinen angegeben werden. Sie steuern den Ablauf einer Routine, z.B. durch Laufschleifen oder Bedingungen. Sie können umfangreich werden und ihrerseits wieder Folgen von SQL-Anweisungen enthalten.

SQL-Anweisung	Funktion	siehe
COMPOUND	SQL-Anweisungen in einem gemeinsamen Kontext ausführen	"COMPOUND - SQL-Anweisungen in gemeinsamem Kontext ausführen"
CALL	Prozedur aufrufen	"CALL - Prozedur ausführen"
CASE	SQL-Anweisungen bedingt ausführen	"CASE - SQL-Anweisungen bedingt ausführen"
FOR	SQL-Anweisungen in einer Schleife ausführen	"FOR - SQL-Anweisungen in einer Schleife ausführen"
IF	SQL-Anweisungen bedingt ausführen	"IF - SQL-Anweisungen bedingt ausführen"
ITERATE	zum nächsten Schleifendurchlauf wechseln	"ITERATE - zum nächsten Schleifendurchlauf wechseln"
LEAVE	Schleife oder COMPOUND-Anweisung beenden	"LEAVE - Schleife oder COMPOUND-Anweisung beenden"
LOOP	SQL-Anweisungen in einer Schleife ausführen	"LOOP - SQL-Anweisungen in einer Schleife ausführen"
REPEAT	SQL-Anweisungen in einer Schleife ausführen	"REPEAT - SQL-Anweisungen in einer Schleife ausführen"
RETURN [1]	Rückgabewert einer User Defined Function (UDF) liefern	"RETURN - Rückgabewert einer User Defined Function (UDF) liefern"
SET	Wert zuweisen	"SET - Wert zuweisen"
WHILE	SQL-Anweisungen in einer Schleife ausführen	"WHILE - SQL-Anweisungen in einer Schleife ausführen"

Tabelle 28: Kontroll- und Diagnoseanweisungen von Routinen

[1] nur für UDFs

In SESAM/SQL ab V9.0 sind geschachtelte Aufrufe von Routinen erlaubt. Die CALL-Anweisung gehört deshalb zu den in einer Routine erlaubten SQL-Anweisungen.

7.7 COMPOUND-Anweisung in Routinen

Die COMPOUND-Anweisung ist eine der Kontrollanweisungen in Routinen. Sie führt weitere SQL-Anweisungen in einem gemeinsamen Kontext aus. Für diese SQL-Anweisungen gelten gemeinsame lokale Daten, gemeinsame lokale Cursor und gemeinsame Fehler-Routinen.

Die detaillierte Beschreibung der COMPOUND-Anweisung finden Sie auf "COMPOUND - SQL-Anweisungen in gemeinsamem Kontext ausführen".

Lokale Daten

Lokale Daten sind Variablen oder Fehlernamen, die nur innerhalb der COMPOUND-Anweisung angesprochen werden können.
Die Namen der lokalen Daten müssen sich voneinander unterscheiden.

Für Variablen wird ein Datentyp und ggf. ein Standardwert definiert. Sie besitzen keine Indikatorvariable. Sie können in lokalen Cursor-Definitionen, lokalen Fehler-Routinen und den SQL-Anweisungen der COMPOUND-Anweisung verwendet werden.

Fehlernamen definieren zum leichteren Verständnis einen Namen für einen Fehler (ohne Angabe eines zugehörigen SQLSTATE) oder einen Namen für einen SQLSTATE. Sie können in lokalen Fehler-Routinen verwendet werden, siehe "COMPOUND - SQL-Anweisungen in gemeinsamem Kontext ausführen".

Lokale Cursor

Mit der Definition lokaler Cursor werden Cursor festgelegt, die nur innerhalb der COMPOUND-Anweisung angesprochen werden können.
Die Namen der lokalen Cursor müssen sich voneinander unterscheiden.

Lokale Cursor können in lokalen Fehler-Routinen und den SQL-Anweisungen der COMPOUND-Anweisung verwendet werden.

Lokale Cursor werden ohne die Klausel WITH HOLD definiert. Die SQL-Anweisungen STORE und RESTORE dürfen nicht auf lokale Cursor angewendet werden.

Gemeinsame Fehler-Routinen

Mit der Definition von Fehler-Routinen wird festgelegt, wie reagiert werden soll, wenn bei der Verarbeitung einer SQL-Anweisung im Rahmen der COMPOUND-Anweisung ein SQLSTATE '00000' gemeldet wird.

Wenn ein SQLSTATE auftritt, der in einer Fehler-Routine spezifiziert wurde, dann wird die Fehler-Routine für den SQLSTATE ausgeführt. Für sonstige SQLSTATEs führt SESAM/SQL automatisch eine Fehlerbehandlung durch.

In den Fehler-Routinen wird die Art der Fehlerbehandlung in Abhängigkeit vom SQLSTATE definiert. Dort entscheiden im Fehlerfall weitere SQL-Anweisungen, ob die Routine fortgesetzt oder beendet werden soll. Änderungen, die im Rahmen der COMPOUND-Anweisung durchgeführt wurden, können rückgängig gemacht werden.

7.8 Diagnoseinformationen in Routinen

SESAM/SQL stellt Diagnoseinformationen in Routinen bereit. Die SQL-Norm verwendet dafür den Begriff „Diagnostics Management“.

Diagnoseinformationen werden für eine zuvor ausgeführte SQL-Anweisung in einem Diagnosebereich bereitgestellt. Bei Routinen in SESAM/SQL kann es zu einem Zeitpunkt mehrere Diagnosebereiche geben (für eine SQL-Anweisung, bei Aufruf einer (Fehler-)Routine), speziell bei geschachtelten Routinen.

> ¡ An der ESQL-Cobol-Schnittstelle, also im Anwenderprogramm, hat der Diagnosebereich den Namen „SQLda“.

Mit folgenden SQL-Anweisungen, die nur in Routinen verwendet werden dürfen, kann lesend und/oder schreibend auf einen Diagnosebereich zugegriffen werden:

SQL-Anweisung	**Funktion**	**siehe**
GET DIAGNOSTICS	Diagnoseinformationen zu einer Anweisung ausgeben	"GET DIAGNOSTICS - Diagnoseinformationen ausgeben"
SIGNAL	Fehler in Routine melden	"SIGNAL - Fehler in Routine melden"
RESIGNAL	Fehler in lokaler Fehler-Routine melden	"RESIGNAL - Fehler in lokaler Fehler-Routine melden"

Tabelle 29: Kontroll- und Diagnoseanweisungen von Routinen

Durch diese Diagnoseanweisungen und die nachfolgend beschriebenen selbst definierten SQLSTATEs können Sie die Programmierung von Routinen verbessern. Sie können auftretende Fehler genauer analysieren und differenziert darauf reagieren.

Erfolg einer SQL-Anweisung in einer Routine

Der Erfolg einer SQL-Anweisung in einer Routine wird zur einfacheren Beschreibung in diesem Handbuch so definiert:

- Die SQL-Anweisung war **erfolgreich**, wenn sie mit SQLSTATE '00000' beendet wurde.
- Die SQL-Anweisung war **fehlerfrei**, wenn sie mit SQLSTATE '00000', einem SQLSTATE der Klassen '01xxx' (Warnung) oder '02xxx' (keine Daten) beendet wurde.
- Die SQL-Anweisung in einer Routine war **fehlerhaft**, wenn sie nicht fehlerfrei beendet wurde.

> ¡ Eine Routine wird nach einer fehlerfreien SQL-Anweisung fortgesetzt, wenn keine Fehler-Routinen für die SQLSTATEs der Klassen '01000' und '02000' definiert sind. Wenn z.B. bei einer SQL-Anweisung in einer Prozedur eine Warnung auftritt, dann wird die entsprechende CALL-Anweisung mit SQLSTATE '00000' beendet.

Selbst definierte SQLSTATEs

Mit SESAM/SQL ab V9.0 können Sie SQLSTATEs selbst definieren. Für sie ist die Klasse '46Sxx' (x ist eine Zahl oder ein Großbuchstabe) reserviert. In dieser Klasse können Sie so bis zu 1296 SQLSTATEs selbst definieren. Diese Klasse wird weder von der SQL-Norm noch von SESAM/SQL verwendet.

Selbst definierte SQLSTATEs können Sie in den Diagnoseanweisungen SIGNAL und RESIGNAL angeben.Mit einem selbst definierten SQLSTATE können Sie in der Diagnoseanweisung SIGNAL gezielt eine bestimmte Fehler-Routine aufrufen. In der Fehler-Routine können Sie mit der Diagnoseanweisung RESIGNAL die Routine gezielt abbrechen. In beide Anweisungen können Sie auch zusätzliche Diagnoseinformationen in den Diagnosebereich eintragen.

Für selbst definierte SQLSTATEs gibt es keine vorgefertigten SESAM-Meldungstexte. Wenn ein selbst definierter SQLSTATE als unbehandelter SQLSTATE in das Anwendungsprogramm gelangt, dann generiert SESAM/SQL daraus die Meldung `SEW46xx (&00)`. Als Insert `(&00)` erscheint dann der `MESSAGE_TEXT` aus dem Diagnosebereich. Sie können damit in den Diagnoseanweisungen SIGNAL und RESIGNAL indirekt einen eigenen Meldungstext (ohne begleitenden Hilfetext) erzeugen.

SQLSTATE '45000' (unbehandelter SQLSTATE)

Mit SESAM/SQL können Sie in einer COMPOUND-Anweisung einen lokalen Fehlernamen für einen SQLSTATE definieren, siehe Abschnitt „Lokale Daten“.

Sie können aber auch einen Fehlernamen ohne Verknüpfung mit einem SQLSTATE definieren.

Mit diesem Fehlernamen können Sie in der Diagnoseanweisung SIGNAL eine bestimmte Fehler-Routine aufrufen. Wenn diese Fehler-Routine nicht existiert oder mit RESIGNAL (ohne Angabe eines SQLSTATE) verlassen wird, dann dann wird die Routine mit dem SQLSTATE '45000' beendet.

SESAM/SQL erzeugt daraus folgende Meldung:

```
SEW4500 UNHANDLED USER DEFINED EXCEPTION (&00). (&01)
```

Das Insert `(&00)` enthält den Fehlernamen. Wenn bei SIGNAL bzw. RESIGNAL ein `MESSAGE_TEXT` angegeben wurde, dann erscheint er als Insert `(&01)`.

Bei geeigneter Wahl des Fehlernamens und ggf. eines entsprechenden `MESSAGE_TEXT` erhält der Anwender somit eine aussagekräftige Meldung.

GET DIAGNOSTICS

GET DIAGNOSTICS ermittelt Informationen zu einer zuvor in einer Routine ausgeführten SQL-Anweisung und trägt diese in einen Prozedurparameter (Ausgabe) oder eine lokale Variable ein. Die Informationen beziehen sich auf die Anweisung selbst oder auf die davon betroffenen Objekte der Datenbank.

GET DIAGNOSTICS ändert weder Inhalt noch Reihenfolge von Diagnosebereichen. D.h. unmittelbar aufeinander folgende GET DIAGNOSTICS-Anweisungen werten dieselbe Diagnoseinformation aus.

Die detaillierte Beschreibung der GET DIAGNOSTICS-Anweisung finden Sie auf "GET DIAGNOSTICS - Diagnoseinformationen ausgeben".

SIGNAL

SIGNAL meldet in einer Routine einen Fehler oder einen selbst definierten SQLSTATE.

Die detaillierte Beschreibung der SIGNAL-Anweisung finden Sie auf "SIGNAL - Fehler in Routine melden".

SIGNAL löscht den aktuellen Diagnosebereich und trägt optional folgende Diagnoseinformationen in den aktuellen Diagnosebereich ein:

- Bei Angabe eines Fehlernamens wird dieser als CONDITION_IDENTIFIER eingetragen. Sonst wird eine Zeichenkette der Länge 0 zugewiesen.
- Der RETURNED_SQLSTATE wird versorgt:
  - Bei Angabe eines SQLSTATE wird dieser als RETURNED_SQLSTATE eingetragenn
  - Wenn für den angegebenen Fehlernamen ein SQLSTATE definiert ist, dann wird für RETURNED_SQLSTATE der definierte SQLSTATE eingetragen
  - Sonst wird der SQLSTATE '45000' eingetragen
- Bei Angabe von MESSAGE_TEXT werden MESSAGE_TEXT, MESSAGE_LENGTH und MESSAGE_OCTET_LENGTH entsprechend versorgt. Sonst wird MESSAGE_TEXT eine Zeichenkette der Länge 0 zugewiesen.

Die Routine wird mit einer Fehler-Routine fortgesetzt oder beendet:

- Wenn RETURNED_SQLSTATE '45000' und für den RETURNED_SQLSTATE eine lokale Fehler-Routine definiert ist, dann wird diese Fehler-Routine ausgeführt.
- Wenn RETURNED_SQLSTATE = '45000' und für den in CONDITION_IDENTIFIER eingetragenen Fehlernamen eine lokale Fehler-Routine definiert ist, dann wird diese Fehler-Routine ausgeführt.
- Sonst liegt ein unbehandelter SQLSTATE vor. Die Routine wird mit dem in RETURNE-D_SQLSTATE eingetragenen SQLSTATE beendet.

Weitere Hinweise:

- Mit SIGNAL kann die Ausführung einer bestimmten Fehler-Routine erreicht werden.
- Eine unmittelbar hinter der SIGNAL-Anweisung stehende SQL-Anweisung wird nur dann durchlaufen, wenn die durch SIGNAL aufgerufene Fehler-Routine mit CONTINUE definiert ist und fehlerfrei beendet wurde.
- Wenn die bei SIGNAL in den Diagnosebereich eingetragenen Werte (wie z.B. MESSAGE_TEXT) gelesen werden sollen, dann muss GET CURRENT DIAGNOSTICS entweder unmittelbar nach SIGNAL stehen (siehe vorherigen Hinweis) oder es muss in der aufgerufenen Fehler-Routine GET STACKED DIAGNOSTICS verwendet werden. Diese Fehler-Routine muss nicht notwendigerweise Teil der aktuellen COMPOUND-Anweisung sein. Sie kann auch eine Fehler-Routine einer übergeordneten Routine sein, die die Routine mit der SIGNAL-Anweisung verwendet hat. Im letzteren Fall wird dann also der Diagnosebereich der aufrufenden Anweisung ausgewertet.
- Eine Routine wird nach einer fehlerfreien,aber nicht erfolgreichen SQL-Anweisung fortgesetzt. Selbst wenn in einem solchen Fall eine Fehler-Routine mit EXIT oder UNDO durchlaufen wurde, beendet sich die Routine mit SQLSTATE '00000', es sei denn in der Fehler-Routine selbst wurde eine SQL-Anweisung fehlerhaft beendet. Mit der SIGNAL-Anweisung kann in einem derartigen Fall die Routine mit einem selbst definierten SQLSTATE beendet werden.

RESIGNAL

RESIGNAL meldet in einer lokalen Fehler-Routine eine Bedingung oder einen SQLSTATE. Im Unterschied zu SIGNAL ist die Angabe eines Fehlernamens oder eines SQLSTATE optional.

Die detaillierte Beschreibung der RESIGNAL-Anweisung finden Sie auf "RESIGNAL - Fehler in lokaler Fehler-Routine melden".

RESIGNAL verwendet den Diagnosebereich der SQL-Anweisung, die die Fehler-Routine aktiviert hat, als aktuellen Diagnosebereich und ändert ggf. folgende Diagnoseinformationen::

- Wenn weder Fehlername noch SQLSTATE angegeben werden, dann bleiben CONDITION_IDENTIFIER und RETURNED_SQLSTATE unverändert. Es gilt:
  - RETURNED_SQLSTATE darf keinen SQLSTATE der Klassen '01xxx' oder '02xxx' enthalten. Sonst wird RESIGNAL mit Fehler beendet.
  - Wenn MESSAGE_TEXT= angegeben ist, dann muss RETURNED_SQLSTATE entweder einen selbst definierten SQLSTATE oder den Wert '45000' enthalten. Sonst wird RESIGNAL mit Fehler beendet.
- Der aktuelle Diagnosebereich wird ggf. modifiziert:
  - Bei Angabe eines Fehlernamens wird dieser als CONDITION_IDENTIFIER eingetragen. Sonst wird eine Zeichenkette der Länge 0 zugewiesen.
  - Bei Angabe eines SQLSTATE wird dieser als RETURNED_SQLSTATE eingetragen..
  - Wenn für den angegebenen Fehlernamen ein SQLSTATE definiert ist, dann wird für RETURNED_SQLSTATE der definierte SQLSTATE eingetragen. Sonst wird der SQLSTATE '45000' eingetragen.
- Bei Angabe von MESSAGE_TEXT werden MESSAGE_TEXT, MESSAGE_LENGTH und MESSAGE_OCTET_LENGTH entsprechend versorgt. Sonst wird MESSAGE_TEXT eine Zeichenkette der Länge 0 zugewiesen.

Die Routine, in der die lokale Fehler-Routine mit der RESIGNAL-Anweisung durchlaufen wurde, wird mit dem in RETURNED_SQLSTATE eingetragenen SQLSTATE beendet.

Weitere Hinweise:

- Eine Routine wird auch nach Durchlaufen einer Fehler-Routine, die mit EXIT oder UNDO definiert ist, mit SQLSTATE '00000' beendet, es sei denn in der Fehler-Routine selbst wurde eine SQL-Anweisung fehlerhaft beendet. Mit RESIGNAL können Sie den SQLSTATE, der die Fehler-Routine ausgelöst hat, zurückmelden.
- Eine SIGNAL-Anweisung, die in einer Fehler-Routine aufgerufen wird, hat den gleichen Effekt wie eine RESIGNAL-Anweisung mit explizit angegebenem Fehlernamen oder SQLSTATE.

Beispiele für den Einsatz der Diagnoseanweisungen

Unterschiedliche Situationen bei der Abfrage des SQLSTATEs

```
CREATE PROCEDURE proc1() MODIFIES SQL DATA
   BEGIN ATOMIC
      DECLARE state1, state2, state3 CHAR(5);
      DECLARE EXIT HANDLER FOR SQLEXCEPTION
         BEGIN
            DELETE FROM tab1; ------------------------------------------ (3)
            GET STACKED DIAGNOSTICS CONDITION state2 = RETURNED_SQLSTATE;
            GET CURRENT DIAGNOSTICS CONDITION state3 = RETURNED_SQLSTATE;
            ... ------------------------------------------------------- (2)
         END;
      ...
      UPDATE tab2 SET ...;
      GET CURRENT DIAGNOSTICS CONDITION state1 = RETURNED_SQLSTATE; ----- (1)
      ...
   END
```

(1) Die lokale Variable `state1` wird nur dann versorgt, wenn die UPDATE-Anweisung erfolgreich oder fehlerfrei durchlaufen wurde. Sie enthält dann entweder den SQLSTATE '00000', eine Warnung oder den SQLSTATE '02000' (keine Daten). Die Fehler-Routine wird nicht durchlaufen.

(2) Wenn die UPDATE-Anweisung mit Fehler und die DELETE-Anweisung ohne Fehler durchlaufen wurde, dann enthält `state2` den SQLSTATE der UPDATE-Anweisung, die den Fehler verursacht hat.
`state3` enthält den SQLSTATE der DELETE-Anweisung ('00000', eine Warnung oder '02000' (keine Daten)).
`state1` wird nicht versorgt, da die Prozedur wegen der Fehler-Routine (EXIT) abgebrochen wurde.

(3) Wenn auch die DELETE-Anweisung der Fehler-Routine mit Fehler durchlaufen wurde, dann wird die Prozedur wegen des unbehandelten SQLSTATEs sofort abgebrochen. Es wird keine der GET DIAGNOSTICS-Anweisungen durchlaufen.

Wenn die Fehler-Routine (statt mit EXIT) mit CONTINUE definiert ist und ohne Fehler durchlaufen wird, dann wird `state1` auch nach einer UPDATE-Anweisung, die mit Fehler durchlaufen wurde, versorgt. `state1` erhält dann den SQLSTATE der UPDATE-Anweisung, die den Fehler verursacht hat.

Spezielle Behandlung des SQLSTATE '02000'

Nach SQLSTATE '02000' (keine Daten) wird eine Routine normalerweise fortgesetzt. Im folgenden Beispiel wird das in einem Fall akzeptiert, in einem anderen Fall soll es zu einem Fehler führen.

```
CREATE PROCEDURE proc2(OUT par1 INTEGER, OUT par2 INTEGER) MODIFIES SQL DATA
   BEGIN ATOMIC
      DELETE FROM tab1;
      GET DIAGNOSTICS par1 = ROW_COUNT;
      DELETE FROM tab2;
      GET DIAGNOSTICS par2 = ROW_COUNT;
      IF par2 = 0
         THEN SIGNAL SQLSTATE '46SA1'
              SET MESSAGE_TEXT = 'tab2 must contain at least one record';
      END IF;
   END
```

Wenn die DELETE-Anweisungen ohne Fehler durchlaufen wurden, dann wird die jeweilige Anzahl der gelöschten Sätze in den beiden Ausgabeparametern eingetragen. Bei Tabelle `tab1` darf die Anzahl auch 0 sein. Wenn aber Tabelle `tab2` leer ist, dann wird die Prozedur abgebrochen. Wegen der Klausel ATOMIC werden auch die Löschungen in der Tabelle `tab1` rückgängig gemacht. SESAM/SQL generiert die Meldung:

```
SEW46A1 TAB2 MUST CONTAIN AT LEAST ONE RECORD
```

Merken des aufgetretenen SQLSTATE

Nach einem unbehandelten SQLSTATE wird eine Prozedur abgebrochen und genau dieser SQLSTATE gemeldet. Wenn Sie dieses Ereignis zusätzlich in einer Tabelle protokollieren möchten, dann definieren Sie z.B. nachfolgende Fehler-Routine. Mit der RESIGNAL-Anweisung wird der aufgetretene SQLSTATE zurückgemeldet. Ohne die RESIGNAL-Anweisung würde sich die Prozedur mit SQLSTATE '00000' beenden.

```
CREATE PROCEDURE proc3() MODIFIES SQL DATA
   BEGIN ATOMIC
```

```
      DECLARE error CHAR(5);
      DECLARE UNDO HANDLER FOR SQLEXCEPTION
      BEGIN
         GET DIAGNOSTICS CONDITION error = RETURNED_SQLSTATE;
         INSERT INTO logging_tab
           VALUES (CURRENT_TIMESTAMP(3),'SQLSTATE ' || error || ' occurred');
         RESIGNAL;
      END;
   -- procedure body
   ...
   END
```

Suche nach leeren Tabellen

Die Anzahl leerer Tabellen soll über eine User Defined Function ermittelt werden. Wenn die Anzahl leerer Tabellen die eingegebene Anzahl übersteigt, dann soll die Suche mit Fehler abgebrochen werden.

```
CREATE FUNCTION check_tables(IN max_nbr INTEGER)
                RETURNS INTEGER READS SQL DATA
   BEGIN
      DECLARE "TABLE ERROR" CONDITION;
      DECLARE nbr_empty_tables integer DEFAULT 0;
      DECLARE CONTINUE HANDLER FOR "TABLE ERROR"
         BEGIN
            nbr_empty_tables = nbr_empty_tables + 1;
            IF nbr_empty_tables > max_nbr
               THEN RESIGNAL SET MESSAGE_TEXT = 'TOO MANY EMPTY TABLES';
            END IF;
         END;
   IF (SELECT COUNT(*) FROM tab1) = 0 THEN SIGNAL "TABLE ERROR";
   END IF;
   IF (SELECT COUNT(*) FROM tab2) = 0 THEN SIGNAL "TABLE ERROR";
   END IF;
   IF (SELECT COUNT(*) FROM tab3) = 0 THEN SIGNAL "TABLE ERROR";
   END IF;
   RETURN nbr_empty_tables;
   END
SELECT check_tables(2) INTO :NBR-EMPTY-TABLES FROM TABLE(DEE)
```

Wenn die Anzahl leerer Tabellen die eingegebene Anzahl nicht übersteigt, dann wird die Anzahl leerer Tabellen in der Benutzervariablen `:NBR-EMPTY-TABLES` abgelegt.

Wenn aber mehr als zwei leere Tabellen existieren, wird die Suche mit SQLSTATE '45000' beendet. SESAM/SQL generiert dann folgende Meldung:

```
SEW4500 UNHANDLED USER DEFINED EXCEPTION (TABLE ERROR). TOO MANY EMPTY TABLES
```

8 SQL-Anweisungen

Dieses Kapitel beschreibt die SQL-Anweisungen. Es gliedert sich in zwei Teile:

- Inhaltliche Zusammenstellung
- Alphabetischer Nachschlageteil

8.1 Inhaltliche Zusammenstellung

In diesem Abschnitt sind die SQL-Anweisungen nach inhaltlichen Gesichtspunkten zusammengestellt. Die Einteilung der Anweisungen orientiert sich an der SQL-Norm.

Die SESAM/SQL-spezifischen Anweisungen sind grau unterlegt.

8.1.1 SQL-Anweisungen zur Schemadefinition und -verwaltung

Schema

SQL-Anweisung	Funktion
CREATE SCHEMA	Schema erzeugen
DROP SCHEMA	Schema löschen

Tabelle 30: SQL-Anweisungen für Schemata

Basistabelle

SQL-Anweisung	Funktion
ALTER TABLE	Basistabelle ändern
CREATE TABLE	Basistabelle erzeugen
DROP TABLE	Basistabelle löschen

Tabelle 31: SQL-Anweisungen für Basistabellen

View

SQL-Anweisung	Funktion
CREATE VIEW	View erzeugen
DROP VIEW	View löschen

Tabelle 32: SQL-Anweisungen für Views

Privilegien

SQL-Anweisung	Funktion
GRANT	Privilegien vergeben
REVOKE	Privilegien entziehen

Tabelle 33: SQL-Anweisungen für Privilegien

Prozedur (Stored Procedure)

SQL-Anweisung	Funktion
CREATE PROCEDURE	Prozedur erzeugen
DROP PROCEDURE	Prozedur löschen

Tabelle 34: SQL-Anweisungen für Prozeduren

User Defined Function (UDF)

SQL-Anweisung	Funktion
CREATE FUNCTION	UDF erzeugen
DROP FUNCTION	UDF löschen

Tabelle 35: SQL-Anweisungen für User Defined Functions

8.1.2 SQL-Anweisungen zum Abfragen und Ändern von Daten

Ohne Cursor

SQL-Anweisung	Funktion
DELETE	Sätze löschen
INSERT	Sätze in Tabelle einfügen
MERGE	Sätze in Tabelle einfügen oder Spaltenwerte ändern
SELECT...INTO	Einzelnen Satz lesen (statische SELECT-Anweisung)
SELECT (ohne INTO)	Einzelnen Satz lesen (dynamische SELECT-Anweisung)
UPDATE	Spaltenwerte ändern

Tabelle 36: SQL-Anweisungen zum Abfragen und Ändern ohne Cursor

Mit Cursor

Die folgenden SQL-Anweisungen können mit einem statischen und mit einem dynamischen Cursor verwendet werden.

Bei den ausführbaren Anweisungen muss bei einem dynamischen Cursor die zugehörige Cursorbeschreibung zum Ausführungszeitpunkt der Anweisung vorbereitet sein.

Bei manchen Anweisungen gibt es Abweichungen oder Einschränkungen, wenn ein dynamischer Cursor verwendet wird. Diese sind in der Tabelle genannt.

SQL-Anweisung	Funktion
CLOSE	Cursor schließen
DECLARE...CURSOR	Cursor vereinbaren (nicht ausführbar) dynamischer Cursor: statt der Cursorbeschreibung wird der Anweisungsbezeichner für die Cursorbeschreibung angegeben
DELETE...CURRENT	Aktuellen Satz löschen
FETCH	Cursor positionieren und Spaltenwerte lesen
OPEN	Cursor öffnen dynamischer Cursor: zusätzlich USING-Klausel
RESTORE	Cursor wiederherstellen
STORE	Cursorposition speichern
UPDATE...CURRENT	Aktuellen Satz ändern

Tabelle 37: SQL-Anweisungen zum Abfragen und Ändern mit Cursor

8.1.3 SQL-Anweisungen zur Transaktionsverwaltung

SQL-Anweisung	Funktion
SET TRANSACTION	Eigenschaften der SQL-Transaktion festlegen
COMMIT WORK	SQL-Transaktion beenden
ROLLBACK WORK	SQL-Transaktion zurücksetzen

Tabelle 38: SQL-Anweisungen zur Transaktionsverwaltung

8.1.4 SQL-Anweisungen zur Session-Steuerung

SQL-Anweisung	Funktion
SET CATALOG	Datenbanknamen voreinstellen
SET SCHEMA	Schemanamen voreinstellen
SET SESSION AUTHORIZATION	Berechtigungsschlüssel festlegen
PERMIT	Benutzeridentifikation für SESAM/SQL V1 angeben

Tabelle 39: SQL-Anweisungen zur Session-Steuerung

8.1.5 SQL-Anweisungen der dynamischen SQL

Dynamisch formulierte Anweisung

SQL-Anweisung	Funktion
EXECUTE	Vorbereitete Anweisung ausführen
EXECUTE IMMEDIATE	Dynamisch formulierte Anweisung ausführen
PREPARE	Dynamisch formulierte Anweisung vorbereiten

Tabelle 40: SQL-Anweisungen für dynamisch formulierte Anweisung

Deskriptor

SQL-Anweisung	Funktion
ALLOCATE DESCRIPTOR	SQL-Deskriptorbereich anfordern
DEALLOCATE DESCRIPTOR	SQL-Deskriptorbereich freigeben
DESCRIBE	Datentypen von Ein- oder Ausgabewerten abfragen
GET DESCRIPTOR	SQL-Deskriptorbereich lesen
SET DESCRIPTOR	SQL-Deskriptorbereich ändern

Tabelle 41: SQL-Anweisungen für Deskriptor

8.1.6 WHENEVER-Anweisung zur ESQL-Fehlerbehandlung

SQL-Anweisung	Funktion
WHENEVER	Fehlerbehandlung definieren (nicht ausführbar)

Tabelle 42: WHENEVER-Anweisung zur ESQL-Fehlerbehandlung

8.1.7 SQL-Anweisungen zur Verwaltung der Speicherstruktur

Storage Group

SQL-Anweisung	Funktion
ALTER STOGROUP	Storage Group ändern
CREATE STOGROUP	Storage Group erzeugen
DROP STOGROUP	Storage Group löschen

Tabelle 43: SQL-Anweisungen für Storage Groups

Space

SQL-Anweisung	Funktion
ALTER SPACE	Space-Parameter ändern
CREATE SPACE	Space erzeugen
DROP SPACE	Space löschen

Tabelle 44: SQL-Anweisungen für Spaces

Index

SQL-Anweisung	Funktion
CREATE INDEX	Index erzeugen
DROP INDEX	Index löschen
REORG STATISTICS	Globale Statistik neu erzeugen

Tabelle 45: SQL-Anweisungen für Indizes

8.1.8 SQL-Anweisungen zur Verwaltung von Benutzereinträgen

SQL-Anweisung	Funktion
CREATE SYSTEM_USER	Systemzugang erzeugen
CREATE USER	Berechtigungsschlüssel erzeugen
DROP USER	Berechtigungsschlüssel löschen
DROP SYSTEM_USER	Systemzugang löschen

Tabelle 46: SQL-Anweisungen zur Verwaltung von Benutzereinträgen

8.1.9 Utility-Anweisungen

Utility-Anweisungen sind Anweisungen in SQL-Syntax für die Datenbankverwaltung.

Die Utility-Anweisungen sind im Handbuch „ SQL-Sprachbeschreibung Teil 2: Utilities“ beschrieben.

8.1.10 Kontrollanweisungen

Routine (Stored Procedure und UDF)

SQL-Anweisung[1]	Funktion
COMPOUND	Anweisungen im Kontext
CALL	Prozedur aufrufen
CASE	Anweisungen bedingt ausführen
FOR	Anweisungen in einer Schleife ausführen
IF	Anweisungen bedingt auführen
ITERATE	zum nächsten Schleifendurchlauf wechseln
LEAVE	Schleife oder COMPOUND-Anweisung beenden
LOOP	Anweisungen in einer Schleife ausführen
REPEAT	Anweisungen in einer Schleife ausführen
RETURN	Rückgabewert einer User Defined Function (UDF) liefern
SET	Wert zuweisen
WHILE	Anweisungen in einer Schleife ausführen

Tabelle 47: SQL-Anweisungen für Prozeduren

[1]nur innerhalb einer CREATE PROCEDURE- oder CREATE FUNCTION-Anweisung

8.1.11 Diagnoseanweisungen

Routine (Stored Procedure und UDF)

SQL-Anweisung	Funktion
GET DIAGNOSTICS	Diagnoseinformationen zu einer Anweisung ausgeben
SIGNAL	Fehler in Routine melden
RESIGNAL	Fehler in lokaler Fehler-Routine melden

Tabelle 48: SQL-Anweisungen für Prozeduren

8.2 Alphabetische Beschreibung

Dieser Abschnitt beschreibt ausführlich Syntax und Funktion der SQL-Anweisungen.

8.2.1 Beschreibungsformat

In diesem Abschnitt sind die SQL-Anweisungen in einem einheitlichen Stil beschrieben. Die Anweisungen sind alphabetisch angeordnet. Pro Anweisung gibt es einen Eintrag, der den Namen der Anweisung als Kopfzeile hat.

Aufbau eines Eintrags

Jeder Eintrag besteht aus mehreren Abschnitten.
Bei einem Eintrag können Abschnitte fehlen, wenn sie für die entsprechende Anweisung keine Bedeutung haben. Außerdem können nach der Erklärung der Syntax weitere Abschnitte vorhanden sein, die Besonderheiten der jeweiligen Anweisung beschreiben. Die wichtigsten Abschnitte sind hier zusammengestellt.

Anweisungsname - Kurzbeschreibung

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

```
ANWEISUNGSNAME KLAUSEL parameter ...
```

parameter

Erklärungen zum Parameter.

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

Beispiele

Dieser Abschnitt zeigt ein oder mehrere Beispiele für die Verwendung der Anweisung.
Die meisten Beispiele beziehen sich auf die Beispieldatenbank AUFTRAGKUNDEN.

Handbuchbeispiele werden durch nebenstehendes Symbol gekennzeichnet, wenn sie als Bestandteil einer Anweisungsdatei oder eines ESQL-COBOL-Programms in der Beispieldatenbank von SESAM /SQL enthalten sind (siehe „ Basishandbuch“).

Siehe auch

Verwandte Anweisungen

8.2.2 SQL-Anweisungen in Routinen

In folgenden SQL-Anweisungen zur Erstellung und Gestaltung von Routinen können weitere SQL-Anweisungen verwendet werden:

- CREATE FUNCTION, CREATE PROCEDURE
- CASE, COMPOUND (dort auch in Fehler-Routinen), FOR, IF, LOOP, REPEAT, WHILE

Für einige dieser Anweisungen sind Einschränkungen zu beachten.

Um diese Anweisungen leichter lesbar zu machen, wird das Syntaxelement *routine_sql_anweisung* für diese weiteren SQL-Anweisungen hier zentral beschrieben.

---

routine_sql_anweisung ::=

{

case_anweisung

for_anweisung

if_anweisung

iterate_anweisung

leave_anweisung

loop_anweisung

repeat_anweisung

set_anweisung

while_anweisung

return_anweisung

call_anweisung

single_row_select_anweisung

insert_anweisung

update_searched_anweisung

delete_searched_anweisung

merge_anweisung

open_anweisung

fetch_anweisung

update_positioned_anweisung

delete_positioned_anweisung

close_anweisung

get_diagnostics_anweisung

signal_anweisung

resignal_anweisung

}

routine_sql_anweisung

routine_sql_anweisung hat eine maximale Länge von 32000 Zeichen.

Die erlaubten SQL-Anweisungen sind in folgenden Gruppen dargestellt:

- Kontrollanweisungen

  case_anweisung

  CASE-Anweisung, die bedingt weitere SQL-Anweisungen durchführt, siehe Abschnitt „CASE - SQL-Anweisungen bedingt ausführen“.

  for_anweisung

  FOR-Anweisung, die weitere SQL-Anweisungen in einer Schleife durchführt, siehe Abschnitt „FOR - SQL-Anweisungen in einer Schleife ausführen“.

  if_anweisung

  IF-Anweisung, die bedingt weitere SQL-Anweisungen durchführt, siehe Abschnitt „IF - SQL-Anweisungen bedingt ausführen“.

  iterate_anweisung

  ITERATE-Anweisung, die zum nächsten Schleifendurchlauf wechselt, siehe Abschnitt „ITERATE - zum nächsten Schleifendurchlauf wechseln“.

  leave_anweisung

  LEAVE-Anweisung, die Schleifen oder COMPOUND-Anweisungen abbricht, siehe Abschnitt „LEAVE - Schleife oder COMPOUND-Anweisung beenden“.

  loop_anweisung

  LOOP-Anweisung, die in einer Schleife weitere SQL-Anweisungen durchführt, siehe Abschnitt „LOOP - SQL-Anweisungen in einer Schleife ausführen“.

  repeat_anweisung

  REPEAT-Anweisung, die weitere SQL-Anweisungen in einer Schleife durchführt, siehe Abschnitt „REPEAT - SQL-Anweisungen in einer Schleife ausführen“.

  set_anweisung

  SET-Anweisung, die einem Prozedurparameter oder einer lokalen Prozedurvariablen einen Wert zuweist, siehe Abschnitt „SET - Wert zuweisen“.

  while_anweisung

  WHILE-Anweisung, die weitere SQL-Anweisungen in einer Schleife durchführt, siehe Abschnitt „WHILE - SQL-Anweisungen in einer Schleife ausführen“.

  return_anweisung

RETURN-Anweisung, die einen Rückgabewert für die UDF liefert, siehe Abschnitt „RETURN - Rückgabewert einer User Defined Function (UDF) liefern“ .
Diese Anweisung darf in Prozeduren nicht verwendet werden.

call_anweisung

CALL-Anweisung, die eine weitere Prozedur aufruft, siehe Abschnitt „CALL - Prozedur ausführen“.

> **i** Die Pragmas DEBUG ROUTINE und LOOP LIMIT haben vor einer CALL-Anweisung in einer Prozedur keine Wirkung, siehe Abschnitt „CALL - Prozedur ausführen“.Pragmas zur Optimierung können auch bei einer CALL-Anweisung in einer Prozedur angegeben werden. Sie wirken sich dann auf die Optimierung der Aufrufwerte aus.

- SQL-Anweisungen zum Abfragen und Ändern von Daten ohne Cursor

  single_row_select_anweisung

  SELECT-Anweisung, die einen einzelnen Satz liest, siehe Abschnitt „SELECT - Einzelnen Satz lesen“.

  insert_anweisung

  INSERT-Anweisung, die Sätze in eine bestehende Tabelle einfügt, siehe Abschnitt „INSERT - Sätze in Tabelle einfügen“.
  Diese Anweisung darf in UDFs nicht verwendet werden.

  update_searched_anweisung

  UPDATE-Anweisung, die in einer Tabelle die Spalten der Sätze ändert, die eine bestimmte Suchbedingung erfüllen, siehe Abschnitt „UPDATE - Spaltenwerte ändern“.
  Diese Anweisung darf in UDFs nicht verwendet werden.

  delete_searched_anweisung

  DELETE-Anweisung, die in einer Tabelle die Sätze löscht, die eine bestimmte Suchbedingung erfüllen, siehe Abschnitt „DELETE - Sätze löschen“
  "DELETE - Sätze löschen".
  Diese Anweisung darf in UDFs nicht verwendet werden.

  merge_anweisung

  MERGE-Anweisung, die in Abhängigkeit von einer bestimmten Bedingung Sätze in einer Tabelle ändert oder Sätze in eine Tabelle einfügt, siehe Abschnitt „MERGE - Sätze in Tabelle einfügen oder Spaltenwerte ändern“.Diese Anweisung darf in UDFs nicht verwendet werden.

- SQL-Anweisungen zum Abfragen und Ändern von Daten mit Cursor

  Diese Anweisungen sind nur erlaubt für einen lokalen Cursor, der innerhalb einer COMPOUND-Anweisung definiert ist.

  open_anweisung

  OPEN-Anweisung, die einen Cursor öffnet, siehe Abschnitt „OPEN - Cursor öffnen“.

  fetch_anweisung

  FETCH-Anweisung, die einen Cursor positioniert und ggf. den aktuellen Satz liest, siehe Abschnitt „FETCH - Cursor positionieren und Satz lesen“.

  update_positioned_anweisung

UPDATE-Anweisung, die in einer Tabelle die Spalten des Satzes ändert, auf den der Cursor positioniert ist, siehe Abschnitt „UPDATE - Spaltenwerte ändern“.
Diese Anweisung darf in UDFs nicht verwendet werden.

delete_positioned_anweisung

DELETE-Anweisung, die in einer Tabelle den Satz löscht, auf den der Cursor positioniert ist, siehe Abschnitt „DELETE - Sätze löschen“. Diese Anweisung darf in UDFs nicht verwendet werden.

close_anweisung

CLOSE-Anweisung, die einen Cursor schließt, siehe Abschnitt „CLOSE - Cursor schließen“.

- Diagnoseanweisungen

  get_diagnostics_anweisung

  GET DIAGNOSTICS-Anweisung zur Ausgabe von Diagnoseinformationen, siehee Abschnitt „GET DIAGNOSTICS - Diagnoseinformationen ausgeben“.

  signal_anweisung

  SIGNAL-Anweisung, die einen Fehler in der Routine meldet, siehe Abschnitt „SIGNAL - Fehler in Routine melden“.

  resignal_anweisung

  RESIGNAL-Anweisung, die einen Fehler in der Fehler-Routine meldet siehe Abschnitt „RESIGNAL - Fehler in lokaler Fehler-Routine melden“.

8.2.3 Beschreibung der SQL-Anweisungen

- ALLOCATE DESCRIPTOR - SQL-Deskriptorbereich anfordern
- ALTER SPACE - Space-Parameter ändern
- ALTER STOGROUP - Storage Group ändern
- ALTER TABLE - Basistabelle ändern
- CALL - Prozedur ausführen
- CASE - SQL-Anweisungen bedingt ausführen
- CLOSE - Cursor schließen
- COMMIT WORK - Transaktion beenden
- COMPOUND - SQL-Anweisungen in gemeinsamem Kontext ausführen
- CREATE FUNCTION - User Defined Function (UDF) erzeugen
- CREATE INDEX - Index erzeugen
- CREATE PROCEDURE - Prozedur erzeugen
- CREATE SCHEMA - Schema erzeugen
- CREATE SPACE - Space erzeugen
- CREATE STOGROUP - Storage Group erzeugen
- CREATE SYSTEM_USER - Systemzugang erzeugen
- CREATE TABLE - Basistabelle erzeugen
- CREATE USER - Berechtigungsschlüssel erzeugen
- CREATE VIEW - View erzeugen
- DEALLOCATE DESCRIPTOR - SQL-Deskriptorbereich freigeben
- DECLARE CURSOR - Cursor vereinbaren
- DELETE - Sätze löschen
- DESCRIBE - Datentypen von Ein- und Ausgabewerten abfragen
- DROP FUNCTION - User Defined Function (UDF) löschen
- DROP INDEX - Index löschen
- DROP PROCEDURE - Prozedur löschen
- DROP SCHEMA - Schema löschen
- DROP SPACE - Space löschen
- DROP STOGROUP - Storage Group löschen
- DROP SYSTEM_USER - Systemzugang löschen
- DROP TABLE - Basistabelle löschen
- DROP USER - Berechtigungsschlüssel löschen
- DROP VIEW - View löschen
- EXECUTE - Vorbereitete Anweisung ausführen
- EXECUTE IMMEDIATE - Dynamisch formulierte Anweisung ausführen
- FETCH - Cursor positionieren und Satz lesen
- FOR - SQL-Anweisungen in einer Schleife ausführen

- GET DIAGNOSTICS - Diagnoseinformationen ausgeben
- GET DESCRIPTOR - SQL-Deskriptorbereich lesen
- GRANT - Privilegien vergeben
- IF - SQL-Anweisungen bedingt ausführen
- INCLUDE - ESQL-Programmteile einfügen
- INSERT - Sätze in Tabelle einfügen
- ITERATE - zum nächsten Schleifendurchlauf wechseln
- LEAVE - Schleife oder COMPOUND-Anweisung beenden
- LOOP - SQL-Anweisungen in einer Schleife ausführen
- MERGE - Sätze in Tabelle einfügen oder Spaltenwerte ändern
- OPEN - Cursor öffnen
- PERMIT - Benutzeridentifikation für SESAM/SQL V1.x angeben
- PREPARE - Dynamisch formulierte Anweisungen vorbereiten
- REORG STATISTICS - Globale Statistik neu erzeugen
- REPEAT - SQL-Anweisungen in einer Schleife ausführen
- RESIGNAL - Fehler in lokaler Fehler-Routine melden
- RESTORE - Cursor wiederherstellen
- RETURN - Rückgabewert einer User Defined Function (UDF) liefern
- REVOKE - Privilegien entziehen
- ROLLBACK WORK - Transaktion zurücksetzen
- SELECT - Einzelnen Satz lesen
- SET - Wert zuweisen
- SET CATALOG - Datenbanknamen voreinstellen
- SET DESCRIPTOR - SQL-Deskriptorbereich ändern
- SET SCHEMA - Schemanamen voreinstellen
- SET SESSION AUTHORIZATION - Berechtigungsschlüssel festlegen
- SET TRANSACTION - Transaktionseigenschaften festlegen
- SIGNAL - Fehler in Routine melden
- STORE - Cursorposition speichern
- UPDATE - Spaltenwerte ändern
- WHENEVER - Fehlerbehandlung definieren
- WHILE - SQL-Anweisungen in einer Schleife ausführen

8.2.3.1 ALLOCATE DESCRIPTOR - SQL-Deskriptorbereich anfordern

ALLOCATE DESCRIPTOR legt einen SQL-Deskriptorbereich an. Der Deskriptorbereich dient bei dynamisch formulierten Anweisungen und Cursorbeschreibungen als Schnittstelle zwischen dem Anwendungsprogramm und der SQL-Datenbank.

Aufbau und Verwendung eines Deskriptorbereichseintrags sind im Abschnitt „Deskriptorbereich“ beschrieben. Nach dem Anlegen des Bereichs mit ALLOCATE DESCRIPTOR ist der Inhalt zunächst noch nicht definiert.

`ALLOCATE DESCRIPTOR GLOBAL` *deskriptor* `[WITH MAX` *anzahl* `]`

deskriptor `::= {` *alphanumerisches_literal* `| :` *benutzervariable* `}`

anzahl `::= {` *ganzzahl* `| :` *benutzervariable* `}`

GLOBAL

Der angelegte Deskriptorbereich kann in jeder Übersetzungseinheit der aktuellen SQL-Session verwendet werden.

deskriptor

Zeichenkette, die den Namen des SQL-Deskriptorbereichs enthält. Für *deskriptor* können Sie ein alphanumerisches Literal (nicht in der sedezimaler Form) oder eine alphanumerische Benutzervariable vom SQL-Datentyp CHAR(*n*) angeben, mit 1 <= *n* <= 18.

Der Deskriptorbereichsname kann mit einem oder mehreren Leerzeichen beginnen und enden. Nach Entfernen von führenden und abschließenden Leerzeichen muss die verbleibende Zeichenfolge ein einfacher Name sein (siehe Abschnitt „Einfache Namen“).

Zwei Deskriptorbereichsnamen gelten als gleich, wenn nach Entfernen der Leerzeichen die verbleibenden einfachen Namen gleich sind (siehe Abschnitt „Gleichheit von einfachen Namen“).

anzahl

Maximale Anzahl von Einträgen im SQL-Deskriptorbereich.

Für *anzahl* können Sie eine Ganzzahl oder eine Benutzervariable vom SQL-Datentyp SMALLINT angeben mit 1 <= *anzahl* <= 1000.

anzahl bestimmt die Größe des reservierten SQL-Deskriptorbereichs.
Wenn Sie längere alphanumerische Werte im Deskriptorbereich ablegen, reicht der Platz im Deskriptorbereich unter Umständen nicht aus, und Sie erhalten einen entsprechendenSQLSTATE. In diesem Fall müssen Sie *anzahl* erhöhen (siehe Beispiel).

Zur Speicherung von SQL-Deskriptorbereichen wird in UTM-Anwendungen das „UTM-Vorgangsmemory“ verwendet. Reicht dieser Speicherplatz nicht aus, so erfolgt eine Fehlermeldung.

WITH MAX *anzahl* nicht angegeben:
Für *anzahl* ist der Wert 20 voreingestellt.

Beispiele

SQL-Deskriptorbereich für maximal 100 Einträge anlegen:

```
ALLOCATE DESCRIPTOR GLOBAL :demo_desc WITH MAX 100
```

SQL-Deskriptorbereich für 100 Einträge anlegen. Der Deskriptorbereich soll so groß sein, dass die Einträge Werte vom Typ CHAR(80) speichern können:

```
ALLOCATE DESCRIPTOR GLOBAL :demo_desc WITH MAX 200
```

Siehe auch

DEALLOCATE DESCRIPTOR, DESCRIBE, GET DESCRIPTOR, SET DESCRIPTOR

8.2.3.2 ALTER SPACE - Space-Parameter ändern

ALTER SPACE ändert Parameter für den Catalog-Space oder einen Anwender-Space.

Welche Anwender-Spaces definiert sind, erfahren Sie im View SPACES des INFORMATION_SCHEMA (siehe Kapitel „Informationsschemata“).

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

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

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

space

Name des Space, für den Parameter geändert werden sollen.

Der einfache Spacename kann durch einen Datenbanknamen qualifiziert werden.

Der universelle Benutzer darf den Spacenamen `"CATALOG"` (in Anführungszeichen) angeben, auch wenn er nicht der Eigentümer des Space ist. Dabei darf er den Parameter NO LOG nicht angeben.

PCTFREE *prozent*

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

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

NO LOG

Logging ausschalten.

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

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

USING STOGROUP *stogroup*

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

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

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

Beispiel

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

```
ALTER SPACE indexspace PCTFREE 20 USING STOGROUP stogroup3
```

Siehe auch

CREATE SPACE, CREATE STOGROUP

8.2.3.3 ALTER STOGROUP - Storage Group ändern

ALTER STOGROUP ändert die Definition einer Storage Group.

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

Welche Storage Groups definiert sind, erfahren Sie im View STOGROUPS des INFORMATION_SCHEMA (siehe Kapitel „Informationsschemata“).

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

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

stogroup

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

ADD VOLUMES (*volumename*,...)

Fügt neue Privatplatten zur Storage Group hinzu. *volumename* gibt das Datenträgerkennzeichen der Platten als alphanumerisches Literal an. Jedes Datenträgerkennzeichen darf nur einmal für eine Storage Group angegeben werden.

Bestand die Storage Group bisher aus Privatplatten, so müssen die neu hinzugefügten Platten den gleichen Gerätetyp besitzen.

Eine Storage Group darf insgesamt max. 100 Platten umfassen.

ON *gerätetyp*

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

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

DROP VOLUMES (*volumename*,...)

Löscht einzelne Privatplatten aus der Definition der Storage Group. *volumename* gibt das Datenträgerkennzeichen der Platte als alphanumerisches Literal an.

Die letzte Platte einer Storage Group können Sie nicht löschen.

PUBLIC

Die Storage Group wird auf das voreingestellte Pubset der BS2000-Benutzerkennung gesetzt, unter der der DBH läuft. Alle Privatplatten werden aus der Definition der Storage Group gelöscht.

TO *catid*

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

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

Auswirkungen von ALTER STOGROUP

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

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

Beispiele

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

```
ALTER STOGROUP stogroup4 PUBLIC

ALTER STOGROUP stogroup4 TO 'O'
```

Das folgende Beispiel ändert die Storage Group STOGROUP5 von PUBLIC auf Privatplatten.
Die Katalogkennung für die Dateien der Storage Group bleibt unverändert.

```
ALTER STOGROUP auftragkunden.stogroup5

      ADD VOLUMES ('DX017A','DX017B') ON 'D3435'
```

Siehe auch

CREATE STOGROUP

8.2.3.4 ALTER TABLE - Basistabelle ändern

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

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

Eine BLOB-Tabelle kann mit ALTER TABLE geändert werden. Im Abschnitt „Besonderheiten für BLOB-Tabellen“ ist beschrieben, was dabei zu beachten ist.

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

Die Tabellenart können Sie mit ALTER TABLE nicht ändern! Änderungen der Tabellenart nehmen Sie mit der UTILITY-Anweisung MIGRATE vor (siehe Handbuch „ SQL-Sprachbeschreibung Teil 2: Utilities“).

Welche Basistabellen definiert sind, erfahren Sie im View BASE_TABLES des INFORMA-TION_SCHEMA (siehe Kapitel „Informationsschemata“).

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

```
ALTER TABLE tabelle

{

   ADD [COLUMN] spaltendefinition ,...

       [ADD INDEX indexdefinition ,... [USING SPACE space ]] |

   ALTER [COLUMN] spalte aktion [, spalte aktion ] ...
          [USING FILE fehlerdatei [PASSWORD kennwort ]] |

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

   ADD [CONSTRAINT integritätsbedingungsname ] tabellenbedingung
       [,[CONSTRAINT integritätsbedingungsname ] tabellenbedingung ],... |

   DROP CONSTRAINT integritätsbedingungsname { CASCADE | RESTRICT }

}


aktion ::=
{

   DROP DEFAULT |
```

```
    SET datentyp [CALL DML call_dml_voreinst ] |
    SET voreinstellung
}

indexdefinition ::= index ({ spalte [LENGTH länge ]},...)

voreinstellung ::= DEFAULT
{
  alphanumerisches_literal
  national_literal
  numerisches_literal
  zeit_literal
  CURRENT_DATE
  CURRENT_TIME(3)
  LOCALTIME(3)
  CURRENT_TIMESTAMP(3)
  LOCALTIMESTAMP(3)
  USER
  SYSTEM_USER
  NULL
  REF( tabelle )
}
```

tabelle

Name einer Basistabelle.

ADD [COLUMN] *spaltendefinition*,...

Fügt der Basistabelle neue Spalten hinzu. Die neuen Spalten werden nach den vorhandenen Spalten eingefügt. *spaltendefinition* definiert die Spalten, siehe Abschnitt „Spaltendefinition“.

Enthält eine *spaltendefinition* einen von NULL verschiedenen Defaultwert, so wird dieser in alle vorhandenen Sätze der Tabelle eingefügt; dies kann je nach Anzahl der vorhandenen Sätze entsprechend dauern.

Sie dürfen in *spaltendefinition* keine Primärschlüsselbedingung definieren.

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

Soll eine FOR REF-Spalte angefügt werden, so ist es sinnvoll die Spalte zunächst nicht mit der FOR REF-Klausel zu definieren, denn in diesem Fall würde die Voreinstellung für die REF-Spalte in jede Zeile der Spalte eingetragen werden. Effizienter in Bezug auf den Speicherbedarf ist es, zunächst die Spalte mit dem Datentyp

CHAR(237) zu definieren. Bei dieser Spaltendefinition wird in jede Zeile der NULL-Wert eingetragen. Danach kann die Spalte mit ALTER COLUMN *spalte* SET DEFAULT REF(*tabelle*) geändert werden. Dies beeinflusst bis dahin bestehende Zeileneinträge nicht mehr.

ADD INDEX *indexdefinition*

Definition eines oder mehrerer Indizes für die neu eingefügten Spalten.

Bei der Index-Definition gelten die Regeln und Randbedingungen der CREATE IN-DEX-Anweisung, siehe Abschnitt „CREATE INDEX - Index erzeugen“

¡ Das Pragma UTILITY MODE ON darf nicht zusammen mit ADD INDEX verwendet werden.

index

Name des neuen Index.

spalte

Name der Spalte der Basistabelle, die zum Index gehören soll. Es dürfen nur Spalten angegeben werden, die in der ADD COLUMN-Klausel spezifiziert sind.

LENGTH *länge*

Gibt an, bis zu welcher Länge die Spalte in den Index einbezogen werden soll.

LENGTH *länge* nicht angegeben:

Die Spalte wird in ihrer ganzen Länge in Bytes in den Index einbezogen.

USING SPACE *space*

Name des Space, in dem der oder die Indizes gespeichert werden sollen.

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

USING SPACE *space* nicht angegeben:
Der Index wird im Space der Basistabelle gespeichert. Bei einer partitionierten Tabelle wird der Index auf dem Space der ersten Partition gespeichert.

ALTER [COLUMN] *spalte*

spalte ist der Name der Spalte, die geändert werden soll.

Die Änderungen der Spalte werden in der folgenden Reihenfolge ausgeführt:

- DROP DEFAULT
- SET *datentyp*
- SET *voreinstellung*

Für eine Spalte darf dieselbe Änderungsart nur einmal veranlasst werden.

DROP DEFAULT

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

Die zu Grunde liegende Basistabelle darf keine CALL-DML-Tabelle sein.

SET *datentyp*

Neuer Datentyp der Spalte.

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

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

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

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

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

Ursprünglicher Datentyp	**Neuer Datentyp**	**Neuer Datentyp**	**Neuer Datentyp**	**Neuer Datentyp**	**Neuer Datentyp**	**Neuer Datentyp**	**Neuer Datentyp**	**Neuer Datentyp**	**Neuer Datentyp**
	INTEGER SMALL-INT DECIMAL NUMERIC	REAL DOUBLE PRE-CISION FLOAT	VAR-CHAR	CHAR	NVAR-CHAR	NCHAR	DATE	TIME(3)	TIME-STAMP (3)
INTEGER SMALLINT DECIMAL NUMERIC	ja	ja [1]	nein	ja	nein	ja [1]	nein	nein	nein
REAL DOUBLE PRE-CISION FLOAT	ja	ja	nein	ja	nein	ja	nein	nein	nein
VARCHAR	nein	nein	ja [2]	nein	nein	nein	nein	nein	nein
CHAR	ja	ja [1]	nein	ja	nein	ja[4]	ja [1]	ja [1]	ja [1]
NVAR-CHAR	nein	nein	nein	nein	ja [3]	nein	nein	nein	nein
NCHAR	ja	ja	nein	ja[4]	nein	ja	ja	ja	ja

DATE	nein	nein	nein	ja	nein	ja	ja	nein	nein
TIME(3)	nein	nein	nein	ja	nein	ja	nein	ja	nein
TIME-STAMP(3)	nein	nein	nein	ja	nein	ja	nein	nein	ja

Tabelle 49: Zulässige und verbotene Kombinationen beim Ändern von Datentypen

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

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

[3)]Eine Spalte des Datentyps NVARCHAR darf nur in den neuen Datentyp NVARCHAR mit *neue_länge* >= *alte_länge* geändert werden. Die übrigen Datentypen dürfen nicht in den Datentyp NVARCHAR geändert werden und umgekehrt.

[4)] Für die Datenbank muss eine Code-Tabelle ungleich _NONE_ definiert sein.

SESAM/SQL konvertiert satzweise alle Werte von *spalte* in den neuen Datentyp. Bei multiplen Spalten konvertiert SESAM/SQL die signifikanten Werte aller Ausprägungen, deren Positionsnummer kleiner oder gleich der Dimension des neuen Datentyps ist. Dabei kann sich die Position eines Elements innerhalb der multiplen Spalte ändern: Ist das Ergebnis der Konvertierung für ein Spaltenelement der NULL-Wert, werden alle nachfolgenden Elemente, deren Positionsnummer kleiner oder gleich der Dimension des neuen Datentyps ist, nach links verschoben, der NULL-Wert wird hinten angehängt. Bei der Konvertierung eines Spaltenwerts gelten (außer bei CHAR <-> NCHAR) die gleichen Regeln wie bei der Konvertierung eines Werts durch den CAST-Ausdruck (siehe „Regeln für die Konvertierung eines Wertes in einen anderen Datentyp“). Bei der Konvertierung eines Spaltenwerts von CHAR nach NCHAR und umgekehrt gelten die gleichen Regeln wie bei der Transliteration eines Werts durch den TRANSLATE-Ausdruck, bei dem in der USING-Klausel CATALOG_DEFAULT verwendet wird (siehe Abschnitt „TRANSLATE() - Zeichenkette transliterieren bzw. transcodieren“). Diese Regeln gelten auch für die Konvertierung des Spaltenelement-Wertes einer multiplen Spalte.

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

Das Runden eines Wertes ist kein Konvertierungsfehler.

Beispiel

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

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

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

  Beispiel
  Der Wert 'Kundendienst' einer Spalte vom alphanumerischen Datentyp CHAR(12) bzw. vom National-

Datentyp NCHAR(12) soll in den Datentyp CHAR(6) bzw. NCHAR(6) konvertiert werden. Der ursprüngliche Spaltenwertwird ersetzt durch den Wert 'Kunden'. SESAM/SQL gibt eine Warnung aus.

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

  Beispiel
  Eine multiple Spalte des alphanumerischen Datentyps (7) CHAR (20) bzw. des National-Datentyps (7) NCHAR (20) soll in den Datentyp (5) CHAR (20) bzw. (5) NCHAR (20) konvertiert werden. Bei einigen Tabellensätzen enthalten alle 7 Elemente der multiplen Spalte einen alphanumerischen Wert.
- Nicht konvertierbare Werte
  Durch eine Datentypänderung kommt es bei bestimmten Spaltenwerten zu einem Wertverlust mit Fehlermeldung (data exception).

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

    Beispiel
    Der Wert 9999 einer Spalte vom Datentyp INTEGER soll in den Datentyp NUMERIC(2,0) konvertiert werden.
  - Eine Spalte vom alphanumerischen Datentyp CHAR bzw. vom National-Datentyp NCHAR wird in einen numerischen Datentyp geändert. Der ursprüngliche Wert der Spalte ist nicht als numerischer Wert darstellbar.

    Beispiel
    Der Wert 'Otto' einer Spalte vom alphanumerischen Datentyp CHAR(4) bzw. vom National-Datentyp NCHAR(4) soll in den Datentyp INTEGER konvertiert werden.
  - Die Länge des Wertes einer ursprünglich numerischen Spalte oder der Spalte eines Zeit-Datentyps ist für den alphanumerischen Zieldatentyp CHAR bzw. für den National-Datentyp NCHAR zu groß.

    Beispiel
    Der Wert 9999 einer Spalte vom Datentyp INTEGER soll in den alphanumerischen Datentyp CHAR (2) bzw. in den National-Datentyp NCHAR(2) konvertiert werden.

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

CALL DML *call_dml_voreinst*

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

CALL DML *call_dml_voreinst* nicht angegeben:

Bezieht sich die Datentypänderung auf die Spalte einer CALL-DML/SQL-Tabelle, so behält *spalte* den ihr in der Spaltendefinition zugeordneten nicht-signifikanten Attributwert.

Bezieht sich die Datentypänderung auf die Spalte einer Nur-CALL-DML-Tabelle, also einer Tabelle mit „alten“ Attributformaten aus den SESAM-Versionen < V13.1, so erhält *spalte* folgenden nicht-signifikanten Attributwert:

- das Leerzeichen, wenn der Datentyp von *spalte* alphanumerisch ist
- die Ziffer 0, wenn der Datentyp von *spalte* numerisch ist.

SET *voreinstellung*

Legt einen neuen SQL-Defaultwert für die Spalte fest.

Die zu Grunde liegende Basistabelle darf keine CALL-DML-Tabelle sein.

spalte darf keine multiple Spalte sein.

voreinstellung muss den Zuweisungsregeln für Defaultwerte genügen (siehe Abschnitt „Defaultwerte für Tabellenspalten“).

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

USING FILE *fehlerdatei* [PASSWORD *kennwort*]

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

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

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

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

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

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

PASSWORD *kennwort*

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

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

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

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

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

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

Für *spalte* darf keine Primärschlüsselbedingung definiert sein.

Sie dürfen nicht alle Spalten der Basistabelle angeben.

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

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

CASCADE

Die benannte(n) Spalte(n) und zugehörige Indizes werden gelöscht.
Integritätsbedingungen anderer Tabellen oder Spalten, die *spalte* verwenden, werden ebenfalls gelöscht.
Alle Routinen werden gelöscht, die sich direkt oder indirekt auf diese Spalte beziehen.

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

RESTRICT

Das Löschen einer Spalte ist nur eingeschränkt möglich:
Die Spalte kann nicht gelöscht werden, wenn sie in einer View-Definition oder in einer Routine verwendet

wird. Für die zu löschende Spalte darf nur dann ein Index definiert sein, wenn keine der verbleibenden Spalten der Basistabelle in der betreffenden Indexdefinition benannt ist. Entsprechendes gilt für Integritätsbedingungen.

Das Pragma UTILITY MODE kann eingeschaltet werden, wenn für die Spalte kein Index definiert ist.

ADD CONSTRAINT-Klausel

Fügt der Basistabelle Integritätsbedingungen hinzu.

> ¡ Das Hinzufügen mehrerer Integritätsbedingungen in einer ALTER TABLE-Anweisung ist performanter als das Hinzufügen jeweils einer Integritätsbedingung in entsprechend vielen ALTER TABLE-Anweisungen.

CONSTRAINT *integritätsbedingungsname*

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

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

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

tabellenbedingung

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

DROP CONSTRAINT *integritätsbedingungsname* {CASCADE, RESTRICT}

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

CASCADE

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

RESTRICT

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

Besonderheiten für CALL-DML-Tabellen

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

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

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

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

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

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

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

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

Besonderheiten für BLOB-Tabellen

Auch BLOB-Tabellen können mit ALTER TABLE geändert werden. Diese Änderungen können jedoch dazu führen, dass die BLOB-Tabelle nicht mehr durch CLI-Aufrufe angesprochen werden kann. Mögliche Änderungen und deren Auswirkungen sind deshalb nachfolgend beschrieben:

- Wird eine neue Spalte in eine BLOB-Tabelle eingefügt, so hat dies keine Auswirkungen auf die Durchführbarkeit von CLI-Aufrufen.
- Auch zusätzliche Integritätsbedingungen auf BLOB-Tabellen können mit der ADD CONSTRAINT-Klausel ohne negative Folgen definiert werden.
- Wird jedoch eine der Spalten OBJ_NR, SLICE_NR, SLICE_VAL oder OBJ_REF gelöscht oder ihr Typ geändert, so können BLOB-Werte nicht mehr mit CLI-Funktionen bearbeitet werden.

Fehlerdatei der SQL-Anweisung ALTER TABLE

Beim Ändern einer Spalte (ALTER COLUMN) können Sie den Namen einer Fehlerdatei angeben. Bei Bedarf können Sie die Fehlerdatei über ein BS2000-Kennwort schützen. Die Fehlerdatei dient zur Aufnahme von Spaltenwerten, bei denen Konvertierungsfehler auf Grund einer Datentypänderung zu Datenverlust führten. Haben Sie eine Fehlerdatei benannt und treten bei Datentypänderungen Konvertierungsfehler auf, richtet SESAM/SQL die Fehlerdatei in der Kennung des DBH als SAM-Datei ein, sofern sie noch nicht vorhanden ist.

Soll die Fehlerdatei nicht in der DBH-Kennung liegen, müssen Vorbereitungen getroffen worden sein, siehe Abschnitt „Datenbankdateien und Jobvariablen auf fremden Benutzerkennungen“ im „ Basishandbuch“.

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

SESAM/SQL protokolliert die ursprünglichen Spaltenwerte zusammen mit der zugehörigen Fehlermeldung oder Warnung in die Fehlerdatei. Ist bereits eine Fehlerdatei vorhanden, so wird ihr Inhalt nicht überschrieben. Die neuen Einträge fügt SESAM/SQL an die bereits vorhandenen Einträge an.
Die Fehlerdatei unterliegt nicht der Transaktionssicherung. Sie bleibt erhalten, auch wenn implizit oder explizit die Transaktion zurückgesetzt wird, in der SESAM/SQL Einträge in die Fehlerdatei schreibt.
Den Inhalt der Fehlerdatei können Sie sich mit dem Kommando SHOW-FILE auf dem Bildschirm anzeigen lassen.

Inhalt der Fehlerdatei

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

---

eintrag ::=

satz_id
spaltenname [*posnr*]
sqlstate
spaltenwert

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

satz_id

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

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

spaltenname

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

sqlstate

SQLSTATE der zugehörigen Fehlermeldung bzw. Warnung.

spaltenwert

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

Datentyp der Spalte, die den ursprünglichen Wert enthält	**Darstellung des *spaltenwert* in der Fehlerdatei**
Datentyp einer CALL-DML-Tabellen Spalte von SESAM bis V13.1	Zeichenkette mit einer maximalen Länge von 54 Zeichen
CHAR VARCHAR	Zeichenkette mit einer maximalen Länge von 54 Zeichen
NCHAR NVARCHAR	Zeichenkette mit einer maximalen Länge von 27 Code Units

INTEGER, SMALLINT, NUMERIC, DECIMAL,	entsprechendes numerisches Literal (Ganzzahl oder Festpunktzahl)
FLOAT, REAL, DOUBLE PRECISION	entsprechendes numerisches Literal (Gleitpunktzahl)
DATE	Datum-Zeitliteral
TIME	Uhrzeit-Zeitliteral
TIMESTAMP	Zeitstempel-Zeitliteral

Tabelle 50: Darstellung der Datentypen von *spaltenwert*

Zeichenketten werden in der Fehlerdatei ohne umgebende Hochkommata dargestellt.

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

Beispiel

Das folgende Beispiel zeigt eine Fehlerdatei, die die ursprünglichen Spaltenwerte der Basistabelle LEISTUNG enthält.

Die Basistabelle LEISTUNG hat folgenden Aufbau:

```
SQL CREATE TABLE leistung
   (lnr    INTEGER CONSTRAINT lnr_primary PRIMARY KEY,
    anr    INTEGER CONSTRAINT l_anr_notnull NOT NULL,
    ldatum DATE, ...)
```

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

```
ALTER TABLE leistung ALTER COLUMN lsatz SET NUMERIC(5,2)

USING FILE 'ERR.LEISTUNG'
```

Auszug aus der Fehlerdatei ERR.LEISTUNG:

```
satz_id   spaltenname              sqlstate  spaltenwert

2               LSATZ                 22SA4         1500
3               LSATZ                 22SA4         1500
4               LSATZ                 22SA4         1200
5               LSATZ                 22SA4         1200
.
.
11              LSATZ                 22SA4         1200
```

Bei der Umwandlung von LSATZ von NUMERIC (5,0) in NUMERIC(5,2) können die Sätze mit dem angegebenen Primärschlüssel nicht konvertiert werden.

Beispiele

In den folgenden Beispielen werden die Tabelleneigenschaften der Tabellen KUNDE und AUFTRAG geändert:

Die Spalten KTELEFON und KINFO werden in die Tabelle KUNDE eingefügt.

```
ALTER TABLE kunde
            ADD COLUMN ktelefon CHARACTER(25), kinfo CHARACTER(50)
```

Der Datentyp der Spalte KNR in der Tabelle KUNDE wird geändert.
Der ursprüngliche Datentyp war NUMERIC, der neue Datentyp ist INTEGER.

```
ALTER TABLE kunde
            ALTER COLUMN knr SET INTEGER
```

Die Spalte KINFO der Tabelle KUNDE wird gelöscht.
Die Spalte KINFO wird nur gelöscht, wenn sie in keiner View-Definition verwendet wird. Für die Spalte KINFO dürfen nur dann ein Index oder eine Integritätsbedingung definiert sein, wenn keine der verbleibenden Spalten der Basistabelle in der Definition benannt ist.

```
ALTER TABLE kunde
            DROP COLUMN kinfo RESTRICT
```

Der Tabelle KUNDE wird eine Eindeutigkeitsbedingung hinzugefügt, die sich auf die Spalte KNR bezieht.

```
ALTER TABLE kunde
            ADD CONSTRAINT knr_unique UNIQUE (knr)
```

Die Referenzbedingung der Spalte KNR der Tabelle AUFTRAG auf die Spalte KNR der Tabelle KUNDE wird gelöscht. Sie können die Namen von verwendeten Integritätsbedingungen nachschlagen in den Views TABLE_CONSTRAINTS, REFERENTIAL_CONSTRAINTS und CHECK_CONSTRAINTS des INFORMATION-SCHEMA.

```
ALTER TABLE auftrag
            DROP CONSTRAINT a_knr_ref_kunde CASCADE
```

Siehe auch

CREATE TABLE

8.2.3.5 CALL - Prozedur ausführen

CALL führt eine Prozedur aus. CALL kann auch innerhalb einer Routine zur Ausführung einer anderen Prozedur verwendet werden (geschachtelte Aufrufe von Routinen),

Die CALL-Anweisung ist eine nicht-atomare SQL-Anweisung, da in der gerufenen Prozedur nicht-atomare Anweisungen enthalten sein können.

Prozeduren und ihre Verwendung in SESAM/SQL sind detailliert im Kapitel „Routinen“ beschrieben.

Welche Routinen definiert sind und welche Routinen einander verwenden, erfahren Sie in den Views für Routinen des INFORMATION_SCHEMA (siehe Kapitel „Informationsschemata“).

Wenn eine Prozedur Eingabeparameter erwartet, dann müssen die entsprechenden Werte (die Argumente) in der CALL-Anweisung an die Prozedur übergeben werden.

Ausgabewerte von Prozeduren, die außerhalb einer Routine aufgerufen werden, werden in entsprechenden Benutzervariablen oder im SQL-Deskriptorbereich abgelegt. Ausgabewerte von Prozeduren, die in einer übergeordneten Routine aufgerufen werden, werden in Ausgabeparameter oder in lokale Variablen der übergeordneten Prozedur eingetragen.

Die Pragmas DEBUG ROUTINE, DEBUG VALUE und LOOP LIMIT können zusätzlich verwendet werden. Siehe Abschnitt „Pragmas und Annotationen“.Sie werden nur interpretiert, wenn sie vor einer extern (also aus einer Anwendung) aufgerufenen CALL-Anweisung stehen und vererben ihre Wirkung dann auf alle direkt oder indirekt enthaltenen CALL-Anweisungen und User Defined Functions. Sie haben vor einer CALL-Anweisung in einer Prozedur keine Wirkung.

Pragmas zur Optimierung können auch bei einer CALL-Anweisung in einer Prozedur angegeben werden. Sie wirken sich dann auf die Optimierung der Aufrufwerte aus.

Zur Ausführung einer Prozedur benötigt der aktuelle Berechtigungsschlüssel das EXECUTE-Privileg für die auszuführende Prozedur, nicht aber diejenigen Privilegien, die benötigt werden, um die in der Prozedur enthaltenen DML-Anweisungen ausführen zu können. Zusätzlich werden die SELECT-Privilegien für die Tabellen benötigt, die in den Aufrufparametern der Routine über Unterabfragen angesprochen werden.

CALL *prozedur argumente*

prozedur ::= *routine*

argumente ::= ([*ausdruck* [{, *ausdruck* }...]])

---

prozedur

Name der auszuführenden Prozedur. Der einfache Prozedurname kann durch einen Datenbank- und Schemanamen qualifiziert werden.

([*ausdruck* [{, *ausdruck* }...]])

Liste der Argumente. Die Anzahl der Argumente muss mit der Anzahl der Parameter aus der Prozedur-Definition übereinstimmen. Die Argumente müssen in ihrer Reihenfolge mit den Parametern korrespondieren. Wenn kein Parameter für die Prozedur definiert ist, dann besteht die Liste nur aus den runden Klammern.

Wenn der n-te Parameter vom Typ IN oder INOUT ist, dann wird ihm vor Prozedur-Ausführung der Wert des n-ten Arguments zugewiesen.

Wenn der n-te Parameter vom Typ OUT oder INOUT ist, dann gilt:

- Wenn die CALL-Anweisung statisch formuliert ist, dann muss das n-te Argument eine Benutzervariable (ggf. mit Indikatorvariable) sein.
  Dieselbe Benutzervariable darf nicht als Argument für mehrere Parameter vom Typ OUT oder INOUT verwendet werden.
- Wenn die CALL-Anweisung dynamisch formuliert ist, dann muss das n-te Argument ein Platzhalter („?“) sein.

Nach Ausführung der Prozedur werden die Werte für die Parameter vom Typ OUT oder INOUT in die entsprechenden Benutzervariablen oder einen SQL-Deskriptorbereich übertragen.

Der Datentyp des n-ten Arguments muss mit dem Datentyp des n-ten Parameters verträglich sein. Für Eingabeparameter siehe die Hinweise im Abschnitt „Eingabeparameter für Routinen versorgen“. Für Ausgabeparameter siehe Abschnitt „Werte in Prozedurparameter (Ausgabe) oder lokale Variable eintragen“.

Wenn bei einer statisch formulierten SQL-Anweisung ein Parameter als Benutzervariable angegeben wird, dann geht SESAM/SQL bei der Vorübersetzung (ohne Datenbankkontakt) davon aus, dass es sich dabei um einen Parameter vom Typ IN oder IN-OUT handelt und überträgt diesen Wert an den DBH. Auch wenn es sich um einen reinen Ausgabeparameter handelt, muss daher entweder der Wert entsprechend dem Datentyp korrekt initialisiert werden oder die Benutzervariable erhält eine Indikatorvariable, die dann mit dem Wert -1 zu versorgen ist.

CALL und Transaktionssicherung

CALL leitet für Prozeduren, die außerhalb einer Routine aufgerufen werden, eine SQL-Transaktion ein, wenn keine Transaktion offen ist. Da eine Prozedur nur DML-Anweisungen enthält, wird mit CALL also eine SQL-Transaktion zur Datenmanipulation eingeleitet.

Die Prozeduranweisungen laufen im gleichen Isolationslevel und Transaktionsmodus ab wie die CALL-Anweisung (siehe Abschnitt „SET TRANSACTION - Transaktionseigenschaften festlegen“).

Wenn der Transaktionsmodus READ ONLY eingestellt ist, dann darf die Prozedur keine SQL-Anweisungen zum Ändern von Daten enthalten.

CALL und Zeitfunktionen

Kommen die Zeitfunktionen CURRENT_DATE, CURRENT_TIME(3), LOCALTIME(3), CURRENT_TIMESTAMP(3) und LOCALTIMESTAMP(3) innerhalb einer Anweisung mehrmals vor, werden diese simultan ausgewertet, siehe Abschnitt „Zeitfunktionen“. Diese Aussage gilt auch für Prozeduranweisungen. Dies bedeutet aber nicht, dass die Zeitfunktionen aller Anweisungen eines Prozedurablaufes simultan ausgewertet werden:

- Die Zeitfunktionen der CALL-Anweisung werden simultan ausgewertet, wenn sie als Wert in Eingabeparametern auftreten.
- Die Zeitfunktionen einer jeden Prozeduranweisung werden für sich simultan ausgewertet. Unterschiedliche Prozeduranweisungen liefern damit i.A. auch unterschiedliche Zeitwerte.
- Die Zeitfunktionen der COMPOUND-Anweisung werden simultan ausgewertet, wenn sie als Standardwert in Variablen-Definitionen auftreten.
- Die Zeitfunktionen einer IF-Anweisung werden für alle Suchbedingungen, sowohl im IF- wie auch im ELSIF-Zweig, simultan ausgewertet. Die Zeitfunktionen der Prozeduranweisungenin den THEN- und ELSE-Zweigen der IF-Anweisung werden aber wieder für sich simultan ausgewertet.

- Die Zeitfunktionen in Cursor-Beschreibungen lokaler Cursor werden simultan bei der OPEN-Anweisung für den Cursor ausgewertet.

Beispiel

Die Prozedur `GetCurrentYear` (siehe "CREATE PROCEDURE - Prozedur erzeugen") wird aufgerufen.

```
CALL ProcSchema.GetCurrentYear (OUT myvar)
```

Siehe auch

CREATE PROCEDURE, DROP PROCEDURE

8.2.3.6 CASE - SQL-Anweisungen bedingt ausführen

Die CASE-Anweisung führt SQL-Anweisungen abhängig von bestimmten Werten (einfache CASE-Anweisung) oder Bedingungen (CASE-Anweisung mit Suchbedingung) aus.

Die CASE-Anweisung darf nur in einer Routine angegeben werden, d.h. im Rahmen einer CREATE PROCEDURE- oder CREATE FUNCTION-Anweisung. Routinen und ihre Verwendung in SESAM/SQL sind detailliert im Kapitel „Routinen“ beschrieben.

Die CASE-Anweisung ist eine nicht-atomare SQL-Anweisung, d.h. in ihr können weitere (atomare oder nicht-atomare) SQL-Anweisungen auftreten.

Wenn die *suchbedingung* oder ein *ausdruck* einer CASE-Anweisung eine Tabelle anspricht, dann muss der Berechtigungsschlüssel, der die Routine mit CREATE PROCEDURE oder CREATE FUNCTION erzeugt, das SELECT-Privileg für diese Tabelle besitzen.

Ausführungshinweise

Die CASE-Anweisung ist eine nicht-atomare Anweisung:

- Wenn die CASE-Anweisung Teil einer COMPOUND-Anweisung ist, dann gelten die dort beschriebenen Regeln, insbesondere die dort definierten Fehler-Routinen.
- Wenn die CASE-Anweisung **nicht** Teil einer COMPOUND-Anweisung ist und eine der SQL-Anweisungen einen SQLSTATE meldet, dann werden ggf. nur die Änderungen dieser SQL-Anweisung rückgängig gemacht. Die CASE-Anweisung und die Routine, in der sie enthalten ist, werden abgebrochen. Die SQL-Anweisung, in der die Routine verwendet wurde, liefert den betreffenden SQLSTATE zurück.

Siehe auch

CREATE PROCEDURE, CREATE FUNCTION

Format der einfachen CASE-Anweisung

```
CASE ausdruckx
   WHEN ausdruck1 , ... THEN routine_sql_anweisung; [ routine_sql_anweisung; ] ...
   ...
   [ ELSE routine_sql_anweisung; [ routine_sql_anweisung; ] ... ]
END CASE
```

ausdruck

Ausdruck, dessen Auswertung einen alphanumerischen Wert, einen National-Wert, einen numerischen Wert oder einen Zeitwert ergibt.
Es darf kein multipler Wert mit Dimension > 1 sein.

ausdruck darf keine Benutzervariable enthalten.
Eine Spalte darf nur in einer Unterabfrage angegeben werden.

Die Werte von *ausdruckx* und *ausdruck1, ...* müssen verträgliche Datentypen haben (siehe Abschnitt „Verträglichkeit von Datentypen“).

routine_sql_anweisung

SQL-Anweisung, die in der THEN- oder ELSE-Klausel in Abhängigkeit von den Werten von *ausdruckx* und *ausdruck1, ...* ausgeführt werden soll.
Eine SQL-Anweisung wird mit „;“ (Semikolon) abgeschlossen.
Mehrere SQL-Anweisungen können nacheinander angegeben werden. Sie werden in der angegebenen Reihenfolge ausgeführt.
Vor der Durchführung einer SQL-Anweisung werden keine Privilegien geprüft.Eine SQL-Anweisung in einer Routine darf auf die Parameter der Routine und (wenn die Anweisung Teil einer COMPOUND-Anweisung ist) auf lokale Variablen, jedoch nicht auf Benutzervariablen zugreifen.

Syntax und Bedeutung von *routine_sql_anweisung* sind zentral im Abschnitt „SQL-Anweisungen in Routinen“ beschrieben. Die dort genannten SQL-Anweisungen dürfen verwendet werden.

Ausführungshinweise

Der *ausdruckx* der CASE-Anweisung wird berechnet.

Die WHEN-Klauseln werden von oben nach unten bewertet.

Die Ausdrücke *ausdruck1,...* der WHEN-Klauseln werden von links nach rechts berechnet.

Wenn ein derart berechneter Wert eines Ausdrucks mit dem Wert von *ausdruckx* übereinstimmt, dann wird der zugehörige THEN-Zweig ausgeführt und danach die CASE-Anweisung beendet.

Wenn keiner der berechneten Werte mit *ausdruckx* uebereinstimmt, aber ein ELSE-Zweig vorhanden ist, dann wird der ELSE-Zweig der CASE-Anweisung ausgeführt und danach die CASE-Anweisung beendet.

Wenn keiner der berechneten Werte mit *ausdruckx* uebereinstimmt und kein ELSE-Zweig vorhanden ist, dann wird die CASE-Anweisung mit SQLSTATE '20000' beendet.

Beispiel

Einfache CASE-Anweisung zur Berechnung des Feiertagszuschlages beim Lohn.

```
CASE MOD(JULIAN_DAY_OF_DATE(CURRENT_DATE),7)
  WHEN 0,1,2,3,4 /* heute ist normaler Werktag */
       THEN UPDATE lohntabelle SET lohn = zeit_lohn;
  WHEN 5 /* heute ist Samstag, 25% Aufschlag */
       THEN UPDATE lohntabelle SET lohn = zeit_lohn * 1.25;
  WHEN 6 /* heute ist Sonntag, 50% Aufschlag */
       THEN UPDATE lohntabelle SET lohn = zeit_lohn * 1.50;
END CASE
```

Obige CASE-Anweisung könnte auch als UPDATE-Anweisung mit einem entsprechenden *case_ausdruck* gestaltet werden.

```
UPDATE lohntabelle
   SET lohn = zeit_lohn * CASE MOD(JULIAN_DAY_OF_DATE(CURRENT_DATE),7)
                          WHEN 0,1,2,3,4 /* heute ist normaler Werktag */
                               THEN 1.00
                          WHEN 5 /* heute ist Samstag, 25% Aufschlag */
                               THEN 1.25
                          WHEN 6 /* heute ist Sonntag, 50% Aufschlag */
                               THEN 1.50
                          END
```

Format der CASE-Anweisung mit Suchbedingung

CASE WHEN *suchbedingung* THEN THEN *routine_sql_anweisung;* [*routine_sql_anweisung;*] ...

...

[ELSE *routine_sql_anweisung;* [*routine_sql_anweisung;*] ...]

END CASE

suchbedingung

Suchbedingung, deren Auswertung einen Wahrheitswert ergibt.
Wenn das Ergebnis der Suchbedingung „unbestimmt“ ist, dann wird in der THEN-Klausel keine SQL-Anweisung ausgeführt.

routine_sql_anweisung

Siehe „Format der einfachen CASE-Anweisung“.

Ausführungshinweise

Die WHEN-Klauseln werden von oben nach unten bewertet.

Die *suchbedingung* der WHEN-Klausel wird ausgewertet.

Wenn eine derart berechnete Suchbedingung den Wahrheitswert WAHR ergibt, dann wird der zugehörige THEN-Zweig ausgeführt und danach die CASE-Anweisung beendet.

Wenn keine der berechneten Suchbedingungen den Wahrheitswert WAHR ergibt, aber ein ELSE-Zweig vorhanden ist, dann wird der ELSE-Zweig der CASE-Anweisung ausgeführt und danach die CASE-Anweisung beendet.

Wenn keine der berechneten Suchbedingungen den Wahrheitswert WAHR ergibt und kein ELSE-Zweig vorhanden ist, dann wird die CASE-Anweisung mit SQLSTATE '20000' beendet.

Beispiel

CASE-Anweisung mit Suchbedingung.

```
CASE
   WHEN (EXISTS(select * from T1 where cola = 17))
        THEN update T1 set colb = colb * 1.05;
   WHEN (EXISTS(select * from T2 where colx = 27))
        THEN insert into T2 (pk, coly) values (*, 423);
END CASE
```

8.2.3.7 CLOSE - Cursor schließen

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

Die Cursorbeschreibung bleibt erhalten. Die aktuelle Cursorposition kann vor dem Schließen mit STORE gesichert werden (gilt nicht für lokale Cursor in Prozeduren).

Ein Cursor kann beliebig oft geschlossen und eventuell mit neuen Variablenwerten wieder geöffnet werden.

`CLOSE` *cursor*

---

cursor

Name des zu schließenden Cursors.

Beispiel
In dem folgenden Beispiel wird der Cursor CUR_KONTAKTE geschlossen.


CLOSE cur_kontakte

Siehe auch

DECLARE CURSOR, FETCH, OPEN, RESTORE, STORE

8.2.3.8 COMMIT WORK - Transaktion beenden

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

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

```
COMMIT [WORK]
```

SQL-Transaktion

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

Transaktion unter openUTM

Die Anweisung COMMIT WORK ist nicht zulässig, wenn Sie mit openUTM arbeiten. In diesem Fall wird die Transaktionssteuerung komplett mit UTM-Sprachmitteln durchgeführt. Die Synchronisation von SESAM/SQL- und UTM-Transaktionen wird durch openUTM gewährleistet. Eine UTM-Transaktion endet mit dem Setzen eines Sicherungspunktes.

Transaktion einleiten

Folgende SQL-Anweisungen leiten keine Transaktion ein:

- DECLARE CURSOR (nicht ausführbar)
- PERMIT
- SET CATALOG
- SET SCHEMA
- SET SESSION AUTHORIZATION
- SET TRANSACTION
- WHENEVER (nicht ausführbar)
- Utility-Anweisungen

Die Anweisungen EXECUTE und EXECUTE IMMEDIATE leiten nur dann eine SQL-Transaktion ein, wenn die jeweils auszuführende, dynamisch formulierte Anweisung eine Transaktion einleitet.

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

Anweisungen innerhalb von Transaktionen

Folgende Anweisungen dürfen nicht innerhalb einer Transaktion ausgeführt werden:

- SET SESSION AUTHORIZATION
- SET TRANSACTION
- Utility-Anweisungen

Eine SQL-Anweisung zur Datenmanipulation (abfragen, ändern) darf nicht innerhalb einer Transaktion ausgeführt oder vorbereitet werden, in der eine SQL-Anweisung zur Definition oder Verwaltung von Schemata, Speicherstrukturen oder Benutzereinträgen ausgeführt wird.

CALL-DML-Transaktion

Innerhalb einer CALL-DML-Transaktion ist die SQL-Anweisung COMMIT WORK nicht erlaubt (siehe Abschnitt „SQL-Anweisungen in CALL-DML-Transaktionen“).

Auswirkungen von COMMIT WORK

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

Auswirkungen auf nachfolgende Transaktionen

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

Auswirkungen auf Cursor (gilt nicht für lokale Cursor in Prozeduren)

COMMIT WORK schließt alle innerhalb der Transaktion geöffneten Cursor. Soll eine Cursorposition über das Transaktionsende hinaus gerettet werden, können Sie die Position mit der Anweisung STORE sichern und später mit RESTORE wiederherstellen.

Es ist möglich, einen Cursor mit der WITH HOLD-Klausel zu definieren. Ein solcher Cursor wird auch durch (erfolgreiches) COMMIT WORK nicht geschlossen. In einer nachfolgenden Transaktion kann er mit FETCH positioniert werden.

Auswirkungen auf Voreinstellungen

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

Verhalten von SESAM/SQL im Fehlerfall

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

Siehe auch

ROLLBACK WORK, SET TRANSACTION

8.2.3.9 COMPOUND - SQL-Anweisungen in gemeinsamem Kontext ausführen

Die COMPOUND-Anweisung führt weitere SQL-Anweisungen einer Routine in einem gemeinsamen Kontext aus. Für diese SQL-Anweisungen gelten gemeinsame lokale Daten (Variablen und Fehlernamen), gemeinsame lokale Cursor und gemeinsame lokale FehlerRoutinen.

> ¡ Die Schreibweise „COMPOUND“ (groß) wurde lediglich in Analogie zur bisherigen Notation der SQL-Anweisungen für diese SQL-Anweisung gewählt. Es gibt kein SQL-Schlüsselwort „COMPOUND“.

Die COMPOUND-Anweisung darf nur in einer Routine angegeben werden, d.h. im Rahmen einer CREATE PROCEDURE- oder CREATE FUNCTION-Anweisung. Sie ist dann die einzige Anweisung der Routine. Routinen und ihre Verwendung in SESAM/SQL sind detailliert im Kapitel „Routinen“ beschrieben.

```
[ marke :]
BEGIN [[NOT] ATOMIC]
[ lokale_daten ]
[ lokale_cursor ]
[ lokale_fehlerbehandlung ]
[ routine_sql_anweisung; [ routine_sql_anweisung; ]... ]
END [ marke ]
```

---

marke

Die Marke vor der COMPOUND-Anweisung (Anfangsmarke) bezeichnet den Anfang der COMPOUND-Anweisung. Sie darf nicht identisch sein mit einer anderen Marke innerhalb der COMPOUND-Anweisung.

Die Anfangsmarke muss nur dann angegeben werden, wenn die COMPOUND-Anweisung über eine LEAVE-Anweisung verlassen werden soll.

Die Marke am Ende der COMPOUND-Anweisung (Endemarke) bezeichnet das Ende der COMPOUND-Anweisung. Wenn die Endemarke angegeben ist, dann muss auch die Anfangsmarke angegeben sein. Beide Marken müssen identisch sein.

[NOT] ATOMIC

Bestimmt, ob die COMPOUND-Anweisung atomar oder nicht-atomar ist. Diese Angabe wirkt sich auf die lokale Fehlerbehandlung aus, siehe "COMPOUND - SQL-Anweisungen in gemeinsamem Kontext ausführen". Wenn keine Angabe gemacht wird, dann gilt: NOT ATOMIC.

lokale_daten

Definiert lokale Variablen und Fehlernamen für die COMPOUND-Anweisung, siehe Abschnitt „Lokale Daten“.

Die SQL-Anweisungen der COMPOUND-Anweisung können nur auf lokale Daten zugreifen, die in der COMPOUND-Anweisung definiert sind. Sie können nicht auf Benutzervariablen zugreifen.

lokale_cursor

Definiert lokale Cursor für die COMPOUND-Anweisung, siehe Abschnitt „Lokale Cursor“.

Die SQL-Anweisungen der COMPOUND-Anweisung können nur auf Cursor zugreifen, die in der COMPOUND-Anweisung definiert sind.

lokale_fehlerbehandlung

Definiert lokale Fehler-Routinen für die COMPOUND-Anweisung, siehe Abschnitt „Lokale Fehler-Routinen“.

SQLSTATEs der Klasse '40xxx' und SQLSTATEs ab Klasse '50xxx' können in den lokalen Fehler-Routinen nicht behandelt werden. Bei Auftreten eines derartigen SQL-STATE wird die Routine sofort abgebrochen. Bei einem SQLSTATE der Klasse '40xxx' wird zusätzlich die gesamte Transaktion zurückgesetzt.

SQLSTATEs, die in den Fehler-Routinen weder in Form einer Klasse noch explizit angeben wurden („unbehandelte SQLSTATEs“) werden von keiner Fehler-Routine behandelt. Das Gleiche gilt, wenn keine lokale Fehlerbehandlung definiert ist. In diesen Fällen führt SESAM/SQL automatisch die Fehlerbehandlung in folgender Weise aus:

- SQLSTATEs der Klassen '01xxx' (Warnung) oder '02xxx' (keine Daten) werden ignoriert, d.h. die Routine wird wie bei erfolgreicher Ausführung der SQL-Anweisung (SQLSTATE = '00000') fortgesetzt.
- Für SQLSTATEs, die nicht in den Klassen '01xxx', '02xxx' oder '40xxx' liegen, werden folgende Aktionen ausgeführt:
  - Offene lokale Cursor werden geschlossen.
  - Bei Angabe von ATOMIC in der COMPOUND-Anweisung werden alle Änderungen rückgängig gemacht, die im Rahmen der COMPOUND-Anweisung durchgeführt wurden.
  - Bei Angabe von NOT ATOMIC (Standardwert) in der COMPOUND-Anweisung werden nur die Änderungen rückgängig gemacht, die im Rahmen der fehlerhaften SQL-Anweisung durchgeführt worden sind.
  - Die COMPOUND-Anweisung und damit die Routine wird abgebrochen. Die SQL-Anweisung, in der die Routine verwendet wurde, liefert den betreffenden SQLSTATE zurück.

routine_sql_anweisung

SQL-Anweisung, die in der COMPOUND-Anweisung ausgeführt werden soll.
Eine SQL-Anweisung wird mit „;“ (Semikolon) abgeschlossen.
Mehrere SQL-Anweisungen können nacheinander angegeben werden. Sie werden in der angegebenen Reihenfolge ausgeführt.
In der COMPOUND-Anweisung darf keine (weitere) COMPOUND-Anweisung angegeben werden. D.h., es sind keine geschachtelten COMPOUND-Anweisungen erlaubt (Ausnahme: lokale Fehler-Routinen, siehe "COMPOUND - SQL-Anweisungen in gemeinsamem Kontext ausführen".)

Vor der Durchführung einer SQL-Anweisung werden keine Privilegien geprüft.
Eine SQL-Anweisung in einer Routine darf auf die Parameter der Routine und (wenn die Anweisung Teil einer COMPOUND-Anweisung ist) auf lokale Variablen, jedoch nicht auf Benutzervariablen zugreifen.

Syntax und Bedeutung von *routine_sql_anweisung* sind zentral im Abschnitt „SQL-Anweisungen in Routinen“ beschrieben. Die dort genannten SQL-Anweisungen dürfen verwendet werden.

Beispiel

Beispiele finden Sie im Kapitel „Routinen“ und in der Beispieldatenbank von SESAM/SQL (siehe „ Basishandbuch“).

Siehe auch

CREATE PROCEDURE, CREATE FUNCTION, CALL, DROP PROCEDURE, DROP FUNCTION, CASE, FOR, IF, ITERATE, LEAVE, LOOP, REPEAT, SET, WHILE, RETURN, GET DIAGNOSTICS, SIGNAL, RESIGNAL, SELECT, INSERT, UPDATE, DELETE, MERGE, OPEN, FETCH, UPDATE, DELETE, CLOSE

Lokale Daten

Lokale Daten sind Variablen oder Fehlernamen, die nur innerhalb der COMPOUND-Anweisung angesprochen werden können.

Für Variablen wird ein Datentyp und ggf. ein Standardwert definiert. Sie besitzen keine Indikatorvariable. Sie können in lokalen Cursor-Definitionen, lokalen Fehler-Routinen und den SQL-Anweisungen der COMPOUND-Anweisung verwendet werden.

> ¡ *Empfehlung* Die Namen von lokalen Variablen sollten sich (z.B. durch Vergabe eines Präfixes wie `par_`) von Spaltennamen unterscheiden.

Fehlernamen definieren zum leichteren Verständnis einen Namen für einen Fehler (ohne Angabe eines zugehörigen SQLSTATE) oder einen Namen für einen SQLSTATE. Sie können in lokalen Fehler-Routinen verwendet werden, siehe "COMPOUND - SQL-Anweisungen in gemeinsamem Kontext ausführen".

---

lokale_daten `::= DECLARE` *deklaration* `; [DECLARE` *deklaration* `;] ...`

deklaration `::=`
`{`

lokale_variable `[,` *lokale_variable* `], ...` *datentyp* `[` *voreinstellung* `] |`
fehlername `CONDITION [FOR` *sqlstate* `];`

`}`

sqlstate `::= SQLSTATE [VALUE]` *alphanumerisches_literal*

Mehrere Variablen des gleichen Datentyps mit dem gleichen SQL-Defaultwert können nacheinander, getrennt durch „,“ (Komma), angegeben werden.
Die Definition eines lokalen Datums wird mit „;“ (Semikolon) abgeschlossen. Mehrere Definitionen können nacheinander angegeben werden.

lokale_variable

Name der lokalen Variablen.
Die Namen aller lokalen Variablen müssen sich voneinander, von den lokalen Fehlernamen und von den Namen der Parameter der Routine unterscheiden.

datentyp

Datentyp der lokalen Variablen.
Es gibt nur einfache lokale Variablen. *dimension* darf nicht angegeben werden.

voreinstellung

Legt den SQL-Defaultwert für die lokale Variable fest.
Es gelten die Zuweisungsregeln für Defaultwerte, siehe Abschnitt „Defaultwerte für Tabellenspalten“).

fehlername

Name für einen Fehler oder einen SQLSTATE.
Alle Fehlernamen müssen sich voneinander, von den lokalen Variablen und von den Namen der Parameter der Routine unterscheiden.

FOR *sqlstate*

SQLSTATE (alphanumerisches Literal der Länge 5), der durch *fehlername* bezeichnet wird.
Die Einschränkungen für die Menge der SQLSTATEs ist zu beachten, siehe „Lokale Fehler-Routinen “.

FOR *sqlstate* nicht angegeben

Lokale Fehlernamen ohne FOR-Klausel können nur durch eine SIGNAL- oder RESIGNAL-Anweisung ausgelöst werden. Sie werden auf den SQLSTATE '45000' (unbehandelter Anwenderfehler) abgebildet und an das Benutzerprogramm gemeldet. *fehlername* erscheint dabei als Insert der Fehlermeldung.

Beispiel

Definition lokaler Variablen:

```
DECLARE a,b,c SMALLINT DEFAULT 0;

DECLARE mytim TIME(3) DEFAULT CURRENT_TIME;
```

Definition von Fehlernamen:

```
DECLARE Tab_not_accessible CONDITION FOR SQLSTATE '42SQK';
DECLARE "CHECK problem" CONDITION FOR SQLSTATE '23SA1';
DECLARE "Unknown problem" CONDITION;
```

Lokale Cursor

Mit der Definition lokaler Cursor werden Cursor festgelegt, die nur innerhalb der COMPOUND-Anweisung angesprochen werden können.
Die Namen der lokalen Cursor müssen sich voneinander unterscheiden.

Lokale Cursor können in lokalen Fehler-Routinen und den SQL-Anweisungen der COMPOUND-Anweisung verwendet werden.

Die SQL-Anweisungen STORE und RESTORE sind bei lokalen Cursorn nicht erlaubt.

---

lokale_cursor `::= { ` *declare_cursor_anweisung* `; } ...`

Eine Cursor-Definition wird mit „;“ (Semikolon) abgeschlossen.

Mehrere Cursor-Definitionen können nacheinander angegeben werden.

declare_cursor_anweisung

DECLARE CURSOR-Anweisung (siehe Abschnitt „DECLARE CURSOR - Cursor vereinbaren“), mit der der lokale Cursor definiert wird. Die Klausel WITH HOLD darf nicht angegeben werden.
Ein lokaler Cursor unterscheidet sich von einem gewöhnlichen Cursor nur durch den beschränkten Gültigkeitsbereich.

Beispiel

Siehe Abschnitt „DECLARE CURSOR - Cursor vereinbaren“.

Lokale Fehler-Routinen

---

lokale_fehlerbehandlung `::=` *fehler_routine* `; [` *fehler_routine* `;]...`

fehler_routine `::= DECLARE { CONTINUE | EXIT | UNDO } HANDLER FOR`
fehlerliste `{` *routine_sql_anweisung* `|` *compound_anweisung* `}`

fehlerliste `::= {` *klassenliste* `|` *sqlstate_oder_fehlerliste* `}`

klassenliste `::= { SQLEXCEPTION | SQLWARNING | NOT FOUND }`
`[{ SQLEXCEPTION | SQLWARNING | NOT FOUND }] ...]`

sqlstate_oder_fehlerliste `::= {` *sqlstate* `|` *fehlername* `}`
`[,{` *sqlstate* `|` *alphanumerisches_literal* `|` *fehlername* `},...]`

sqlstate `::= SQLSTATE [VALUE]` *alphanumerisches_literal*

Mit der Definition lokaler Fehler-Routinen wird festgelegt, wie reagiert werden soll, wenn bei der Verarbeitung einer SQL-Anweisung im Rahmen der COMPOUND-Anweisung ein SQL-STATE '00000' gemeldet wird.

Es können die SQLSTATEs der Klassen '0xxxx' (mit Ausnahme des SQLSTATE = '00000'), '1xxxx', '2xxxx', '3xxxx' und '4xxxx' (mit Ausnahme der Klasse '40xxx') behandelt werden.

Fehler-Routinen werden mit „;“ (Semikolon) abgeschlossen.
Mehrere Fehler-Routinen können nacheinander angegeben werden.

Wenn ein SQLSTATE '45000' auftritt (definiert in der *klassenliste* oder als *sqlstate* oder *fehlername*), dann wird die Fehler-Routine für den angegebenen SQLSTATE ausgeführt.

Tritt, ausgelöst durch eine SIGNAL- oder RESIGNAL-Anweisung, der SQLSTATE '45000' (unbehandelte Anwender-Bedingung) auf, dann wird der *fehlername* der Fehler-Information ausgewertet und die entsprechende Fehler-Routine ausgeführt.

DECLARE

Art der Fehlerbehandlung in Abhängigkeit vom SQLSTATE. Siehe auch den Abschnitt „Erfolg einer SQL-Anweisung in einer Routine“.

CONTINUE

- Die Änderungen, die im Rahmen der fehlerhaften SQL-Anweisung durchgeführt worden sind, werden rückgängig gemacht.
- Die SQL-Anweisung der Fehler-Routine wird ausgeführt.
- Wenn diese SQL-Anweisung erfolglos beendet wurde, dann wird die Routine abgebrochen und dieser SQLSTATE wird an den Anwender zurückgemeldet.
- Wenn diese SQL-Anweisung fehlerfrei beendet wurde, dann wird die Routine fortgesetzt. Die SQL-Anweisung, die den SQLSTATE gemeldet hat und damit die Fehlerbehandlung auslöste, wird als erfolgreich angesehen.

EXIT

- Die Änderungen, die im Rahmen der fehlerhaften SQL-Anweisung durchgeführt worden sind, werden rückgängig gemacht.
- Die SQL-Anweisung der Fehler-Routine wird ausgeführt.
  Offene lokale Cursor werden geschlossen.
- Wenn diese SQL-Anweisung erfolglos beendet wurde, dann wird die Routine abgebrochen und dieser SQLSTATE wird an den Anwender zurückgemeldet.
- Wenn diese SQL-Anweisung fehlerfrei beendet wurde, dann wird die Routine beendet. Die COMPOUND-Anweisung wird als erfolgreich angesehen (SQL-STATE = '00000' wird zurückgemeldet).

UNDO (nur erlaubt bei Angabe von ATOMIC in der COMPOUND-Anweisung)

- Alle Änderungen, die im Rahmen der COMPOUND-Anweisung durchgeführt wurden, werden rückgängig gemacht.
- Die SQL-Anweisung der Fehler-Routine wird ausgeführt.
  Offene lokale Cursor werden geschlossen.
- Wenn diese SQL-Anweisung erfolglos beendet wurde, dann wird die Routine abgebrochen und dieser SQLSTATE wird an den Anwender zurückgemeldet.
- Wenn diese SQL-Anweisung fehlerfrei beendet wurde, dann wird die Routine beendet. Die COMPOUND-Anweisung wird als erfolgreich angesehen (SQL-STATE = '00000' wird zurückgemeldet).

klassenliste

Angabe von SQLSTATE-Mengen:

- SQLWARNING bezeichnet die SQLSTATEs der Klasse 01xxx (Warnung).
- NOT FOUND bezeichnet die SQLSTATEs der Klasse 02xxx (keine Daten).
- SQLEXCEPTION bezeichnet alle sonstigen SQLSTATEs der Klassen 0xxxxx bis 4xxxx (mit Ausnahme des SQLSTATEs '00000' und der Klasse 40xxx), die im Rahmen einer Fehler-Routine behandelt werden können.

sqlstate

Explizite Angabe von SQLSTATEs.
Jedes alphanumerische Literal muss einen SQLSTATE in 5 Zeichen (Ziffern oder Großbuchstaben) darstellen.
Es dürfen nur SQLSTATEs angegeben werden, die im Rahmen einer Fehler-Routine behandelt werden können.

¡ Jeder SQLSTATE darf nur einmal in einer der Fehler-Routinen der COM-POUND-Anweisung auftreten.

Folgende Auflistung zeigt Beispiele für SQLSTATEs, für die eine eigene Fehlerbehandlung sinnvoll sein kann (siehe Handbuch „ Meldungen“):

01004	Zeichenkette wurde rechts verkürzt
20000	CASE-Anweisung ohne Treffer enthält keine ELSE-Klausel
21000	Ergebnistabelle enthält mehr als 1 Zeile
22001	Zeichenkette wurde rechts verkürzt
22003	Numerischer Wert zu groß oder zu klein
22SA1	Kommastellen abgeschnitten oder gerundet
23SA0	Referenzbedingung verletzt
23SA1	CHECK-Bedingung verletzt
23SA2	Eindeutigkeitsbedingung verletzt
23SA3	NOT-NULL-Bedingung verletzt
23SA4	NOT-NULL-Bedingung des Primärschlüssels verletzt
23SA5	Eindeutigkeitsbedingung des Primärschlüssels verletzt
24SA1	Cursor ist nicht geschlossen

24SA2	Cursor ist nicht geöffnet
24SA3	Cursor ist nicht auf einem Satz positioniert

Andere SQLSTATEs, z.B. Syntaxfehler, werden am besten über eine der zuvor beschriebenen Mengen von SQLSTATEs behandelt.

fehlername

Name für einen Fehler oder einen SQLSTATE, siehe „Lokale Daten“.
Es dürfen nur solche Fehlernamen angegeben werden, die im Rahmen einer Fehler-Routine behandelt werden können.

¡ Jeder *fehlername* darf nur einmal in einer der Fehler-Routinen der COMPOUND-Anweisung auftreten.

routine_sql_anweisung

SQL-Anweisung, die in der Fehler-Routine ausgeführt werden soll.

Syntax und Bedeutung von *routine_sql_anweisung* sind zentral im Abschnitt „SQL-Anweisungen in Routinen“ beschrieben. Die dort genannten SQL-Anweisungen dürfen verwendet werden.

compound_anweisung

COMPOUND-Anweisung, die mehrere SQL-Anweisungen enthält , siehe Abschnitt „COMPOUND - SQL-Anweisungen in gemeinsamem Kontext ausführen“.
Eine hier angegebene COMPOUND-Anweisung darf außer den zulässigen *routine_sql_anweisungen* keine Definitionen lokaler Daten, Cursor oder Fehler-Routinen enthalten.

Beispiel

Definition einer einfachen Fehlerbehandlung mit zwei Fehler-Routinen.

```
DECLARE CONTINUE HANDLER FOR SQLWARNING,NOT FOUND
   SET eot=1;
DECLARE EXIT HANDLER FOR SQLSTATE '23SA0'
   BEGIN END;
```

8.2.3.10 CREATE FUNCTION - User Defined Function (UDF) erzeugen

CREATE FUNCTION erzeugt eine UDF und speichert ihre Definition in der Datenbank.

UDFs und ihre Verwendung in SESAM/SQL sind detailliert im Kapitel „Routinen“ beschrieben.

Jede Routine, die in der UDF aufgerufen wird, muss bereits existieren. Geschachtelte Aufrufe von Routinen sind damit möglich, rekursive Aufrufe aber nicht.

Der aktuelle Berechtigungsschlüssel muss Eigentümer des Schemas sein, zu dem die UDF gehört. Er muss auch für alle Tabellen und Spalten, die in der UDF angesprochen werden, diejenigen Privilegien besitzen, die benötigt werden, um die in der UDF enthaltenen DML-Anweisungen ausführen zu können.

Der aktuelle Berechtigungsschlüssel muss das EXECUTE-Privileg für die in der UDF direkt aufgerufenen Routine besitzen. Zusätzlich muss er für alle Tabellen und Spalten, die in der UDF angesprochen werden, diejenigen Privilegien besitzen, die benötigt werden, um die in der Routine enthaltenen DML-Anweisungen ausführen zu können.

Der aktuelle Berechtigungsschlüssel erhält automatisch das EXECUTE-Privileg für die erzeugte UDF. Hat er für die betreffenden Privilegien sogar die Berechtigung, diese weitergeben zu dürfen, dann darf er auch das EXECUTE-Privileg an andere Berechtigungsschlüssel weitergegeben.

Die UDF und die Objekte, die in der UDF angesprochen werden, müssen zur gleichen Datenbank gehören. Die Namen dieser Objekte werden gegebenenfalls durch den Datenbank und Schemanamen der UDF ergänzt.

```
CREATE FUNCTION
  udf ([ udf_parameter_definition [, udf_parameter_definition ]...])
  RETURNS datentyp
  { READS SQL DATA | CONTAINS SQL }
  { routine_sql_anweisung | compound_anweisung }

udf ::= routine
udf_parameter_definition ::= [IN] routinenparameter datentyp
```

---

udf

Name für die UDF (maximale Länge: 31 Zeichen). Der einfache Name der UDF muss innerhalb der Routinen-Namen des Schemas eindeutig sein. Er kann durch einen Datenbank- und Schemanamen qualifiziert werden. Wenn die CREATE FUNCTION-Anweisung innerhalb einer CREATE SCHEMA-Anweisung angegeben wird, dann darf der UDF-Name nur mit den Datenbank- und Schemanamen der CREATE SCHEMA-Anweisung qualifiziert werden.

```
([ udf_parameter_definition [{, udf_parameter_definition }...]])
```

Liste der UDF-Aufrufparameter. Die Anzahl der UDF-Aufrufparameter ist beliebig. Sie wird nur von der maximalen Anweisungslänge begrenzt. Wenn kein Parameter definiert wird, dann besteht die Liste nur aus den runden Klammern.

udf_parameter_definition

Definition eines UDF-Aufrufparameters.
UDF-Aufrufparameter besitzen keine Indikatorvariable.

routinenparameter

Name des UDF-Aufrufparameters. Die Namen der UDF-Aufrufparameter müssen sich voneinander unterscheiden.

datentyp

Datentyp des UDF-Aufrufparameters.
Es sind nur einfache UDF-Aufrufparameter erlaubt.
dimension darf nicht angegeben werden.

RETURNS *datentyp*

Datentyp des UDF-Rückgabewertes.
Es sind nur einfache UDF-Rückgabewerte erlaubt.
dimension darf nicht angegeben werden.

READS SQL DATA

Die UDF kann SQL-Anweisungen zum Lesen von Daten enthalten, jedoch keine SQL-Anweisungen zum Ändern von Daten. Diese Aussage wird geprüft. Im Fehlerfall wird die Anweisung mit SQLSTATE abgewiesen.

¡ Aufgerufene Routinen dieser UDF dürfen die Angabe MODIFIES SQL DATA **nicht** enthalten.

CONTAINS SQL

Die UDF enthält weder SQL-Anweisungen zum Lesen von Daten noch zum Ändern von Daten. Diese Aussage wird geprüft. Im Fehlerfall wird die Anweisung mit SQLSTATE abgewiesen.

¡ UDFs enthalten stets SQL-Anweisungen, d.h. CONTAINS SQL liegt stets vor. Der in der SQL-Norm vorgesehene Fall NO SQL tritt nicht auf.
Aufgerufene Routinen dieser Prozedur dürfen die Angaben MODIFIES SQL DATA und READS SQL DATA **nicht** enthalten.

routine_sql_anweisung

Eine UDF enthält genau eine nicht-atomare SQL-Anweisung oder genau eine RETURN-Anweisung. Die nicht-atomare SQL-Anweisung muss mindestens eine RETURN-Anweisung enthalten.Nicht-atomare SQL-Anweisungen in SESAM/SQL sind COMPOUND (ohne Angabe von ATOMIC), CASE, FOR, IF, LOOP, REPEAT und WHILE. Sie können weitere atomare oder nicht-atomare SQL-Anweisungen enthalten. Atomare SQL-Anweisungen sind die weiteren, in einer Routine zulässigen SQL-Anweisungen.

Vor der Durchführung einer SQL-Anweisung werden keine Privilegien geprüft.Eine SQL-Anweisung in einer UDF darf auf die Parameter der UDF und (wenn die Anweisung Teil einer COMPOUND-Anweisung ist) auf lokale Variablen, jedoch nicht auf Benutzervariablen zugreifen.

Syntax und Bedeutung von *routine_sql_anweisung* sind zentral im Abschnitt „SQL-Anweisungen in Routinen“ beschrieben. Die dort genannten SQL-Anweisungen mit Ausnahme der SQL-Anweisungen zum Ändern von Daten (INSERT, UP-DATE, MERGE, DELETE) dürfen verwendet werden.

compound_anweisung

COMPOUND-Anweisung, die mehrere SQL-Anweisungen enthält und dafür ggf. gemeinsame lokale Daten, Cursor und Fehler-Routinen definiert, siehe Abschnitt „COM-POUND - SQL-Anweisungen in gemeinsamem Kontext ausführen“.

Randbedingungen

SESAM/SQL bietet zur Steuerung von Routinen, die SQL-Anweisungen COMPOUND, CASE, FOR, IF, ITERATE, LEAVE, LOOP, REPEAT, SET und WHILE an. Diese SQL-Anweisungen werden auch als Kontrollanweisungen bezeichnet.

Diagnoseinformationen erhalten Sie in Routinen mit den Diagnoseanweisungen GET DIAGNOSTICS, SIGNAL und RESIGNAL.

In SESAM/SQL sind geschachtelte Aufrufe von Routinen erlaubt. Die CALL-Anweisung gehört deshalb zu den in einer Routine erlaubten Anweisungen.

Eine Routine darf keine SQL-Anweisungen zur Transaktionsverwaltung (siehe "SQL-Anweisungen zur Transaktionsverwaltung") enthalten. Auf lokale Cursor kann deshalb nicht transaktionsübergreifend zugegriffen werden. STORE- oder RESTORE-Anweisungen gehören nicht zu den in einer Routine erlaubten Anweisungen; sie sind in einer Routine nicht sinnvoll.

Eine Routine darf keine dynamisch formulierten SQL-Anweisungen oder Cursorbeschreibungen enthalten, siehe Abschnitt „Dynamische SQL“.

Eine Routine kann in einer UDF in einer dynamisch formulierten SQL-Anweisung aufgerufen werden. Wenn eine Prozedur Parameter vom Typ OUT oder INOUT enthält, dann müssen in einer dynamisch formulierten CALL-Anweisung die korrespondierenden Argumente in Form von Platzhaltern angegeben werden.

Beispiel

Die folgende UDF `GetCurrentYear` liefert das aktuelle Jahr als Zahl. Sie enthält keine SQL-Anweisungen zum Lesen oder Ändern von Daten.

```
CREATE FUNCTION GetCurrentYear (IN "TIME" TIMESTAMP(3))
   RETURNS DECIMAL(4)
```

```
CONTAINS SQL
RETURN EXTRACT (YEAR FROM "TIME")
```

Weitere Beispiele finden Sie im Kapitel „Routinen“ und in der Beispieldatenbank von SESAM/SQL (siehe „ Basishandbuch“).

Siehe auch

DROP FUNCTION, COMPOUND, CASE, FOR, IF, ITERATE, LEAVE, LOOP, REPEAT, SET, WHILE, CALL, RETURN, SELECT, INSERT, OPEN, FETCH, CLOSE, GET DIAGNOSTICS, SIGNAL, RESIGNAL

8.2.3.11 CREATE INDEX - Index erzeugen

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

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

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

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

`CREATE INDEX` *indexdefinition* `,...ON TABLE` *tabelle* `[USING SPACE` *space* `]`

indexdefinition `::=` *index* `({` *spalte* `[LENGTH` *länge* `]},...)`

---

indexdefinition

Definition eines oder mehrerer Indizes.

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

index

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

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

spalte

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

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

LENGTH *länge*

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

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

ON TABLE *tabelle*

Name der Basistabelle, für die ein Index definiert wird.

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

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

USING SPACE *space*

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

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

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

USING SPACE *space* nicht angegeben:
Der Index wird im Space der Basistabelle gespeichert. Bei einer partitionierten Tabelle wird der Index auf dem Space der ersten Partition gespeichert.

Besonderheiten für CALL-DML-Tabellen

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

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

Index und Integritätsbedingung

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

Beispiele

Das folgende Beispiel erzeugt einen zusammengesetzten Index für die Spalten KNR und FIRMA der Tabelle KUNDE. Die Spalte FIRMA wird dabei bis zu einer Länge von 10 Zeichen in den Index einbezogen. Der Index wird auf dem Space INDEXSPACE gespeichert.

```
CREATE INDEX kund_ind (knr,firma LENGTH 10)
ON TABLE kunde USING SPACE indexspace
```

Mit der CREATE INDEX-Anweisung wird der Index NAT_KUND_IND für die Spalten NAT_KNR und NAT_FIRMA der Tabelle NAT_KUNDE definiert. Die Spalte NAT_FIRMA hat den National-Datentyp NCHAR. Werte der Spalte NAT_FIRMA werden bis zu einer Länge von 5 Zeichen bei der Indexbildung berücksichtigt (1 Zeichen = 2 Byte). Der Index soll auf dem Space NAT_INDEXSPACE angelegt werden.

```
CREATE INDEX nat_kund_ind(nat_knr, nat_firma LENGTH 10)
ON TABLE nat_kunde USING SPACE nat_indexspace
```

Siehe auch

DROP INDEX

8.2.3.12 CREATE PROCEDURE - Prozedur erzeugen

CREATE PROCEDURE erzeugt eine Prozedur und speichert ihre Definition in der Datenbank..

Prozeduren und ihre Verwendung in SESAM/SQL sind detailliert im Kapitel „Routinen“ beschrieben.

Jede Routine, die in der Prozedur aufgerufen wird, muss bereits existieren. Geschachtelte Aufrufe von Routinen sind damit möglich, rekursive Aufrufe aber nicht.

Der aktuelle Berechtigungsschlüssel muss Eigentümer des Schemas sein, zu dem die Prozedur gehört. Er muss auch für alle Tabellen und Spalten, die in der Prozedur angesprochen werden, diejenigen Privilegien besitzen, die benötigt werden, um die in der Prozedur enthaltenen DML-Anweisungen ausführen zu können.

Der aktuelle Berechtigungsschlüssel muss das EXECUTE-Privileg für jede in der Prozedur aufgerufene Routine besitzen. Zusätzlich muss er für alle Tabellen und Spalten, die in der Prozedur angesprochen werden, diejenigen Privilegien besitzen, die benötigt werden, um die in der Prozedur enthaltenen DML-Anweisungen ausführen zu können.

Der aktuelle Berechtigungsschlüssel erhält automatisch das EXECUTE-Privileg für die erzeugte Prozedur. Hat er für die betreffenden Privilegien sogar die Berechtigung, diese weitergeben zu dürfen, dann darf er auch das EXECUTE-Privileg an andere Berechtigungsschlüssel weitergegeben.

Die Prozedur und die Objekte, die in der Prozedur angesprochen werden, müssen zur gleichen Datenbank gehören. Die Namen dieser Objekte werden gegebenenfalls durch den Datenbank- und Schemanamen der Prozedur ergänzt.

```
CREATE PROCEDURE prozedur ([ prozedurparameter_definition [, prozedurparameter_definition ] ...])

{ MODIFIES SQL DATA | READS SQL DATA | CONTAINS SQL }

{ routine_sql_anweisung | compound_anweisung }

prozedur ::= routine

prozedurparameter_definition ::= [IN | OUT | INOUT] routinenparameter datentyp
```

---

prozedur

Name für die Prozedur (maximale Länge: 31 Zeichen). Der einfache Prozedurname muss innerhalb der Routinen-Namen des Schemas eindeutig sein. Er kann durch einen Datenbank- und Schemanamen qualifiziert werden.
Wenn die CREATE PROCEDURE-Anweisung innerhalb einer CREATE SCHEMA-Anweisung angegeben wird, dann darf der Prozedurname nur mit den Datenbank- und Schemanamen der CREATE SCHEMA-Anweisung qualifiziert werden.

([*prozedurparameter_definition* [{, *prozedurparameter_definition* }...]])

Liste der Prozedurparameter. Die Anzahl der Prozedurparameter ist beliebig. Sie wird nur von der maximalen Anweisungslänge begrenzt. Wenn kein Parameter definiert wird, dann besteht die Liste nur aus den runden Klammern.

prozedurparameter_definition

Definition eines Prozedurparameters.
Prozedurparameter besitzen keine Indikatorvariable.

IN: Der Prozedurparameter ist ein Eingabeparameter.

OUT: Der Prozedurparameter ist ein Ausgabeparameter.

INOUT: Der Prozedurparameter ist Ein- und Ausgabeparameter.

routinenparameter

Name des Prozedurparameters. Die Namen der Prozedurparameter müssen sich voneinander unterscheiden.

datentyp

Datentyp des Prozedurparameters.
Es sind nur einfache Prozedurparameter erlaubt.
dimension darf nicht angegeben werden.

MODIFIES SQL DATA

Die Prozedur kann SQL-Anweisungen zum Ändern von Daten enthalten.

READS SQL DATA

Die Prozedur kann SQL-Anweisungen zum Lesen von Daten enthalten, jedoch keine SQL-Anweisungen zum Ändern von Daten. Diese Aussage wird geprüft. Im Fehlerfall wird die Anweisung mit SQLSTATE abgewiesen.

> ¡ Aufgerufene Routinen dieser Prozedur dürfen die Angaben MODIFIES SQL DATA **nicht** enthalten.

CONTAINS SQL

Die Prozedur enthält weder SQL-Anweisungen zum Lesen von Daten noch zum Ändern von Daten. Diese Aussage wird geprüft. Im Fehlerfall wird die Anweisung mit SQLSTATE abgewiesen.

> ¡ Prozeduren enthalten stets SQL-Anweisungen, d.h. CONTAINS SQL liegt stets vor. Der in der SQL-Norm vorgesehene Fall NO SQL tritt nicht auf.
> Aufgerufene Routinen dieser Prozedur dürfen die Angaben MODIFIES SQL DATA und READS SQL DATA **nicht** enthalten.

routine_sql_anweisung

Eine Prozedur enthält genau eine atomare oder nicht-atomare SQL-Anweisung. Nicht-atomare SQL-Anweisungen in SESAM/SQL sind COMPOUND (ohne Angabe von ATOMIC), CASE, FOR, IF, LOOP, REPEAT und WHILE. Sie können weitere atomare oder nicht-atomare SQL-Anweisungen enthalten. Atomare SQL-Anweisungen sind die weiteren, in einer Routine zulässigen SQL-Anweisungen.

Vor der Durchführung einer SQL-Anweisung werden keine Privilegien geprüft. Eine SQL-Anweisung in einer Prozedur darf auf die Parameter der Prozedur und (wenn die Anweisung Teil einer COMPOUND-Anweisung ist) auf lokale Variablen, jedoch nicht auf Benutzervariablen zugreifen.

Syntax und Bedeutung von *routine_sql_anweisung* sind zentral im Abschnitt „SQL-Anweisungen in Routinen“ beschrieben. Die dort genannten SQL-Anweisungen mit Ausnahme von RETURN dürfen verwendet werden.

compound_anweisung

COMPOUND-Anweisung, die mehrere SQL-Anweisungen enthält und dafür ggf. gemeinsame lokale Daten, Cursor und Fehler-Routinen definiert, siehe Abschnitt „COMPOUND - SQL-Anweisungen in gemeinsamem Kontext ausführen“.

Randbedingungen

SESAM/SQL bietet zur Steuerung von Routinen die SQL-Anweisungen COMPOUND, CASE, FOR, IF, ITERATE, LEAVE, LOOP, REPEAT, SET und WHILE an. Diese SQL-Anweisungen werden auch als Kontrollanweisungen bezeichnet.

Diagnoseinformationen erhalten Sie in Routinen mit den Diagnoseanweisungen GET DIAGNOSTICS, SIGNAL und RESIGNAL.

In SESAM/SQL sind geschachtelte Aufrufe von Routinen erlaubt. Die CALL-Anweisung gehört deshalb zu den in einer Routine erlaubten Anweisungen.

Eine Routine darf keine SQL-Anweisungen zur Transaktionsverwaltung (siehe "SQL-Anweisungen zur Transaktionsverwaltung") enthalten. Auf lokale Cursor kann deshalb nicht transaktionsübergreifend zugegriffen werden. STORE- oder RESTORE-Anweisungen gehören nicht zu den in einer Routine erlaubten Anweisungen; sie sind in einer Routine nicht sinnvoll.

Eine Routine darf keine dynamisch formulierten SQL-Anweisungen oder Cursorbeschreibungen enthalten, siehe Abschnitt „Dynamische SQL“.

Eine Routine kann in einer dynamisch formulierten SQL-Anweisung aufgerufen werden.
Wenn eine Prozedur Parameter vom Typ OUT oder INOUT enthält, dann müssen in einer dynamisch formulierten CALL-Anweisung die korrespondierenden Argumente in Form von Platzhaltern angegeben werden.

Beispiel

Die folgende Prozedur `GetCurrentYear` liefert das aktuelle Jahr als Zahl. Sie enthält keine SQL-Anweisungen zum Lesen oder Ändern von Daten.

```
CREATE PROCEDURE ProcSchema.GetCurrentYear (OUT current_year INTEGER)
   CONTAINS SQL
   SET current_year = EXTRACT (YEAR FROM CURRENT_DATE)
```

Weitere Beispiele finden Sie im Kapitel „Routinen“ und in der Beispieldatenbankvon SESAM/SQL (siehe „ Basishandbuch“).

Siehe auch

CALL, DROP PROCEDURE, COMPOUND, CASE, FOR, IF, ITERATE, LEAVE, LOOP, REPEAT, SET, WHILE, SELECT, INSERT, UPDATE, DELETE, MERGE, OPEN, FETCH, UPDATE, DELETE, CLOSE, GET DIAGNOSTICS, SIGNAL, RESIGNAL

8.2.3.13 CREATE SCHEMA - Schema erzeugen

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

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

```
CREATE SCHEMA
{ schema [AUTHORIZATION berechtigungsschlüssel ] |  AUTHORIZATION berechtigungsschlüssel }
[ create_table_anweisung |
  create_view_anweisung |
  create_function_anweisung |
  create_procedure_anweisung |
  grant_anweisung |
  create_index_anweisung ] ...
```

schema

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

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

AUTHORIZATION *berechtigungsschlüssel*

Der Berechtigungsschlüssel wird Eigentümer des Schemas.

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

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

create / grant_anweisungen

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

create_table_anweisung

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

create_view_anweisung

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

create_function_anweisung

CREATE FUNCTION-Anweisung, die eine UDF für das Schema erzeugt.

create_procedure_anweisung

CREATE PROCEDURE-Anweisung, die eine Prozedur für das Schema erzeugt.

grant_anweisung

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

create_index_anweisung

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

create / grant_anweisungen nicht angegeben:

Es wird ein leeres Schema eingerichtet.

Arbeitsweise von CREATE SCHEMA

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

Beispiel

Das Beispiel erzeugt das Schema ZUSAETZE und legt die Tabelle BILDER an. Die Privilegien für die Tabelle BILDER werden an den Berechtigungsschlüssel utianw1 vergeben.

```
CREATE SCHEMA zusaetze
CREATE TABLE bilder OF BLOB
(
 MIME ('image / gif'),
 USAGE ('Abbildungen fuer teile.katart.abb'),
 '<Fotograf>Hans Sesamer</Fotograf>'
```

```
) USING SPACE blobspace
GRANT ALL PRIVILEGES ON bilder TO utianw1
```

Siehe auch

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

8.2.3.14 CREATE SPACE - Space erzeugen

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

Sie können max. 999 Anwender-Spaces pro Datenbank definieren.
Ein Anwender-Space kann auf Pubsets mit „großen Dateien“ bis zu 4 TByte groß werden. Sonst kann er bis zu 64 GByte groß werden.

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

Liegt der Catalog-Space der Datenbank in einer DB-Kennung, so müssen Vorbereitungen getroffen worden sein, siehe Abschnitt „Datenbankdateien und Jobvariablen auf fremden Benutzerkennungen“ im „ Basishandbuch“.

Wurde die Datei des Catalog-Space mit Kennwort angelegt, so ist auch für die Dateien der Anwender-Spaces ein Kennwort erforderlich. Das Kennwort muss dem BS2000-Kennwort für die Datei des Catalog-Space entsprechen.

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

space

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

AUTHORIZATION *berechtigungsschlüssel*

Name des Berechtigungsschlüssels, der als Eigentümer des Space eingetragen wird.

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

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

PRIMARY *zuweisung*

Primärzuweisung der Space-Datei in 2K-Einheiten (BS2000-Halfpage). *zuweisung* muss eine vorzeichenlose Ganzzahl von 1 bis 2 147 483 640 sein.

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

SECONDARY *zuweisung*

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

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

PCTFREE *prozent*

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

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

[NO] SHARE

SHARE gibt an, dass die Space-Datei gemeinsam benutzbar ist, d.h. dass nicht nur von der BS2000-Benutzerkennung des DBH auf die Space-Datei zugegriffen werden kann.

NO SHARE gibt an, dass die Space-Datei nicht gemeinsam benutzbar ist.

Aus Datenschutzgründen ist NO SHARE zu empfehlen.

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

[NO] DESTROY

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

[NO] DESTROY nicht angegeben:
Es gilt DESTROY.

NO LOG

Kein Logging.

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

USING STOGROUP *stogroup*

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

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

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

Space-Datei auf Betriebssystemebene

Die Space-Datei wird entweder unter der BS2000-Benutzerkennung des DBH oder der Datenbank mit folgendem Namen angelegt:

:*catid*:$*bk*.*catalog*.*einf_spacename*

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

Beispiel

Das Beispiel erzeugt die Space-Dateien TABLESPACE und INDEXSPCE mit einer Primär- und Sekundärzuweisung von jeweils 192 2K-Einheiten. Beide Dateien haben eine Freiplatzreservierung von 10 Prozent. Sie sind mehrfach benutzbar und werden beim Löschen mit binär Null überschrieben.

Auf dem Space INDEXSPACE sollen ausschließlich Indizes gespeichert werden. Indizes können im Rahmen des Media-Recovery aus den Primärdaten wieder aufgebaut werden. Logging ist daher nicht notwendig und wird mit NO LOG ausgeschaltet.

```
CREATE SPACE tablespace PRIMARY 192 SECONDARY 192
PCTFREE 10 SHARE DESTROY USING STOGROUP stogoup1

CREATE SPACE indexspace PRIMARY 192 SECONDARY 192
PCTFREE 10 SHARE DESTROY NO LOG USING STOGOUP stogroup1
```

Siehe auch

ALTER SPACE, CREATE STOGROUP

8.2.3.15 CREATE STOGROUP - Storage Group erzeugen

CREATE STOGROUP erzeugt eine neue Storage Group. Eine Storage Group beschreibt entweder ein Pubset oder einen Satz von Privatplatten. Die Privatplatten einer Storage Group müssen alle denselben Gerätetyp besitzen (siehe auch „ Basishandbuch“).

Die Storage Group D0STOGROUP ist immer vorhanden.

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

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

stogroup

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

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

VOLUMES (*volumename*,...)

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

Alle Platten einer Storage Group müssen den gleichen Gerätetyp besitzen.

ON *gerätetyp*

Gerätetyp der Privatplatten. *gerätetyp* ist ein alphanumerisches Literal, das als Zeichenfolge oder sedezimal angegeben werden kann.

PUBLIC

Die Storage Group umfasst ein Pubset.

ON *catid*

catid als alphanumerisches Literal.

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

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

Bei der Definition einer Storage Group auf einem Pubset können an Stelle des Parameters PUBLIC auch die Parameter VOLUMES (*volumename*,...) ON gerätetyp angegeben werden, um einzelne Volumes eines Pubsets auszuwählen. Die Kennung, unter der der DBH läuft, muss zur physischen Allokierung auf dem Pubset berechtigt sein. Dies wird bei der Definition der Storage Group nicht geprüft.

Beispiele

Das Beispiel erzeugt die Storage Group STOGROUP3 auf einem Pubset.

```
CREATE STOGROUP stogoup3 PUBLIC
```

Das Beispiel erzeugt eine neue Storage Group STOGROUP4 mit den angegebenen Privatplatten. Die Katalogkennung P soll zur Katalogisierung der Space-Dateien verwendet werden, die auf der Storage Group erzeugt werden.

```
CREATE STOGROUP stogroup4
   VOLUMES ('DY130A','DY130B','DY130C','DY130D') ON 'D3435'
   ON 'P'
```

Siehe auch

DROP STOGROUP

8.2.3.16 CREATE SYSTEM_USER - Systemzugang erzeugen

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

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

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

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

Beachten Sie, dass vor dem Umzug der Datenbank auf einen anderen Rechner, zuerst ein für den neuen Rechner gültiger Systemzugang definiert werden muss. Wenn Ihnen dies aus technischen Gründen nicht möglich sein sollte, wenden Sie sich bitte an Ihren ServiceBeauftragten.

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

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

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

---

utm_benutzer

Systemzugang eines UTM-Systembenutzers definieren.

bs2000_benutzer

Systemzugang eines BS2000-Systembenutzers definieren.

FOR *berechtigungsschlüssel*

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

AT CATALOG *catalog*

Name der Datenbank, für die die Zuordnung des Berechtigungsschlüssels zum Systembenutzer gilt.

utm_benutzer

Angabe des UTM-Benutzers.

rechnername

Symbolischer Rechnername als alphanumerisches Literal.

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

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

* Alle Rechner.

utm_anwendungsname

Name der UTM-Anwendung als alphanumerisches Literal.

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

* Alle UTM-Anwendungen.

utm_benutzerkennung

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

* Alle UTM-Benutzerkennungen.

bs2000_benutzer

Angabe des BS2000-Benutzers.

rechnername

Symbolischer Rechnername als alphanumerisches Literal.

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

* Alle Rechner.

bs2000_benutzerkennung

BS2000-Benutzerkennung als alphanumerisches Literal.

* Alle BS2000-Benutzerkennungen.

Beispiel

Im Beispiel werden zwei bereits definierten Berechtigungsschlüsseln Systembenutzer zugeordnet.

```
CREATE SYSTEM_USER (*,*,'FOTO') FOR utianw1 AT CATALOG auftragkunden

CREATE SYSTEM_USER (*,*,'TEXT') FOR utianw2 AT CATALOG auftragkunden
```

Aus der BS2000-Benutzerkennung FOTO darf der Berechtigungsschlüssel UTIANW1 auf die Datenbank AUFTRAGKUNDEN zugreifen. Der Berechtigungsschlüssel UTIANW2 darf aus der BS2000-Benutzerkennung TEXT aus auf die Datenbank AUFTRAGKUNDEN zugreifen.

Siehe auch

DROP SYSTEM_USER, CREATE USER

8.2.3.17 CREATE TABLE - Basistabelle erzeugen

CREATE TABLE erzeugt eine Basistabelle, in der die Daten permanent gespeichert sind.

SESAM/SQL unterscheidet zwischen:

- SQL-Tabellen, die nur mit SQL bearbeitet werden können.
- BLOB-Tabellen, die ausschließlich BLOB-Objekte enthalten.
- CALL-DML/SQL-Tabellen, die mit CALL-DML und eingeschränkt mit SQL bearbeitet werden können.
- Nur-CALL-DML-Tabellen, die nur mit CALL-DML bearbeitet werden können. Diese CALL-DML-Tabellen können nicht mit CREATE TABLE erzeugt werden. Sie werden mit der MIGRATE-Anweisung erzeugt (siehe Handbuch „ SQL-Sprachbeschreibung Teil 2: Utilities“).

SQL-Tabellen, BLOB-Tabellen und CALL-DML/SQL-Tabellen können auch als partitionierte Tabellen erzeugt werden. Ein partitionierte Tabelle ist eine Basistabelle, deren Daten in mehreren Spaces abgespeichert sind. Die auf einem Space liegenden Daten der Tabelle werden als Partition bezeichnet. In SESAM/SQL werden die Daten satzweise auf die Partitionen aufgeteilt, das Zuordnungskriterum ist der Primärschlüsselwert. Siehe auch Abschnitt „Besonderheiten für partitionierte Tabellen“.
Die Partitionierung kann mit der Utility-Anweisung ALTER PARTITIONING FOR TABLE geändert werden, siehe Handbuch „ SQL-Sprachbeschreibung Teil 2: Utilities“.

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

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

Der für BLOB-Tabellen festgelegte Tabellenaufbau wird im Abschnitt „Besonderheiten für BLOB-Tabellen“ beschrieben.

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

```
CREATE [CALL DML] TABLE tabelle

   { ( deklaration ...) | OF BLOB ( blob-deklaration ) }

   [USING { SPACE space | PARTITION BY RANGE partition ,..., letzte_partition }]


deklaration ::=
{ spaltendefinition | [CONSTRAINT integritätsbedingungsname ] tabellenbedingung }


blob-deklaration ::=
{ mimeklausel [, usageklausel ][, alphanumerisches_literal ] |
  usageklausel [, alphanumerisches_literal ] |
  alphanumerisches_literal }
```

mimeklausel ::= MIME(*alphanumerisches_literal*)

usageklausel ::= USAGE(*alphanumerisches_literal*)

partition ::= PARTITION *partnr* VALUE { < | <= } (*spaltenwert* ,...) ON SPACE *space*

letzte_partition ::= PARTITION *partnr* [VALUE <=( )] ON SPACE *space*

partnr ::= *vorzeichenlose_ganzzahl*

spaltenwert ::=
{ *alphanumerisches_literal* |
national_literal |
numerisches_literal |
zeit_literal }

CALL DML

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

CALL DML nicht angegeben:
Eine SQL- oder BLOB-Tabelle wird erzeugt.
SQL-Tabellen können nur mit SQL bearbeitet werden. BLOB-Tabellen werden nur mit Aufrufen der SESAM-CLI bearbeitet (siehe Kapitel „SESAM-CLI“).

TABLE *tabelle*

Name der neuen Basistabelle. Der einfache Tabellenname muss innerhalb der Basistabellen- und Viewnamen des Schemas eindeutig sein. Er kann durch einen Datenbank und Schemanamen qualifiziert werden.

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

spaltendefinition

Definiert Spalten für die Basistabelle.

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

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

CONSTRAINT *integritätsbedingungsname*

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

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

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

tabellenbedingung

Definiert eine Integritätsbedingung, die für die Basistabelle gilt.

OF BLOB

Erzeugt eine BLOB-Tabelle.

mimeklausel

Mit der *mimeklausel* kann der MIME-Typ definiert werden. Beispielsweise ist der MIME-Typ eines Microsoft ™ Word-Dokuments „application/msword“. Wird die BLOB-Tabelle ohne *mimeklausel* definiert, so ist für den MIME-Typ „application/octet-stream“ voreingestellt. Achten Sie darauf, dass in der *mimeklausel* nur zugelassene MIME-Typen deklariert werden. Eine Liste der wichtigsten MIME-Typen finden Sie zum Beispiel unter http://www.iana.org/assignments/media-types/index.html.

usageklausel

Die *usageklausel* kann für Kommentare zu den BLOB-Objekten benutzt werden (siehe Beispiel am Ende des Abschnitts). Der Defaultwert ist ein Leerzeichen.

alphanumerisches_literal

alphanumerisches_literal muss zusätzlich zu dem im Anhang beschriebenen Format XML-Format haben (siehe Beispiele).

USING-Klausel

Die USING-Klausel legt fest, ob eine nicht-partitionierte (USING SPACE) oder eine partitionierte Tabelle (USING PARTITION BY RANGE) angelegt wird.

USING SPACE *space*

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

USING PARTITION BY RANGE *partition, ... ,letzte_partition*

Gibt an, dass eine partitionierte Tabelle erzeugt werden soll. Die Tabelle muss mindestens aus 2 Partitionen und darf höchstens aus 16 Partitionen bestehen. Alle Partitionen einer Tabelle müssen auf unterschiedlichen Spaces liegen und alle Spaces müssen bereits für die Datenbank definiert sein. Die Tabelle muss einen Primärschlüssel besitzen, dieser kann eine einzelne Spalte oder eine Kombination aus mehreren Spalten sein.

partition-Klausel

Legt die Eigenschaften einer Partition fest.

partnr ist eine vorzeichenlose Ganzzahl von 1 ... 16 und stellt die laufende Nummer der Partition dar. *partnr* muss für die einzelnen Partitionen aufsteigend vergeben werden. Werden weniger als 16 Partitionen definiert, darf die Nummernfolge Lücken enthalten und die erste Partition muss nicht mit 1 beginnen.

(*spaltenwert*,....) ist eine Folge von Spaltenwerten, die für die betreffende Partition die Obergrenze des Primärschlüsselintervalls definiert. Sie müssen immer mindestens einen Spaltenwert angeben, jedoch höchstens so viele Spaltenwerte wie Spalten im Primärschlüssel vorhanden sind. Der Datentyp und der Wert von *spaltenwert* müssen zum Datentyp der entsprechenden Spalte des Primärschlüssel passen; es gelten dieselben Regeln wie für Defaultwerte (siehe Abschnitt „Defaultwerte für Tabellenspalten“).

Durch den vorangestellten Vergleichsoperator wird die Obergrenze mit eingeschlossen oder ausgeschlossen:

<=	Sätze, deren Primärschlüsselwert gleich *spaltenwert*,... ist oder deren Primärschlüsselwert mit *s* ,.. beginnt, gehören zu **dieser** Partition
<	Sätze, deren Primärschlüsselwert gleich *spaltenwert*,... ist oder deren Primärschlüsselwert mit *s* ,.. beginnt, gehören zur **nächsten** Partition

Für den Vergleich gelten die lexikografischen Regeln, siehe Abschnitt „Vergleichsregeln“.

Die angebenen Obergrenzen müssen für die einzelnen Partitionen streng aufsteigend sein.

Die untere Grenze der Partition ergibt sich implizit aus der Obergrenze der vorhergehenden Partition bzw. aus dem niedrigsten Primärschlüsselwert der Tabelle (bei der ersten Partition). Alle Sätze aus dem damit definierten Primärschlüsselintervall gehören zu dieser Partition.

space gibt den Namen des Space an, in dem diese Partition abgespeichert wird. Der Space muss existieren und der Space-Eigentümer muss gleichzeitig Schema-Eigentümer sein. Die Spaces einer partitionierten Tabelle müssen disjunkt sein, d.h. ein Space darf nicht für zwei Partitionen derselben Tabelle verwendet werden.

letzte_partition-Klausel

Für die letzte Partition gelten die gleichen Bedingungen wie für *partition*. Lediglich die Obergrenze darf nicht angegeben werden, da sie sich hier aus dem höchsten Primärschlüsselwert ergibt. Die VALUE-Klausel kann daher auch weggelassen werden.

USING nicht angegeben:
Es wird eine nicht-partitionierte Tabelle auf dem Default-Space des Schema-Eigentümers angelegt und auf der Storage Group D0STOGROUP gespeichert.Der voreingestellte Space ist D0*berechtigungsschlüssel* mit den ersten 10 Zeichen des Berechtigungsschlüssels. Existiert dieser Space noch nicht, wird er erzeugt, wenn der aktuelle Berechtigungsschlüssel das Sonder-Privileg USAGE für die Storage Group D0STOGROUP besitzt.

Besonderheiten fuer CALL-DML-Tabellen:

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

- Es sind nur die Datentypen CHAR, NUMERIC, DECIMAL, INTEGER oder SMALLINT erlaubt.
- Für eine Spalte darf mit DEFAULT kein voreingestellter Wert definiert werden.
- Eine Spalte, die nicht Primärschlüssel ist, muss eine CALL-DML-Klausel enthalten.
- Die Tabelle muss genau eine Primärschlüsselbedingung als Spaltenbedingung oder als Tabellenbedingung enthalten.
- Die Tabellenbedingung definiert einen zusammengesetzten Primärschlüssel und muss einen Namen erhalten, der dem Namen des zusammengesetzten Primärschlüssels in SESAM/SQL V1.x entspricht.
- Ein Spaltenname muss sich vom Integritätsbedingungsnamen der Tabellenbedingung unterscheiden, da dieser Name als Name des zusammengesetzten Primärschlüssels verwendet wird.
- Für SAN (symbolischer Attributsname) gelten folgende Regeln:
  - genau 3 Zeichen
  - erstes Zeichen: Buchstabe; zweites und drittes Zeichen: Buchstaben oder Ziffern
  - nicht zugelassene Zeichen: 0, I, O;
    die Kombinationen NAM und END sind ebenfalls nicht zulässig.

Besonderheiten fuer BLOB-Tabellen:

BLOB-Tabellen dienen in SESAM/SQL als Speicherort für BLOB-Objekte (**B**inary **L**arge **Ob**jects). BLOB-Objekte sind Byte-Ketten variabler Länge, die bis zu 2^{31}-1 Bytes groß sein können. Mit Hilfe von Aufrufen des SESAM-CLI werden die Werte von BLOB-Objekten stückweise in mehreren Zeilen der BLOB-Tabelle gespeichert. Die Struktur

dieser Tabelle wird bereits mit der Anweisung CREATE TABLE *tabelle* OF BLOB festgelegt. Eine Spaltendefinition ist an dieser Stelle nicht möglich.

Eine BLOB-Tabelle besitzt die folgenden Spalten:

- Die Spalte OBJ_NR ist vom Datentyp INTEGER und enthält die laufende Nummer des BLOB-Objekts innerhalb der Tabelle.
- Die Spalte SLICE_NR ist vom Datentyp INTEGER und enthält die laufende Nummer des Teilstücks.
- Die Spalte SLICE_VAL ist vom Typ VARCHAR(31000). Sie enthält jeweils die Teilstücke des BLOB-Werts. Die Einträge ab der Teilstücknummer 1 enthalten den Wert in 31 KB großen Teilstücken. Das letzte Teilstück kann natürlich kleiner sein. In der Zeile mit der Teilstücknummer 0 sind Verwaltungsinformationen für das BLOB-Objekt gespeichert. Die Voreinstellungen für diese Spalte sind die in der OF BLOB-Klausel definierten Attribute. Zusätzlich zu diesen sind auch die Attribute CREATED und UPDATED in den Voreinstellungen enthalten. Diese Attribute geben Auskunft über das Entstehungs- und letzte Änderungsdatum des BLOB-Objekts.
- Die Spalte OBJ_REF ist vom Typ CHAR(237). Bei Teilstücknummer 0 enthält diese Spalte den REF-Wert des BLOB-Objekts. Ansonsten ist der Spaltenwert NULL. Die Spalte erhält als Voreinstellung den REF-Wert für die Klasse dieser Tabelle und ist mit der Bedingung UNIQUE definiert.

Die Spalten OBJ_NR und SLICE_NR bilden den Primärschlüssel einer BLOB-Tabelle. Für diese Primärschlüsselbedingung werden wie üblich intern generierte Namen vergeben, die im selben Schema nicht mehr genutzt werden können.

Es ist für den Anwender möglich, weitere Spalten mit ALTER TABLE anzufügen. (Es muss jedoch darauf geachtet werden, dass die Voreinstellung für diese zusätzlichen Spalten der NULL-Wert ist.)

Die *mimeklausel* und *usageklausel* sowie das *alphanumerische_literal* in der CREATE TABLE...OF BLOB-Anweisung dienen zum Anfügen von Attributen, die das BLOB-Objekt beschreiben. Für alle Attribute zusammen stehen 256 Byte zur Verfügung.

Das Einbinden von BLOB-Werten in reguläre Basistabellen erfolgt mit Hilfe der REF-Spalten (siehe Abschnitt „Spaltendefinition“).

Besonderheiten für partitionierte Tabellen:

Eine partitionierte Tabelle verhält sich weitgehend wie eine nicht-partitionierte Tabelle, d.h. die Spalten, Bedingungen, Indizes und Defaultwerte beziehen sich auf alle Partitionen.

Da die Partitionsgrenzen mithilfe des Primärschlüssels festgelegt werden, sollten Sie beim Erstellen der partitionierten Tabelle Folgendes beachten:

- Sie können die Partitionsgrenzen einer partitionierten Tabelle nach der Erstellung mit ALTER PARTITIONING FOR TABLE verändern. Sie können auch mit den Utility-Anweisungen EXPORT TABLE und IMPORT TABLE eine Tabelle mit veränderten Partitionsgrenzen erzeugen.
- Nach dem Einfügen eines Satzes in eine partitionierte Tabelle kann sein Primärschlüsselwert nicht mehr mit der Anweisung UPDATE geändert werden. Der Satz kann aber gelöscht und mit neuem Primärschlüsselwert wieder eingefügt werden.

Für BLOBs besteht der Primärschlüssel aus den Spalten OBJ_NR und die SLICE_NR. Die Objektnummer wird bei den CLI-Aufrufen SQL_BLOB_OBJ_CREATE oder SQL_BLOB_OBJ_CREAT2 erzeugt. Diese beiden Aufrufe haben unterschiedliche Eigenschaften:

- Bei SQL_BLOB_OBJ_CREATE ("SQL_BLOB_OBJ_CREATE - SQLbocr") wird die Objektnummer fortlaufend aufsteigend vergeben.
- Bei SQL_BLOB_OBJ_CREAT2 ("SQL_BLOB_OBJ_CREAT2 - SQLboc2") geben Sie einen Objektnummernbereich an. Die Objektnummer des BLOBs wird dann von SESAM/SQL innerhalb dieses Bereichs vergeben und auch gleichmäßig innerhalb dieses Bereichs verteilt. Es ist daher sinnvoll, die Partitionsgrenzen mit den Objektnummernbereichen abzustimmen.

Weitere Informationen zu partitionierten Tabellen sowie Nutzungsszenarien finden Sie im „ Basishandbuch“.

Beispiele

Dieses Beispiel zeigt die CREATE TABLE-Anweisung für die nicht-partitionierte Tabelle AUFTRAG der Beispieldatenbank.

```
CREATE TABLE AUFTRAG
(
anr        INTEGER CONSTRAINT anr_primary PRIMARY KEY,
knr        INTEGER CONSTRAINT a_knr_notnull NOT NULL
                   CONSTRAINT a_knr_ref_kunde REFERENCES kunde(knr),
konr       INTEGER CONSTRAINT konr_ref_kontakt
                   REFERENCES kontakt(konr),
adatum     DATE    DEFAULT CURRENT_DATE,
atext      CHARACTER (30),
fertigist  DATE,
fertigsoll DATE,
astnr      INTEGER DEFAULT 1
                   CONSTRAINT astnr_notnull NOT NULL
                   CONSTRAINT astnr_ref_aufstat REFERENCES aufstat(astnr)
)
USING SPACE tablespace
```

Dieses Beispiel zeigt eine entsprechende CREATE TABLE-Anweisung für die Tabelle AUFTRAG der Beispieldatenbank als partitionierte Tabelle.

```
CREATE TABLE auftrag
(anr         INTEGER CONSTRAINT anr_primary PRIMARY KEY,
knr          INTEGER CONSTRAINT a_knr_notnull NOT NULL
                CONSTRAINT a_knr_ref_kunde
                REFERENCES kunde(knr),
konr         INTEGER
                CONSTRAINT konr_ref_kontakt
                REFERENCES kontakt(konr),
adatum       DATE DEFAULT CURRENT_DATE,
atext        CHARACTER (30),
fertigist    DATE,
fertigsoll   DATE,
```

```
astnr        INTEGER DEFAULT 1 CONSTRAINT astnr_notnull NOT NULL,
                               CONSTRAINT astnr_ref_aufstat
                                          REFERENCES aufstat(astnr)
)
USING PARTITION BY RANGE
      PARTITION 02 VALUE <= (299) ON SPACE tablespace,
      PARTITION 03 VALUE <= (399) ON SPACE tablesp002,
      PARTITION 09                ON SPACE tablesp003
```

Dieses Beispiel zeigt eine CREATE TABLE-Anweisung für die partitionierte Tabelle ADRESSEN. Die Daten werden lexikografisch auf 5 Partitionen aufgeteilt: A bis D, E bis K, L bis O, P bis SCH und SCI bis Z. Der Primärschlüssel besteht aus drei Spalten, wobei nur die erste Spalte für die Bestimmung der Partitionsgrenzen verwendet wird.

```
CREATE TABLE adressen
(name CHARACTER (40), vorname CHARACTER (40), pers_nr INTEGER, ...
 PRIMARY KEY (name, vorname, pers_nr))
USING PARTITION BY RANGE
      PARTITION 01 VALUE < ('E')   ON SPACE adr01,
      PARTITION 02 VALUE < ('L')   ON SPACE adr02,
      PARTITION 03 VALUE < ('P')   ON SPACE adr03,
      PARTITION 04 VALUE < ('SCI') ON SPACE adr04,
      PARTITION 05                 ON SPACE adr05
```

Dieses Beispiel zeigt die CREATE TABLE-Anweisung für die CALL-DML-Tabelle FIRMA im Schema FIRMASCH der Datenbank CALLFIRMA (siehe Handbuch „ CALL-DML Anwendungen“).

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

Im Schema ZUSAETZE werden die BLOB-Tabellen BILDER und BESCHREIBUNG der Beispieldatenbank angelegt. Beide Tabellen werden auf dem Space BLOBSPACE gespeichert. Während die Tabelle BLOB Bilder im gif-Format enthält, werden in der Tabelle BESCHREIBUNG Texte zu den Bildern als Word-Dokument gespeichert.

```
CREATE TABLE zusaetze.bilder OF BLOB
    ( MIME ('image / gif'),
      USAGE ('Abbildungen fuer teile.katart.abb'),
            '<Fotograf>Hans Sesamer</Fotograf>')
```

```
    USING SPACE blobspace

CREATE TABLE zusaetze.beschreibung OF BLOB
    ( MIME  ('application / msword'),
      USAGE ('Worddokumente fuer teile.katart.beschr'),
            '<AUTHOR>Herta Sesamer</AUTHOR>')
    USING SPACE blobspace
```

Dieses Beispiel zeigt eine CREATE TABLE-Anweisung für die partitionierte BLOB-Tabelle RECHNUNG. Die Tabelle enthält Rechnungen in Form von Word-Dateien. Die Rechnungen sind nach Quartalen eines Jahres auf die einzelnen Partitionen aufgeteilt.

```
CREATE TABLE rechnung OF BLOB (MIME ('application/msword'))
USING PARTITION BY RANGE
      PARTITION 01 VALUE <= (1000000)   ON SPACE quartal01,
      PARTITION 02 VALUE <= (2000000)   ON SPACE quartal02,
      PARTITION 03 VALUE <= (3000000)   ON SPACE quartal03,
      PARTITION 04                      ON SPACE quartal04
```

Eine Rechnung wird mit der CLI-Funktion SQL_BLOB_OBJ_CREAT2 erzeugt. Dabei wird der Objektnummernbereich der Rechnung (*min_nr*, *max_nr*) so gewählt, dass die Rechnung in der zum Quartal gehörenden Partition abgelegt wird:

```
SQL_BLOB_OBJ_CREAT2(&ref, &catalogId, &minObjNr, &maxObjNr, &SQLdiag);
```

Dieses Beispiel zeigt die CREATE TABLE-Anweisung für die Tabelle HANDBUECHER der Beispieldatenbank:

```
CREATE TABLE handbuecher
(bestellnummer INTEGER,
 sprache NCHAR(20),
 titel NCHAR(30)
)
```

Siehe auch

ALTER TABLE, CREATE SCHEMA, CREATE SPACE

8.2.3.18 CREATE USER - Berechtigungsschlüssel erzeugen

CREATE USER erzeugt einen neuen Berechtigungsschlüssel.

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

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

---

berechtigungsschlüssel

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

AT CATALOG *catalog*

Name der Datenbank, für die der neue Berechtigungsschlüssel gilt.

Beispiel

Das Beispiel definiert die Berechtigungsschlüssel UTIANW1 und UTIANW2 für die Datenbank AUFTRAGKUNDEN.

```
CREATE USER utianw1 AT CATALOG auftragkunden

CREATE USER utianw2 AT CATALOG auftragkunden
```

Siehe auch

DROP USER, CREATE SYSTEM_USER

8.2.3.19 CREATE VIEW - View erzeugen

CREATE VIEW erzeugt einen View. Ein View ist eine Tabelle, die nicht dauerhaft gespeichert ist, sondern deren Zeilen erst bei Bedarf bestimmt werden.

Der aktuelle Berechtigungsschlüssel muss Eigentümer des Schemas sein, für das der View erzeugt wird. Er muss das SELECT-Privileg für die verwendeten Tabellen und das EXECUTE-Privileg für die aufgerufenen UDFs besitzen.

```
CREATE VIEW tabelle

{ [( spalte ,...)] AS abfrageausdruck [WITH CHECK OPTION] |

   ( spalte ,...)  AS VALUES zeile ,... }


zeile ::= { ( ausdruck ,...) | ausdruck }
```

---

tabelle

Name des neuen View. Der einfache Viewname muss innerhalb der Basistabellen- und Viewnamen des Schemas eindeutig sein. Der einfache Viewname kann durch einen Datenbank- und Schemanamen qualifiziert werden.

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

(*spalte* ,...)

Name der Spalte im View. Die Viewspalten müssen bei *abfrageausdruck* nur dann benannt werden, wenn die Spalten der zu Grunde liegenden Tabellen nicht eindeutig benannt sind oder wenn Ergebnisspalten ohne Namen vorkommen.

(*spalte* ,...) nicht angegeben:
Es gelten die Spaltennamen des Abfrage-Ausdrucks.

AS *abfrageausdruck*

Abfrage-Ausdruck, der aus bereits bestehenden Basistabellen und Views die Tabelle erzeugt, die den neuen View bilden soll. Die Spalten des View besitzen denselben Datentyp wie die zu Grunde liegenden Spalten aus dem Abfrage-Ausdruck.

AS VALUES *zeile ,...*

Zeilen, die zusammen den neuen View bilden sollen. Alle Zeilen müssen dieselbe Anzahl von Spalten haben, und korrespondierende Spalten müssen verträgliche Datentypen haben (siehe Abschnitt „Verträglichkeit von Datentypen“). Sind mehrere Zeilen angegeben, so ergibt sich der Datentyp der Spalten des View aus den Regeln im Abschnitt „Datentyp der Ergebnisspalten bei UNION“.

ausdruck

Jeder *ausdruck* in *zeile* muss einfach sein. Die Zeile besteht aus den Werten von *ausdruck* in der angegebenen Reihenfolge. Ein einzelner *ausdruck* liefert also eine Zeile mit einer Spalte.

Die Tabellen, die in *abfrageausdruck* und *zeile* genannt werden, müssen zur gleichen Datenbank gehören wie der View. *abfrageausdruck* und *zeile* dürfen keine Benutzervariablen und keine Fragezeichen als Platzhalter für unbekannte Werte enthalten. Werden im View Spalten benannt, so muss ihre Anzahl gleich der Anzahl der Spalten in der Tabelle von *abfrageausdruck* oder von *zeile* sein.

WITH CHECK OPTION

Sätze, die Sie über den View eingeben oder ändern, werden auf die Einhaltung der im Abfrage-Ausdruck definierten Bedingung geprüft. Der View muss änderbar sein.

Der Abfrage-Ausdruck darf multiple Spalten und UDFs nur in der SELECT-Klausel, nicht in der WHERE-Klausel enthalten.

WITH CHECK OPTION nicht angegeben:
Falls der View änderbar ist, können Sätze in den View eingefügt oder geändert werden, die die Bedingung im Abfrage-Ausdruck nicht erfüllen. Auf solche Sätze kann nicht über den View zugegriffen werden.

Privilegien für den View

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

Wenn der View änderbar ist, erhält der aktuelle Berechtigungsschlüssel die Privilegien IN-SERT, UPDATE und DELETE, falls er diese Privilegien für die zu Grunde liegende Basistabelle besitzt. Die GRANT-Berechtigung zur Weitergabe dieser Privilegien erhält er nur, wenn er die GRANT-Berechtigung für die entsprechenden Privilegien der zu Grunde liegenden Basistabelle besitzt.

Änderbarer View

Ein View ist änderbar, wenn *abfrageausdruck* angegeben ist und der zu Grunde liegende Abfrage-Ausdruck änderbar ist (siehe Abschnitt „Änderbarkeit von Abfrage-Ausdrücken“).

Beispiele

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

```
CREATE VIEW fertig
```

```
AS SELECT * FROM auftrag WHERE fertigist IS NOT NULL
```

Das Beispiel definiert den View AUFERFASSUNG, der die Kundennamen und die dazugehörigen Auftragsnummern aus den Tabellen KUNDE und AUFTRAG heraussucht.

```
CREATE VIEW auferfassung AS SELECT firma, anr

FROM kunde, auftrag WHERE kunde.knr=auftrag.knr
```

Das Beispiel definiert den View WOCHENTAGE, der die Namen von Wochentagen enthält und jedem Wochentag eine Zahl zuordnet.

```
CREATE VIEW wochentage(nr, name)
       AS VALUES (1 , 'Montag')
           ,(2 , 'Dienstag')
           ,(3 , 'Mittwoch')
           ,(4 , 'Donnerstag')
           ,(5 , 'Freitag')
           ,(6 , 'Samstag')
           ,(7 , 'Sonntag')
```

Damit kann man für eine Spalte TAG_NR den Namen des Wochentages selektieren.

```
SELECT ..., (SELECT Name FROM wochentage WHERE nr = tag_nr)
```

Gegenüber der folgenden Version spart das nicht nur Schreibarbeit, sondern hat auch einen anderen Vorteil: Für eine andere Sprache muss man nur einen View austauschen und nicht mehrere SELECT-Ausdrücke ändern:

```
SELECT ..., CASE tag_nr WHEN 1 THEN 'Montag'
                 WHEN 2 THEN 'Dienstag'
                 WHEN 3 THEN 'Mittwoch'
                 WHEN 4 THEN 'Donnerstag'
                 WHEN 5 THEN 'Freitag'
                 WHEN 6 THEN 'Samstag'
                 WHEN 7 THEN 'Sonntag'
   END
```

Diesen Vorteil hat man selbstverständlich auch, wenn WOCHENTAGE eine Basistabelle mit diesem Inhalt ist. Allerdings erfolgt dann bei jeder Verwendung ein Zugriff auf dauerhaft gespeicherte Daten in einer Datei. Bei dem View werden nur Literale verwendet und es ist (ebenso wie beim CASE Ausdruck) kein solcher Zugriff notwendig.

Der View VIEW1 wählt aus der Tabelle AUFTRAG alle Auftragsnummern, Kundennummern, Soll-Termine der Fertigstellung und Auftragsstatusnummern aus, deren Soll-Termine vor dem angegebenen Datum liegen.

```
CREATE VIEW view1 AS SELECT anr,knr,fertigsoll,astnr

FROM auftrag WHERE fertigsoll < DATE'2014-05-01'
```

Auf den View VIEW1 wird ein zweiter View VIEW2 definiert. Dieser wählt Auftragsnummern, Kundennummern, Soll-Termine der Fertigstellung und Auftragsstatusnummern für Soll-Termine nach dem angegebenen Datum aus:

```
CREATE VIEW view2 AS SELECT anr,knr,fertigsoll,astnr
FROM view1 WHERE fertigsoll > DATE'2013-05-01'
```

Der View VIEW2 führt zu folgender Ergebnistabelle:

anr	knr	fertigsoll	astnr
210	106	2014-04-01	3
211	106	2014-04-01	4
250	105	2014-03-01	2

In den View VIEW2 soll jetzt ein neuer Satz aufgenommen werden:

```
INSERT INTO view2 (anr,knr,fertigsoll,astnr)
VALUES (310,100,DATE '2014-06-01',5)
```

Der neue Satz wird aufgenommen, ist aber weder im VIEW1 noch im VIEW2 sichtbar. Der Satz erfüllt die WHERE-Bedingung in der Definition von VIEW2, nicht aber die WHERE-Bedingung in der Definition von VIEW1. Expandiert man in VIEW2 die Definition von VIEW1, dann ergibt sich

```
CREATE VIEW view2 AS
SELECT view1.anr, view1.knr,view1.fertigsoll,view1.astnr
FROM
   (SELECT auftrag.anr, auftrag.knr,auftrag.fertigsoll,auftrag.astnr
   FROM auftrag
   WHERE auftrag.fertigsoll < DATE '2014-05-01') AS view1
WHERE view1.fertigsoll > DATE '2013-05-01'
```

Es wird deutlich, dass die WHERE-Bedingung in VIEW1 auf die Definition von VIEW2 „vererbt“ wird, so dass der in AUFTRAG aufgenommene Satz auch in VIEW2 nicht sichtbar ist.

Wird VIEW2 um WITH CHECK OPTION erweitert, dann wird die INSERT-Anweisung abgewiesen, da nur Sätze aufgenommen werden, die die WHERE-Bedingung von VIEW1 erfüllen.

Die INSERT-Anweisung wird jedoch auch abgewiesen, wenn nur die Definition von VIEW2 um WITH CHECK OPTION erweitert wird. Der aufzunehmende Satz verstößt zwar nicht gegen die WHERE-Bedingung in der Definition von VIEW2, die INSERT-Anweisungwird jedoch abgewiesen, da der Satz die WHERE-Bedingung von VIEW1 nicht erfüllt.

Siehe auch

CREATE SCHEMA, DROP VIEW

8.2.3.20 DEALLOCATE DESCRIPTOR - SQL-Deskriptorbereich freigeben

DEALLOCATE DESCRIPTOR gibt einen SQL-Deskriptorbereich frei.

Der Deskriptorbereich muss mit ALLOCATE DESCRIPTOR angelegt worden sein.

`DEALLOCATE DESCRIPTOR GLOBAL` *deskriptor*

---

deskriptor

Name des freizugebenden SQL-Deskriptorbereichs.
Sie können den Deskriptorbereich nicht freigeben, wenn in derselben Übersetzungseinheit ein geöffneter Cursor mit eingeschaltetem Schubmodus existiert (siehe Abschnitt „Pragma PREFETCH“), und für diesen Cursor eine Anweisung FETCH NEXT... ausgeführt wurde, deren INTO-Klausel den Namen desselben SQL-Deskriptorbereichs enthält.

Beispiel

SQL-Deskriptorbereich freigeben. Der Deskriptorbereichsname steht in der Benutzervariable DEMO_DESC.

```
DEALLOCATE DESCRIPTOR GLOBAL :demo_desc
```

Siehe auch

ALLOCATE DESCRIPTOR, DESCRIBE, GET DESCRIPTOR, SET DESCRIPTOR

8.2.3.21 DECLARE CURSOR - Cursor vereinbaren

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

Die Cursorvereinbarung muss im Programmtext vor allen Anweisungen stehen, in denen der Cursor verwendet wird. Alle Anweisungen, die diesen Cursor verwenden, müssen in derselben Übersetzungseinheit stehen. Dies gilt nicht für lokale Cursor (in Prozeduren).

DECLARE CURSOR ist keine ausführbare Anweisung.

```
DECLARE cursor [SCROLL | NO SCROLL ] CURSOR
               [WITH HOLD | WITHOUT HOLD]
               FOR { cursorbeschreibung | anweisungsbezeichner }

cursorbeschreibung ::=
      abfrageausdruck
       [ORDER BY   sort_ausdruck [ASC | DESC]
               [,  sort_ausdruck [ASC | DESC]]...]
       [FETCH FIRST max ROWS ONLY]
       [FOR { READ ONLY | UPDATE [OF spalte ,...] }]

sort_ausdruck ::= { spalte | { spalte ( posnr ) | spalte[posnr] } | spaltennummer | ausdruck }
posnr ::= vorzeichenlose_ganzzahl
spaltennummer ::= vorzeichenlose_ganzzahl
max ::= vorzeichenlose_ganzzahl
```

---

cursor

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

SCROLL

Der Cursor kann mit FETCH NEXT/PRIOR/FIRST/LAST/RELATIVE/ABSOLUTE in beliebiger Reihenfolge auf jeden Satz der Ergebnistabelle positioniert werden.

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

Wenn Sie SCROLL angeben, ist der Cursor *cursor* nicht änderbar. Es gilt implizit die FOR READ ONLY-Klausel.

NO SCROLL

Die Ergebnistabelle kann nur sequenziell gelesen werden. Der Cursor kann nur auf den jeweils nächsten Satz positioniert werden. Bei FETCH ist nur die Positionsangabe NEXT erlaubt.

WITH HOLD

Ein Cursor kann mit WITH HOLD spezifiziert werden. Ist ein solcher Cursor am Ende der Transaktion im geöffneten Zustand, so bleibt er es auch nach einem COMMIT WORK. WITH HOLD kann bei lokalen Cursorn (in Prozeduren) nicht angegeben werden, siehe Abschnitt „Cursor“.

Ein Cursor WITH HOLD wird dennoch geschlossen, falls der Cursor in einer Transaktion mit OPEN geöffnet oder mit FETCH positioniert wurde und diese Transaktion mit ROLLBACK beendet wird. Auch beim Ende der SQL-Sitzung wird der Cursor in jedem Fall geschlossen.

WITHOUT HOLD

Ein noch offener Cursor wird bei Transaktionsende geschlossen.

cursorbeschreibung

Statischen Cursor vereinbaren.

cursorbeschreibung definiert die Ergebnistabelle und die Eigenschaften des Cursors. Ein Satz der Ergebnistabelle wird frühestens dann bestimmt, wenn der Cursor mit OPEN geöffnet wird. Er wird spätestens beim Ausführen einer FETCH-Anweisung bestimmt.

anweisungsbezeichner

Dynamischen Cursor vereinbaren.

anweisungsbezeichner ist der Name einer dynamisch formulierten Cursorbeschreibung. Eine dynamisch formulierte Cursorbeschreibung kann zur Laufzeit des Programms angegeben werden. Es sind dieselben Klauseln erlaubt wie bei einer statischen Cursorbeschreibung. Eine dynamisch formulierte Cursorbeschreibung muss mit einer PRE-PARE-Anweisung vorbereitet werden, in der der Name *anweisungsbezeichner* verwendet wird.

abfrageausdruck

Abfrage-Ausdruck zur Auswahl von Sätzen und Spalten aus Basistabellen oder Views.

In *abfrageausdruck* wird der Wert für Benutzervariablen, Prozedurparameter und Prozedurvariablen erst beim Öffnen des Cursors ermittelt. Spezial-Literale sowie Zeitfunktionen, die in *abfrageausdruck* vorkommen, werden erst beim Öffnen des Cursors ausgewertet.

ORDER BY

Die ORDER BY-Klausel bezeichnet die Spalten, nach denen die Ergebnistabelle sortiert werden soll. Es wird zuerst nach den Werten der ersten angegebenen Spalte sortiert. Wenn Sätze in der ersten Spalte vorkommen, die nach den Vergleichsregeln gleiche Werte haben (siehe Abschnitt „Vergleich von zwei Zeilen“ff), werden diese gemäß der zweiten Sortierspalte sortiert usw. In SESAM/SQL sind NULL-Werte beim Sortieren kleiner als alle Nicht-NULL-Werte.

Die Reihenfolge der Sätze mit gleichen Werten in allen Sortierspalten ist undefiniert.

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

Wenn Sie ORDER BY angeben, ist der Cursor *cursor* nicht änderbar. Es gilt implizit die FOR READ ONLY-Klausel.

ORDER BY nicht angegeben:Die Reihenfolge der Sätze der Cursortabelle ist undefiniert.

spalte

Name der Spalte in *abfrageausdruck*, nach der sortiert werden soll. *spalte* muss ein einfacher Spaltenname ohne Tabellennamen sein. Er muss zu der Ergebnistabelle gehören, die mit *abfrageausdruck* erzeugt wird.

{*spalte*(*posnr*), *spalte*[*posnr*]}

Element einer multiplen Spalte, nach dem sortiert werden soll. *posnr* ist eine vorzeichenlose Ganzzahl, die die Positionsnummer des Spaltenelements in der multiplen Spalte angibt. Ansonsten muss das Spaltenelement zu der Ergebnistabelle gehören, die mit *abfrageausdruck* erzeugt wird.

> ¡ In der Syntax kursiv gedruckte eckige Klammern sind Sonderzeichen und müssen in der Anweisung angegeben werden.

spaltennummer

Nummer der Spalte, nach der sortiert werden soll.

spaltennummer ist eine vorzeichenlose Ganzzahl mit:1 <= *spaltennummer* <= Anzahl der Ergebnisspalten.

Durch die Angabe der Spaltennummer können Sie auch Spalten als Sortierbegriff verwenden, die keinen oder keinen eindeutigen Spaltennamen haben.

spaltennummer kann eine einfache Spalte oder eine multiple Spalte mit Dimension 1 bezeichnen.

ausdruck

Sie können auch nach Ausdrücken sortieren, die nicht in der Resultattabelle vorkommen, wie zum Beispiel UPPER(*spalte*).

Dabei müssen folgende Bedingungen erfüllt sein:

- *abfrageausdruck* muss ein einfacher SELECT-Ausdruck sein.
- *ausdruck* darf nicht nur aus einem Literal bestehen.
- *ausdruck* darf keine Unterabfrage und keine Mengenfunktion enthalten.
- In *ausdruck* dürfen Spalten von Tabellen aus der FROM-Klausel verwendet werden, auch wenn sie nicht in der SELECT-Liste vorkommen.

ASC

Die Werte der zugehörigen Spalte werden aufsteigend sortiert.

DESC

Die Werte der zugehörigen Spalte werden absteigend sortiert.

FETCH FIRST *max* ROWS ONLY

Begrenzt die von einem Cursor gelieferte Treffermenge auf *max* (vorzeichenlose Ganzzahl > 0) Treffersätze. Wenn die Cursorposition größer als *max* ist, dann wird ein SQL-STATE (keine Daten, Klasse 02xxx) geliefert. Ein Cursor mit dieser Klausel ist nicht-änderbar..

FOR READ ONLY

Die FOR READ ONLY-Klausel legt fest, dass der Cursor *cursor* nur zum Lesen der Sätze der Ergebnistabelle verwendet werden kann (Nicht-änderbarer Cursor oder Cursor zum Lesen).

Ist der zu Grunde liegende Abfrageausdruck nicht änderbar, so gilt die FOR READ ONLY-Klausel implizit (siehe Abschnitt „Änderbarkeit von Abfrage-Ausdrücken“). Sie gilt auch dann, wenn in der Cursorvereinbarung SCROLL, ORDER BY oder FETCH FIRST *max* ROWS ONLY angegeben wurden.

FOR UPDATE

Die FOR UPDATE-Klausel darf nur angegeben werden, wenn der zu Grunde liegende Abfrageausdruck änderbar ist (siehe Abschnitt „Änderbarkeit von Abfrage-Ausdrücken“) und weder SCROLL noch ORDER BY noch FETCH FIRST *max* ROWS ONLY angegeben wurden. Mit der FOR UPDATE-Klausel können Sie festlegen, welche Spalten der zu Grunde liegenden Tabelle über den Cursor mit UPDATE...WHERE CURRENT OF geändert werden dürfen.

Wurde für den betreffenden Cursor ein Pragma PREFETCH vereinbart, so schaltet die FOR UPDATE-Klausel die Wirksamkeit dieses Pragmas aus (siehe Abschnitt „Pragma PREFETCH“).

FOR UPDATE nicht angegeben:Ist der Cursor änderbar (siehe Abschnitt „Cursor definieren“) und wurde keine FOR READ ONLY-Klausel angegeben, können alle Spalten der zu Grunde liegenden Tabelle mit UPDATE... WHERE CURRENT OF geändert werden.

OF *spalte*,...

Nur die angegebenen Spalten dürfen mit UPDATE...WHERE CURRENT OF geändert werden.Für *spalte* geben Sie den Namen einer Spalte der Tabelle an, auf die sich der änderbare Cursor bezieht. *spalte* ist der einfache Name der Spalte aus der zu Grunde liegenden Tabelle, auch wenn im Abfrage-Ausdruck der Cursorbeschreibung neue Spaltennamen definiert werden.

Beispiel

Es wird ein änderbarer Cursor CUR vereinbart. Die zu Grunde liegende Tabelle ist TAB. Nur die Spalte sp der Tabelle TAB darf über den Cursor CUR geändert werden. Dazu wird in der Cursorbeschreibung eine FOR UPDATE-Klausel mit dem Spaltennamen SP angegeben.

```
DECLARE cur CURSOR FOR
   SELECT korr.sp AS spalte FROM tab AS korr
   FOR UPDATE OF sp
```

In der FOR UPDATE-Klausel wird der einfache ursprüngliche Spaltenname SP verwendet, obwohl in der SELECT-Liste der Spaltenname und in der FROM-Klausel die Tabelle umbenannt wurden.

OF *spalte*,... nicht angegeben:Jede Spalte der zu Grunde liegenden Tabelle kann mit UPDATE...WHERE CURRENT OF geändert werden.

Beispiele

Der Cursor CUR_AUFTRAG wählt ANR, KNR, KONR, ATEXT, FERTIGSOLL und ASTNR aus für Aufträge mit einer Auftragsnummer zwischen 300 und 500. Die Sätze werden nach aufsteigender Auftragsnummer sortiert.

```
DECLARE cur_auftrag CURSOR FOR
   SELECT anr, knr, konr, atext, fertigsoll, astnr
   FROM   auftrag
   WHERE  anr BETWEEN 300 AND 500
   ORDER  BY anr ASC
```

Der Cursor CUR_AUFTRAG1 wählt ANR, ADATUM, ATEXT und ASTNR aus für Aufträge, deren Kundennummer in der Benutzervariablen KUNDENNR angegeben wird.

```
DECLARE cur_auftrag1 CURSOR FOR
   SELECT anr, adatum, atext, astnr
   FROM   auftrag
   WHERE  knr= :KUNDENNR
```

Der Cursor CUR_MWST wählt alle Leistungen aus, für die keine Mehrwertsteuer berechnet wird. Der Cursor ist mit WITH HOLD spezifiziert, so dass er auch nach einem COMMIT WORK im geöffneten Zustand bleibt, wenn er am Ende der Transaktion im geöffneten Zustand war.

```
DECLARE cur_mwst CURSOR WITH HOLD FOR
   SELECT lnr, ltext, mwsatz
   FROM   leistung
   WHERE  mwsatz=0.00
   FOR UPDATE
```

Vereinbarung des Schubmodus für einen statischen Cursor.

`--%PRAGMA PREFETCH` *blockungsfaktor*
`DECLARE` *cursor* `CURSOR FOR` *cursorbeschreibung*

Siehe auch

CLOSE, DELETE, FETCH, INSERT, OPEN, PREPARE, SELECT, UPDATE

8.2.3.22 DELETE - Sätze löschen

DELETE löscht Sätze aus einer Tabelle.

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

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

```
DELETE FROM tabelle [[AS] korrelationsname ]
[WHERE { suchbedingung | CURRENT OF cursor }]
```

tabelle

Name der Tabelle, aus der Sätze gelöscht werden sollen. Die Tabelle kann eine Basistabelle oder ein änderbarer View sein.

korrelationsname

Tabellenname, der innerhalb der *suchbedingung* eine Umbenennung für *tabelle* ist.

Bei jeder Spaltenangabe, die sich auf *tabelle* bezieht, müssen Sie den Spaltennamen mit dem neuen Namen *korrelationsname* qualifizieren, wenn der Spaltenname nicht eindeutig ist.

Der neue Name muss eindeutig sein, d.h. *korrelationsname* darf nur einmal in einer Tabellenangabe dieser Suchbedingung vorkommen.

Sie müssen eine Tabelle umbenennen, wenn die Spalten der Tabelle ohne Umbenennung nicht eindeutig angegeben werden können.

Außerdem können Sie eine Tabelle umbenennen, um durch entsprechende Namen einen Ausdruck verständlicher zu formulieren oder um lange Namen abzukürzen.

WHERE-Klausel

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

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

suchbedingung

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

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

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

CURRENT OF *cursor*

Name des Cursors, über den die zu löschenden Sätze ausgewählt werden. Der Cursor muss änderbar sein (siehe Abschnitt „Cursor definieren“), und *tabelle* muss die zu Grunde liegende Tabelle sein.

Der Cursor muss in derselben Übersetzungseinheit vereinbart sein. Er muss offen sein. Er muss vor der DELETE-Anweisung mit FETCH auf einen Satz der Ergebnistabelle positioniert worden sein.

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

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

DELETE ist nicht erlaubt, wenn für den geöffneten Cursor *cursor* der Schubmodus eingeschaltet ist (siehe Abschnitt „Pragma PREFETCH“).

Wurde ein Cursor mit der WITH HOLD-Klausel definiert, dann darf ein DELETE erst erfolgen, nachdem in derselben Transaktion eine FETCH-Anweisung für diesen Cursor ausgeführt wurde.

DELETE und Transaktionssicherung

DELETE leitet außerhalb von Routinen eine SQL-Transaktion ein, wenn keine Transaktion offen ist. Durch die Definition eines Isolationslevels bei konkurrierenden Transaktionen können Sie steuern, welche Auswirkungen die DELETE-Anweisung auf diese Transaktionen hat (siehe Abschnitt „SET TRANSACTION - Transaktionseigenschaften festlegen“).

Tritt während der DELETE-Anweisung ein Fehler auf, so werden alle bereits durchgeführten Löschungen rückgängig gemacht.

Beispiele

Alle in Hannover ansässigen Kunden werden aus der Tabelle KUNDE gelöscht.

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

In der Tabelle KONTAKT sollen alle Kunden gelöscht werden, für die in der Tabelle KUNDE als Land „USA“ eingetragen ist. Die Anweisung wird nur ausgeführt, wenn die Referenzbedingung KONR_REF_KONTAKT der Tabelle AUFTRAG nicht verletzt wird.

```
DELETE FROM kontakt
WHERE knr = (SELECT knr FROM kunde WHERE land='USA')
```

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

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

OPEN cur_kunden
```

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

```
FETCH cur_kunden INTO :KNR, :FIRMA, :ORT

DELETE FROM kunde WHERE CURRENT OF cur_kunden
```

In diesem Beispiel werden mit einem Cursor die Aufträge aus der Tabelle AUFTRAG ausgewählt, die bereits abgelegt sind (ASTNR = 5). Die Einträge zu diesen Aufträgen werden anschließend in den Tabellen LEISTUNG und AUFTRAG gelöscht.

```
DECLARE cur_auftrag1 CURSOR FOR
   SELECT anr, atext FROM auftrag WHERE astnr = 5
   FOR UPDATE

FETCH cur_auftrag1 INTO :AUFTRAG.ANR

DELETE FROM auftrag
   WHERE CURRENT OF cur_auftrag1

DELETE FROM leistung
   WHERE anr = :AUFTRAG.ANR
```

Siehe auch

INSERT, UPDATE

8.2.3.23 DESCRIBE - Datentypen von Ein- und Ausgabewerten abfragen

DESCRIBE schreibt Datentypbeschreibungen von Eingabe- bzw. Ausgabewerten einer dynamisch formulierten Anweisung bzw. Cursorbeschreibung in einen SQL-Deskriptorbereich..

Der SQL-Deskriptorbereich muss vorher mit ALLOCATE DESCRIPTOR angelegt worden sein.

Die dynamisch formulierte Anweisung bzw. Cursorbeschreibung muss vor Ausführung der DESCRIBE-Anweisung mit PREPARE vorbereitet worden sein.

`DESCRIBE [INPUT | OUTPUT]` *anweisungsbezeichner* `USING SQL DESCRIPTOR GLOBAL` *deskriptor*

INPUT

Anzahl von Eingabewerten einer dynamisch formulierten Anweisung bzw. Cursorbeschreibung bestimmen und Datentyp der Eingabewerte beschreiben.

OUTPUT

Anzahl von Ausgabewerten einer dynamisch formulierten SELECT-Anweisung bzw. Cursorbeschreibung bestimmen und Datentyp der Ausgabewerte beschreiben.

anweisungsbezeichner

Dynamisch formulierte Anweisung bzw. Cursorbeschreibung.

deskriptor

Name des SQL-Deskriptorbereichs, in den die Typbeschreibungen geschrieben werden (siehe „Belegung der Deskriptorbereichsfelder“).

Sie können den Namen als alphanumerisches Literal oder mit einer alphanumerischen Benutzervariable angeben.

Diesen SQL-Deskriptorbereich können Sie nicht verwenden, wenn ein geöffneter Cursor mit eingeschaltetem Schubmodus existiert (siehe Abschnitt „Pragma PREFETCH“), und für diesen Cursor eine Anweisung FETCH NEXT... ausgeführt wurde, deren INTO-Klausel den Namen desselben SQL-Deskriptorbereichs enthält.

Belegung der Deskriptorbereichsfelder

Die Felder des SQL-Deskriptorbereichs werden wie folgt belegt:

Das Feld COUNT wird mit der Anzahl der Eingabewerte (DESCRIBE INPUT) bzw. Ausgabewerte (DESCRIBE OUTPUT) belegt.

Bei DESCRIBE INPUT berechnet sich die Anzahl wie folgt aus der Anzahl der Platzhalter der dynamisch formulierten Anweisung bzw. Cursorbeschreibung:

Anzahl der Platzhalter für einfache Werte +
Anzahl der Aggregatselemente von jedem Platzhalter für Aggregate

Bei DESCRIBE OUTPUT berechnet sich die Anzahl wie folgt aus der Anzahl der Ergebnisspalten der dynamisch formulierten SELECT-Anweisung bzw. Cursorbeschreibung:

Anzahl der einfachen Ergebnisspalten +
Anzahl der Spaltenelemente von jeder multiplen Ergebnisspalte

Ist die berechnete Anzahl 0, werden keine weiteren Deskriptorbereichsfelder gesetzt.

Ist die Anzahl größer als die bei ALLOCATE DESCRIPTOR angegebene Maximalanzahl der Einträge, werden keine weiteren Felder gesetzt und ein entsprechender SQLSTATE gesetzt.

Ansonsten werden folgende Felder des SQL-Deskriptorbereichs belegt.

- Für jeden Eingabewert bei DESCRIBE INPUT:
  - TYPE
  - LENGTH (bei alphanumerischem Datentyp, National-Datentyp und Zeit-Datentyp)
  - PRECISION (bei numerischem Datentyp sowie bei TIME und TIMESTAMP)
  - SCALE (bei NUMERIC, DECIMAL, INTEGER und SMALLINT)
  - DATETIME_INTERVAL_CODE (bei Zeit-Datentyp)
  - OCTET_LENGTH
  - NULLABLE mit dem Wert 1
  - REPETITIONS
  - UNNAMED mit dem Wert 1
- Für jeden Ausgabewert bei DESCRIBE OUTPUT:
  - TYPE
  - LENGTH (bei alphanumerischem Datentyp, National-Datentyp und Zeit-Datentyp)
  - PRECISION (bei numerischem Datentyp sowie bei TIME und TIMESTAMP)
  - SCALE (bei NUMERIC, DECIMAL, INTEGER und SMALLINT)
  - DATETIME_INTERVAL_CODE (bei Zeit-Datentyp)
  - OCTET_LENGTH
  - NULLABLE
  - REPETITIONS
  - NAME
  - UNNAMED

Mit welchen Werten die angegebenen Felder belegt werden, ist im Abschnitt „Deskriptorbereichsfelder“ beschrieben.

Alle anderen Felder des SQL-Deskriptorbereichs sind undefiniert.

Beispiel

```
DESCRIBE OUTPUT cur_beschreibung
```

```
USING SQL DESCRIPTOR GLOBAL 'DESKR_BEREICH'
```

Siehe auch

ALLOCATE DESCRIPTOR, DEALLOCATE DESCRIPTOR, GET DESCRIPTOR, SET DESCRIPTOR

8.2.3.24 DROP FUNCTION - User Defined Function (UDF) löschen

DROP FUNCTION löscht eine UDF.

UDFs und ihre Verwendung in SESAM/SQL sind detailliert im Kapitel „Routinen“ beschrieben.

Welche Routinen definiert sind und welche Routinen einander verwenden, erfahren Sie in den Views für Routinen des INFORMATION_SCHEMA (siehe Kapitel „Informationsschemata“).

Das Löschen der UDF entzieht dem aktuellen Berechtigungsschlüssel das EXECUTE-Privileg für diese UDF. EXECUTE-Privilegien, die weitergegeben wurden, werden ebenfalls entzogen.

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

```
DROP FUNCTION udf { CASCADE | RESTRICT }

udf ::= routine
```

---

udf

Name der UDF. Der einfache UDF-Name kann durch einen Datenbank- und Schemanamen qualifiziert werden.

CASCADE

Die UDF *udf* sowie jede Routine, die *udf* direkt oder indirekt aufruft, werden gelöscht. Auch Views, die *udf* direkt oder indirekt verwenden, werden gelöscht.

RESTRICT

Das Löschen der UDF *udf* ist nur möglich, wenn *udf* von keiner anderen Routine oder von keinem View verwendet wird.

Siehe auch

CREATE FUNCTION, CREATE PROCEDURE

8.2.3.25 DROP INDEX - Index löschen

DROP INDEX löscht einen Index. Der Index kann entweder explizit durch die Anweisung CREATE INDEX oder implizit durch die Definition einer Integritätsbedingung (UNIQUE) erzeugt worden sein.

Welche Indizes definiert sind, erfahren Sie im View INDEXES des INFORMATION_SCHEMA (siehe Kapitel „Informationsschemata“).

Wird ein explizit definierter Index zusätzlich von einer Integritätsbedingung verwendet, wird der Index nicht gelöscht, sondern in einen impliziten Index umbenannt. Der neue Indexname beginnt mit UI, gefolgt von einer 16-stelligen Nummer.

Indizes, die nur implizit durch Integritätsbedingungen (UNIQUE) erzeugt wurden, werden erst dann gelöscht, wenn die zugehörige Integritätsbedingung gelöscht wird.

Der aktuelle Berechtigungsschlüssel muss Eigentümer des Schemas sein, zu dem der Index gehört.

```
DROP INDEX index [DEFERRED]
```

index

Name des Index, der gelöscht werden soll.

Der einfache Name des Index kann durch einen Datenbank- und Schemanamen qualifiziert werden.

DEFERRED

Diese Klausel bewirkt ein schnelles Löschen, bei dem nur der zusammenhängende Teil des Index gelöscht wird. Eventuell vorhandene Auslagerungen bleiben erhalten. Bei der nächsten Reorganisation des Anwender-Spaces mit der Utility-Anweisung REORG SPACE werden alle existierenden Tabellen und Indizes auf dem Anwender-Space neu aufgebaut. Damit verschwinden auch die Auslagerungen.
Informationen zur Speicherstruktur von Indizes finden Sie im „ Basishandbuch“.

Ein implizit erzeugter Index (bei einer UNIQUE-Integritätsbedingung) kann nicht explizit gelöscht werden. Nötigenfalls muss der Index explizit mit CREATE INDEX erzeugt werden. Dabei wird nur in den Metadaten der „Generate_Type“ von „implizit“ auf „explizit“ geändert. Anschließend kann die UNIQUE-Integritätsbedingung gelöscht werden. Der Index wird dabei nicht gelöscht und kann jetzt mit DROP INDEX ... DEFERRED gelöscht werden.

Die Klausel DEFERRED kann nur beim expliziten Löschen angegeben werden. Beim impliziten Löschen, z.B. mit DROP SPACE CASCADE, kann sie nicht angegeben werden.

DEFERRED nicht angegeben:
SESAM/SQL löscht die Indizes. Bei sehr großen, zersplitterten Indizes kann dies zeitaufwändig sein.

Siehe auch

CREATE INDEX

8.2.3.26 DROP PROCEDURE - Prozedur löschen

DROP PROCEDURE löscht eine Prozedur.

Prozeduren und ihre Verwendung in SESAM/SQL sind detailliert im Kapitel „Routinen“ beschrieben.

Welche Routinen definiert sind und welche Routinen einander verwenden, erfahren Sie in den Views für Routinen des INFORMATION_SCHEMA (siehe Kapitel „Informationsschemata“).

Das Löschen der Prozedur entzieht dem aktuellen Berechtigungsschlüssel das EXECUTE-Privileg für diese Prozedur. EXECUTE-Privilegien, die weitergegeben wurden, werden ebenfalls entzogen.

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

`DROP PROCEDURE` *prozedur* `{ CASCADE | RESTRICT }`

prozedur `::=` *routine*

---

prozedur

Name der Prozedur. Der einfache Prozedurname kann durch einen Datenbank- und Schemanamen qualifiziert werden.

CASCADE

Die Prozedur *prozedur* und jede Routine, die *prozedur* direkt oder indirekt aufruft, werden gelöscht. Auch Views, die *prozedur* indirekt über eine UDF verwenden, werden gelöscht.

RESTRICT

Das Löschen der Prozedur *prozedur* ist nur möglich, wenn *prozedur* von keiner anderen Routine verwendet wird.

Siehe auch

CREATE PROCEDURE, CREATE FUNCTION, CALL

8.2.3.27 DROP SCHEMA - Schema löschen

DROP SCHEMA löscht ein Datenbankschema.

Welche Schemata definiert sind, erfahren Sie im View SCHEMATA des INFORMATION_SCHEMA (siehe Kapitel „Informationsschemata“).

Der aktuelle Berechtigungsschlüssel muss Eigentümer des Schemas sein.

```
DROP SCHEMA schema { CASCADE | RESTRICT }
```

schema

Name des Schemas.

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

CASCADE

Das Schema *schema* und alle Objekte des Schemas werden gelöscht. Auch Views, Routinen und Integritätsbedingungen, die sich direkt oder indirekt auf die in *schema* enthaltenen Basistabellen, Views und Routinen beziehen, werden gelöscht.

RESTRICT

Das Löschen des Schemas *schema* ist nur möglich, wenn es leer ist. Alle Basistabellen, Views und Routinen des Schemas müssen zuvor gelöscht werden.

Beispiel

Das Beispiel löscht das Schema ZUSAETZE, wenn vorher alle Basistabellen, Views und Routinen des Schemas gelöscht wurden. Das Schema wurde mit dem Datenbank-Namen qualifiziert.

```
DROP SCHEMA auftragkunden.zusaetze RESTRICT
```

Siehe auch

CREATE SCHEMA, DROP TABLE, DROP VIEW, DROP FUNCTION, DROP PROCEDURE

8.2.3.28 DROP SPACE - Space löschen

DROP SPACE löscht einen Anwender-Space.

Welche Anwender-Spaces definiert sind, erfahren Sie im View SPACES des INFORMATION_SCHEMA (siehe Kapitel „Informationsschemata“).

Der aktuelle Berechtigungsschlüssel muss Eigentümer des Space sein.

```
DROP SPACE space { CASCADE | RESTRICT } [FORCED]
```

space

Name des Anwender-Space.
Der einfache Name des Space kann durch einen Datenbanknamen qualifiziert werden.

CASCADE

Der Space *space* wird gelöscht, auch wenn er nicht leer ist. Auch die auf dem Space liegenden Basistabellen und Indizes werden gelöscht. Dies gilt auch für die Views, Routinen und Integritätsbedingungen, die sich direkt oder indirekt auf diese Basistabellen und Indizes beziehen.

RESTRICT

Der Space *space* wird nur gelöscht, wenn er leer ist. Alle Basistabellen und Indizes des Space müssen zuvor gelöscht werden.

FORCED

Der Space *space* wird auch dann gelöscht, wenn er nicht für die Update-Verarbeitung geöffnet werden kann, z. B. weil die BS2000-Datei des Space nicht mehr existiert. Der Space wird dann in SESAM/SQL logisch gelöscht, d.h. aus den Metadaten der Datenbank entfernt. Wenn auch CASCADE angegeben ist, dann wirkt FORCED auch für Spaces, die vom Löschen der Tabellen und Indizes betroffen sind.

FORCED nicht angegeben
Der Space *space* wird nur dann gelöscht, wenn er für die Update-Verarbeitung geöffnet werden kann.

> SESAM/SQL kann den Space für die Update-Verarbeitung öffnen, wenn die BS2000-Datei des Space fehlerfrei geöffnet werden kann, wenn der Space konsistent ist und wenn er sich nicht in einem der Zustände „check pending“, „copy pending“, „load running“, „recover pending“, „reorg pending“ oder „space defect“ befindet (siehe Handbuch „ SQL-Sprachbeschreibung Teil 2: Utilities“.
>
> Die Space-Datei wird mit binär Null überschrieben, wenn dies beim Erzeugen oder Ändern des Space mit der Klausel DESTROY festgelegt wurde und SESAM/SQL auf die BS2000-Datei des Space zugreifen kann.

DROP SPACE und Transaktionen

Nach der Anweisung DROP SPACE darf in derselben Transaktion keine CREATE SPACE-Anweisung folgen.

Siehe auch

CREATE SPACE, ALTER SPACE

8.2.3.29 DROP STOGROUP - Storage Group löschen

DROP STOGROUP löscht eine Storage Group. Eine Storage Group kann nicht gelöscht werden, wenn sie für Spaces verwendet wird oder in der Medientabelle eingetragen ist (siehe „ Basishandbuch“).

Welche Storage Groups definiert sind, erfahren Sie im View STOGROUPS des INFORMATION_SCHEMA (siehe Kapitel „Informationsschemata“).

Der aktuelle Berechtigungsschlüssel muss Eigentümer der Storage Group sein.

```
DROP STOGROUP stogroup RESTRICT
```

stogroup

Name der Storage Group. Die Storage Group muss unbenutzt sein.

Der einfache Name der Storage Group kann durch einen Datenbanknamen qualifiziert werden.

Siehe auch

CREATE STOGROUP, ALTER STOGROUP

8.2.3.30 DROP SYSTEM_USER - Systemzugang löschen

DROP SYSTEM_USER löscht einen Systemzugang, d.h. die Zuordnung eines Berechtigungsschlüssels zu einem Systembenutzer. Es muss eine Kombination von Systembenutzer und Berechtigungsschlüssel angegeben werden, für die mit CREATE SYSTEM_USER ein Systemzugang definiert wurde.

Ein Systemzugang kann nicht gelöscht werden, wenn er die letzte Zuordnung eines Systembenutzers zum Berechtigungsschlüssel des universellen Benutzers ist.

Ist momentan eine SQL-Transaktion des Systembenutzers aktiv, so wird dessen Systemzugang nur gelöscht, wenn noch ein anderer Systemzugang für den Systembenutzer existiert.

Welche Berechtigungsschlüssel welchen Systembenutzern zugeordnet sind, erfahren Sie im View SYSTEM_ENTRIES des INFORMATION_SCHEMA (siehe Kapitel „Informationsschemata“).

Der aktuelle Berechtigungsschlüssel muss das Sonder-Privileg CREATE USER besitzen. Wenn die Zuordnung eines Berechtigungsschlüssels mit dem Sonder-Privileg CREATE USER und mit GRANT-Berechtigung (siehe Abschnitt „GRANT - Privilegien vergeben“) zu einem Systembenutzer gelöscht werden soll, so muss der aktuelle Berechtigungsschlüssel zusätzlich die GRANT-Berechtigung besitzen.

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

utm_benutzer ::= ({ rechnername |*},{ utm_anwendungsname |*},{ utm_benutzerkennung |*})

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

---

utm_benutzer

Systemzugang eines UTM-Systembenutzers löschen.

bs2000_benutzer

Systemzugang eines BS2000-Systembenutzers löschen.

FOR *berechtigungsschlüssel*

Name des Berechtigungsschlüssels, dem der Systembenutzer zugeordnet ist.

AT CATALOG *catalog*

Name der Datenbank, für die die Zuordnung des Systembenutzers zum Berechtigungsschlüssel gelöscht werden soll.

utm_benutzer

Angabe des UTM-Benutzers.

Die Angabe des UTM-Benutzers muss exakt so erfolgen, wie sie mit CREATE SYSTEM_USER definiert wurde. * bedeutet den Systemzugang, der mit * definiert wurde, nicht alle entsprechenden Systemzugänge.

rechnername

Symbolischer Rechnername als alphanumerisches Literal.

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

* Alle Rechner.

utm_anwendungsname

Name der UTM-Anwendung als alphanumerisches Literal.

* Alle UTM-Anwendungen.

utm_benutzerkennung

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

* Alle UTM-Benutzerkennungen.

bs2000_benutzer

Angabe des BS2000-Benutzers.

Die Angabe des BS2000-Benutzers muss exakt so erfolgen, wie sie mit CREATE SYSTEM_USER definiert wurde. * bedeutet den Systemzugang, der mit * definiert wurde, nicht alle entsprechenden Systemzugänge.

rechnername

Symbolischer Rechnername als alphanumerisches Literal. Falls DCAM auf dem Rechner nicht verfügbar ist, ist dem Rechner fest der symbolische Name 'HOME-PROC' zugeordnet.

* Alle Rechner.

bs2000_benutzerkennung

BS2000-Benutzerkennung als alphanumerisches Literal.

* Alle BS2000-Benutzerkennungen.

Beispiel

Im Beispiel werden zwei Systemzugänge gelöscht. Die Systemzugänge müssen genauso angegeben werden, wie sie mit CREATE SYSTEM_USER definiert wurden. Die Berechtigungsschlüssel UTIANW1 und UTIANW2 werden nicht gelöscht.

```
DROP SYSTEM_USER (*,*,'FOTO') FOR utianw1 AT CATALOG auftragkunden
```

```
DROP SYSTEM_USER (*,*,'TEXT') FOR utianw2 AT CATALOG auftragkunden
```

Siehe auch

CREATE SYSTEM_USER, CREATE USER, DROP USER

8.2.3.31 DROP TABLE - Basistabelle löschen

DROP TABLE löscht eine Basistabelle und die zugehörigen Indizes.

Das Löschen einer Basistabelle entzieht dem aktuellen Berechtigungsschlüssel alle Tabellen und Spalten-Privilegien für diese Basistabelle. Tabellen- und Spalten-Privilegien, die weitergegeben wurden, werden ebenfalls entzogen.

Welche Basistabellen definiert sind, erfahren Sie im View BASE_TABLES des INFORMATION_SCHEMA (siehe Kapitel „Informationsschemata“).

Mit DROP TABLE können auch BLOB-Tabellen gelöscht werden. Dabei werden alle enthaltenen BLOB-Objekte gelöscht.

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

```
DROP TABLE tabelle [DEFERRED] { CASCADE | RESTRICT }
```

tabelle

Name der Basistabelle, die gelöscht werden soll.

DEFERRED

Diese Klausel bewirkt ein schnelles Löschen der Tabelle, bei dem nur der zusammenhängende Teil der Tabelle und der zugehörigen expliziten und impliziten Indizes gelöscht werden. Eventuell vorhandene Auslagerungen bleiben erhalten.
Bei der nächsten Reorganisation des Anwender-Spaces mit der Utility-Anweisung REORG SPACE werden alle existierenden Tabellen und Indizes auf dem Anwender-Space neu aufgebaut.
Damit verschwinden auch die nicht gelöschten Auslagerungen.
Informationen zur Speicherstruktur von Basistabellen finden Sie im „ Basishandbuch“.

Bei partitionierten Tabellen gilt die Klausel DEFERRED für alle Partitionen, sie kann nicht auf einzelne Partitionen eingeschränkt werden.

Die Klausel DEFERRED kann nur beim expliziten Löschen angegeben werden. Beim impliziten Löschen, z.B. mit DROP SPACE ... CASCADE, kann sie nicht angegeben werden.

Wenn DEFERRED nur für die Tabelle, nicht aber für die Indizes gelten soll, dann müssen die Indizes zuerst mit DROP INDEX (ohne Angabe von DEFERRED) gelöscht werden. Dann kann die Tabelle mit DROP TABLE ... DEFERRED gelöscht werden.

DEFERRED nicht angegeben:
SESAM/SQL löscht die Tabelle und alle zugehörigen Indizes. Bei sehr großen, zersplitterten Tabellen kann dies zeitaufwändig sein.

CASCADE

Die Basistabelle *tabelle* und alle zugehörigen Indizes werden gelöscht. Außerdem werdenalle Views, Routinen und Integritätsbedingungen, die sich direkt oder indirekt auf *tabelle* beziehen, gelöscht.

RESTRICT

Das Löschen der Basistabelle *tabelle* ist nur eingeschränkt möglich. Die Basistabelle *tabelle* kann nicht gelöscht werden, wenn sie in einer View-Definition, einer Routine oder in einer Integritätsbedingung einer anderen Basistabelle verwendet wird.

Beispiele

Das Beispiel löscht die Tabelle KUNDE nur dann, wenn zuvor alle Integritätsbedingungen anderer Basistabellen gelöscht wurden, die auf die Tabelle KUNDE verweisen. Außerdem darf die Tabelle KUNDE in keiner View-Definition verwendet werden.

```
ALTER TABLE kontakt DROP CONSTRAINT ko_knr_ref_kunde CASCADE

ALTER TABLE auftrag DROP CONSTRAINT a_knr_ref_kunde CASCADE

DROP TABLE kunde RESTRICT
```

Das Beispiel löscht die Tabellen BILDER und BESCHREIBUNG mit allen Indizies, Views und Integritätsbedingungen, die sich auf die Tabellen BILDER und BESCHREIBUNG beziehen.

```
DROP TABLE bilder CASCADE

DROP TABLE beschreibung CASCADE
```

Siehe auch

CREATE TABLE, ALTER TABLE

8.2.3.32 DROP USER - Berechtigungsschlüssel löschen

DROP USER löscht einen Berechtigungsschlüssel und die zugehörigen Systemzugänge. Ein Berechtigungsschlüssel kann nicht gelöscht werden, wenn er Eigentümer von Schemata, Spaces oder Storage Groups ist, wenn er Grantor eines Privilegs ist oder wenn momentan eine SQL-Transaktion des Berechtigungsschlüssels aktiv ist.

Der Berechtigungsschlüssel des universellen Benutzers kann nicht gelöscht werden.

Welche Berechtigungsschlüssel definiert sind, erfahren Sie im View USERS des INFORMATION_SCHEMA. Welcher Berechtigungsschlüssel Eigentümer ist, ist in den Views SCHEMATA, SPACES und STOGROUPS abgelegt. Ob der Berechtigungsschlüssel Grantor eines Privilegs ist, erfahren Sie aus den Views TABLE_PRIVILEGES, COLUMN_PRIVILEGES, USAGE_PRIVILEGES, CATALOG_PRIVILEGES und ROUTINE_PRIVILEGES (siehe Kapitel „Informationsschemata“).

Der aktuelle Berechtigungsschlüssel muss das Sonder-Privileg CREATE USER besitzen. Wenn ein Berechtigungsschlüssel mit dem Sonder-Privileg CREATE USER und mit GRANT-Berechtigung (siehe Abschnitt „GRANT - Privilegien vergeben“) gelöscht werden soll, so muss der aktuelle Berechtigungsschlüssel zusätzlich die GRANT-Berechtigung besitzen.

```
DROP USER berechtigungsschlüssel AT CATALOG catalog RESTRICT
```

berechtigungsschlüssel

Name des Berechtigungsschlüssels, der gelöscht werden soll.

AT CATALOG *catalog*

Name der Datenbank, aus der der Berechtigungsschlüssel gelöscht werden soll.

Siehe auch

CREATE USER, CREATE SYSTEM_USER, DROP SYSTEM_USER

8.2.3.33 DROP VIEW - View löschen

DROP VIEW löscht die Definition eines View.

Das Löschen einer View-Definition entzieht dem aktuellen Berechtigungsschlüssel alle Tabellen- und Spalten-Privilegien für diesen View. Tabellen- und Spalten-Privilegien des View, die weitergegeben wurden, werden ebenfalls entzogen.

Welche Views definiert sind, erfahren Sie im View VIEWS des INFORMATION_SCHEMA. Von welchen Tabellen Views abhängig sind, ist im View VIEW_TABLE_USAGE des INFORMATION_SCHEMA abgelegt (siehe Kapitel „Informationsschemata“).

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

```
DROP VIEW tabelle { CASCADE | RESTRICT }
```

tabelle

Name des View, der gelöscht werden soll.

CASCADE

Der View *tabelle* und alle Views und Routinen, die sich direkt oder indirekt auf *tabelle* beziehen, werden gelöscht.

RESTRICT

Das Löschen des View *tabelle* ist nur eingeschränkt möglich. Der View kann nicht gelöscht werden, wenn er in einer anderen View-Definition oder in einer Routine verwendet wird.

Siehe auch

CREATE VIEW

8.2.3.34 EXECUTE - Vorbereitete Anweisung ausführen

EXECUTE führt eine mit PREPARE vorbereitete Anweisung aus. Platzhalter für Eingabewerte in der dynamisch formulierten Anweisung werden durch aktuelle Werte ersetzt.

Ist die Anweisung eine SELECT-Anweisung, werden die Spaltenwerte des Ergebnissatzes in Benutzervariablen oder einem SQL-Deskriptorbereich abgelegt.

Ist die Anweisung eine CALL-Anweisung, dann werden Werte von Ausgabeparametern in Benutzervariablen oder einem SQL-Deskriptorbereich abgelegt.

Eine EXECUTE-Anweisung kann beliebig oft für eine mit PREPARE vorbereitete Anweisung ausgeführt werden.

Eine Anweisung kann mit EXECUTE nur in der Übersetzungseinheit ausgeführt werden, in der sie vorher mit PREPARE vorbereitet wurde.

`EXECUTE` *anweisungsbezeichner*

`[INTO` *deklaration* `]`

`[USING` *deklaration* `]`

deklaration `::= {` *variable* `| SQL DESCRIPTOR GLOBAL` *deskriptor* `}`

variable `::= :` *benutzervariable* `[[INDICATOR] :` *indikatorvariable* `]`
`[,:` *benutzervariable* `[[INDICATOR] :` *indikatorvariable* `]] ...`

---

anweisungsbezeichner

Bezeichner für die dynamisch formulierte Anweisung, die mit PREPARE vorbereitet wurde.

Enthält der Anweisungstext einen Cursornamen, muss die Cursorbeschreibung für diesen Cursor bei Ausführung der EXECUTE-Anweisung vorbereitet und der Cursor geöffnet sein.

INTO-Klausel

Angabe, wohin die Ausgabewerte der mit *anweisungsbezeichner* angegebenen dynamisch formulierten Anweisung gespeichert werden. Die INTO-Klausel muss in folgenden Fällen angegeben werden:

- die vorbereitete Anweisung ist eine SELECT-Anweisung
- die vorbereitete Anweisung ist eine CALL-Anweisung und die darin aufgerufene Prozedur hat Parameter vom Typ OUT oder INOUT

benutzervariable

Name einer Benutzervariable, der ein Ausgabewert zugewiesen wird.

Der Datentyp einer Benutzervariablen muss mit dem Datentyp des zugehörigen Ausgabewerts verträglich sein (siehe Abschnitt „Werte in Benutzervariable oder Deskriptorbereich lesen“).

Ist ein Ausgabewert ein Aggregat mit mehreren Elementen (SELECT-Anweisung), dann muss die zugehörige Benutzervariable ein Vektor mit derselben Anzahl von Elementen sein.
Die Anzahl der angegebenen Benutzervariablen muss zur Anzahl der Ausgabewerte der mit *anweisungsbezeichner* angegebenen SELECT-Anweisung passen.

Die Anzahl der angegebenen Benutzervariablen muss bei einem Prozeduraufruf mit der CALL-Anweisung zur Anzahl der Prozedurparameter vom Typ OUT oder IN-OUT der aufgerufenen Prozedur passen.

indikatorvariable

Name der Indikatorvariable für die vorangehende Benutzervariable.

Ist die Benutzervariable ein Vektor (SELECT-Anweisung), muss auch die Indikatorvariable ein Vektor mit derselben Anzahl von Elementen sein.

Die Indikatorvariable zeigt an, ob der NULL-Wert übertragen wurde oder Datenverlust aufgetreten ist:

0	Die Benutzervariable enthält den gelesenen Wert. Die Zuweisung war fehlerfrei.
-1	Der Wert, der zugewiesen werden sollte, ist der NULL-Wert.
>0	Bei alphanumerischen Werten und National-Werten: Der Benutzervariable wurde eine verkürzte Zeichenkette zugewiesen. Der Wert der Indikatorvariable gibt die Originallänge in Code Units an.

deskriptor

Name eines SQL-Deskriptorbereichs, der die Datentypbeschreibung für die Ausgabewerte enthält und in den bei Ausführung der mit *anweisungsbezeichner* angegebenen Anweisung die Ausgabewerte (für Prozeduren die Prozedurparameter vom Typ OUT oder INOUT) geschrieben werden.

Der SQL-Deskriptorbereich muss vorher angelegt und geeignet belegt worden sein:

- Der Wert des Felds COUNT muss mit der Anzahl der Ausgabewerte der mit *anweisungsbezeichner* angegebenen Anweisung übereinstimmen (bei Aggregaten ein Ausgabewert pro Element, bei Prozeduren ein Ausgabewert pro Prozedurparameter vom Typ OUT oder INOUT) und es muss gelten:

  0 <= COUNT <= festgelegte Maximalanzahl von Deskriptorbereichseinträgen
- Die Ausgabewerte werden den DATA-Feldern der Deskriptorbereichseinträge in der Reihenfolge der Einträge im Deskriptorbereich zugewiesen. Die Datentypbeschreibung eines Eintrags muss mit dem Datentyp des zugehörigen Ausgabewerts verträglich sein (siehe Abschnitt „Werte in Benutzervariable oder Deskriptorbereich lesen“).

  Ist ein zu übertragender Wert NULL, wird das entsprechende INDICATOR-Feld auf den Wert -1 gesetzt. Wird eine zuzuweisende Zeichenkette abgeschnitten, gibt das entsprechende INDICATOR-Feld die Originallänge an.

USING-Klausel

Angabe, woher die Eingabewerte für die mit *anweisungsbezeichner* angegebene dynamisch formulierte Anweisung gelesen werden. Die INTO-Klausel muss in folgenden Fällen angegeben werden:

- wenn die SELECT-Anweisung Fragezeichen als Platzhalter für Werte enthält
- wenn die in der CALL-Anweisung aufgerufene Prozedur Parameter vom Typ IN oder INOUT besitzt und die korresponierenden Argumente Fragezeichen als Platzhalter für Werte enthalten

benutzervariable

Name einer Benutzervariablen, deren Wert einem Platzhalter in der dynamisch formulierten Anweisung *anweisungsbezeichner* zugewiesen wird.

Der Datentyp einer Benutzervariablen muss mit dem Datentyp des zugehörigen Platzhalters verträglich sein (siehe Abschnitt „Werte für Platzhalter“).

Steht der Platzhalter für ein Aggregat mit mehreren Elementen (SELECT-Anweisung), dann muss die zugehörige Benutzervariable ein Vektor mit derselben Anzahl von Elementen sein.
Die Anzahl der angegebenen Benutzervariablen muss mit der Anzahl der Platzhalter in der SELECT-Anweisung übereinstimmen.

Die Anzahl der angegebenen Benutzervariablen muss bei einem Prozeduraufruf mit der CALL-Anweisung zur Anzahl der Platzhalter für Parameter vom Datentyp IN oder INOUT passen.

Die Benutzervariablen werden in der angegebenen Reihenfolge den Platzhaltern in der dynamisch formulierten Anweisung zugewiesen.

indikatorvariable

Name der Indikatorvariable für die vorangehende Benutzervariable.

Ist die Benutzervariable ein Vektor (SELECT-Anweisung), muss auch die Indikatorvariable ein Vektor mit derselben Anzahl von Elementen sein.

Der Wert der Indikatorvariable zeigt an, ob der NULL-Wert übertragen werden soll:

< 0	Der NULL-Wert soll zugewiesen werden.
>= 0	Der Wert der Benutzervariable soll zugewiesen werden.

deskriptor

Name eines SQL-Deskriptorbereichs, der die Datentypen und Werte für die Platzhalter in der dynamisch formulierten Anweisung *anweisungsbezeichner* enthält.

Der SQL-Deskriptorbereich muss vorher angelegt und geeignet belegt worden sein:

- Der Wert des Deskriptorbereichsfelds COUNT muss mit der Anzahl der benötigten Eingabewerte übereinstimmen (bei Aggregaten ein Eingabewert pro Element, bei Prozeduren ein Ausgabewert pro Prozedurparameter vom Typ OUT
oder INOUT), und es muss gelten:

0 <= COUNT <= festgelegte Maximalanzahl von Deskriptorbereichseinträgen

- Die Werte der DATA-Felder der Deskriptorbereichseinträge (bzw. NULL-Werte bei negativem INDICATOR) werden in der Reihenfolge der Einträge im Deskriptorbereich den Platzhaltern in der dynamisch formulierten Anweisung zugewiesen.
  Die Datentypbeschreibung eines Eintrags muss mit dem Datentyp des zugehörigen Platzhalters verträglich sein (siehe Abschnitt „Werte für Platzhalter“).

Beispiel

```
EXECUTE dyn_statement

INTO SQL DESCRIPTOR GLOBAL 'DESKR_BEREICH'
```

Siehe auch

EXECUTE IMMEDIATE, PREPARE, SELECT

8.2.3.35 EXECUTE IMMEDIATE - Dynamisch formulierte Anweisung ausführen

Mit der Anweisung EXECUTE IMMEDIATE wird eine dynamisch formulierte Anweisung in einem Schritt vorbereitet und ausgeführt. EXECUTE IMMEDIATE entspricht also einer PREPARE-Anweisung mit unmittelbar darauffolgender EXECUTE-Anweisung. Allerdings bleibt die Anweisung nicht vorbereitet und kann nicht mit EXECUTE nochmals ausgeführt werden.

Dynamisch formulierte CALL-Anweisungen können mit EXECUTE IMMEDIATE ausgeführt werden, wenn die aufzurufende Prozedur keine Prozedurparameter oder nur Prozedurparameter vom Typ IN besitzt.

`EXECUTE IMMEDIATE` *anweisungsvariable*

anweisungsvariable `::= :` *benutzervariable*

---

anweisungsvariable

Alphanumerische Benutzervariable, die den Anweisungstext enthält. Für Benutzervariable ist auch ein Datentyp CHAR(*n*) zulässig, mit 256 <= *n* <= 32000.

Folgende Bedingungen müssen erfüllt sein:

- Innerhalb des Anweisungstexts dürfen keine Benutzervariablen und keine Fragezeichen als Platzhalter für unbekannte Werte verwendet werden.
- Der Anweisungstext darf weder SQL-Kommentare noch Kommentare der Wirtssprache enthalten. Ausnahmen sind Pragmas (--%PRAGMA).
- Der Anweisungstext darf keine SELECT-Anweisung und keine Cursorbeschreibung sein.
- Bei einer INSERT-Anweisung darf die RETURN INTO-Klausel nicht angegeben werden.
- Enthält der Anweisungstext einen Cursornamen (DELETE WHERE CURRENT OF, UPDATE WHERE CURRENT OF), muss die Cursorbeschreibung für diesen Cursor bei Ausführung der EXECUTE IMMEDIATE-Anweisung vorbereitet und der Cursor geöffnet sein.

Anweisungen für EXECUTE IMMEDIATE

Folgende Anweisungen können mit EXECUTE IMMEDIATE ausgeführt werden:

ALTER SPACE	DROP SCHEMA
ALTER STOGROUP	DROP SPACE
ALTER TABLE	DROP STOGROUP
COMMIT	DROP SYSTEM_USER
CALL (nur Eingabeparameter; Typ IN)	DROP TABLE
CREATE INDEX	DROP USER
CREATE FUNCTION	DROP VIEW
CREATE PROCEDURE	GRANT
CREATE SCHEMA	INSERT (ohne RETURN INTO-Klausel)

CREATE SPACE	MERGE
CREATE STOGROUP	PERMIT
CREATE SYSTEM_USER	REORG STATISTICS
CREATE TABLE	REVOKE
CREATE USER	ROLLBACK
CREATE VIEW	SET CATALOG
DELETE	SET SCHEMA
DROP FUNCTION	SET SESSION AUTHORIZATION
DROP INDEX	SET TRANSACTION
DROP PROCEDURE	UPDATE

Daneben können auch alle Utility-Anweisungen mit EXECUTE IMMEDIATE ausgeführt werden (siehe Handbuch „ SQL-Sprachbeschreibung Teil 2: Utilities“).

Folgende Anweisungen können nicht mit EXECUTE IMMEDIATE ausgeführt werden:

ALLOCATE DESCRIPTOR	INCLUDE
CLOSE	OPEN
DEALLOCATE DESCRIPTOR	PREPARE
DECLARE CURSOR	RESTORE
DESCRIBE	SELECT
EXECUTE	SET DESCRIPTOR
EXECUTE IMMEDIATE	STORE
FETCH	WHENEVER
GET DESCRIPTOR	

Beispiel

Zur Laufzeit soll eine SQL-Anweisung mit EXECUTE IMMEDIATE übersetzt und ausgeführt werden:

In SOURCESTMT wird die folgende SQL-Anweisung als alphanumerische Zeichenkette eingelesen:

```
CREATE TABLE auftragkunden.auftragsver.aufstat
(astnr INTEGER, astxt CHAR(15))
```

Die Anweisung wird übersetzt und ausgeführt mit:

```
EXEC SQL EXECUTE IMMEDIATE :SOURCESTMT END-EXEC
```

Siehe auch

EXECUTE, PREPARE

8.2.3.36 FETCH - Cursor positionieren und Satz lesen

FETCH positioniert einen Cursor. Die neue Cursorposition ist entweder auf einem Satz, vor dem ersten Satz oder hinter dem letzten Satz der Cursortabelle. Ist die neue Cursorposition auf einem Satz der Cursortabelle, wird dieser Satz zum aktuellen Satz, und die Spaltenwerte dieses Satzes werden gelesen.
Wird bei FETCH kein Satz gelesen, weil die angegebene Position nicht existiert, wird ein entsprechender SQLSTATE gesetzt, der mit WHENEVER NOT FOUND behandelt werden kann.
Ein mit SCROLL vereinbarter Cursor kann mit FETCH auf einen beliebigen Satz der Cursortabelle positioniert werden. Ein mit NO SCROLL vereinbarter Cursor kann nur auf den jeweils nächsten Satz positioniert werden (FETCH NEXT...).

Die Werte des aktuellen Satzes können in Benutzervariable, Prozedurparameter vom Typ INOUT oder OUT, lokale Variable oder in einen SQL-Deskriptorbereich übertragen werden.

Die Cursorvereinbarung mit DECLARE CURSOR muss in derselben Übersetzungseinheit im Programmtext vor der FETCH-Anweisung stehen.

Zum Zeitpunkt des FETCH darf kein mit einer STORE-Anweisung erstellter Sicherungsstand des Cursors existieren. Der Cursor muss geöffnet sein.

Falls der Cursor mit WITH HOLD deklariert wurde, muss der Isolationslevel bzw. der Konsistenzlevel der Transaktion derselbe sein, wie beim Öffnen des Cursors.

Ist für den Cursor der Schubmodus eingeschaltet (siehe Abschnitt „Pragma PREFETCH“) und wurde für den Cursor bereits eine Anweisung FETCH NEXT... ausgeführt, so ist nachfolgend für diesen Cursor nur noch diese FETCH NEXT-Anweisung erlaubt, d.h., dieselbe Anweisung in einer Schleife oder einem Unterprogramm.

```
FETCH { [NEXT] | PRIOR | FIRST | LAST | RELATIVE n | ABSOLUTE n }
      [FROM] cursor
      { INTO variable ,... | SQL DESCRIPTOR GLOBAL deskriptor }

n ::= { ganzzahl | :benutzervariable | routinenparameter | lokale_variable }

variable ::=
{
    :benutzervariable [[INDICATOR] :indikatorvariable ] |
    routinenparameter |
    lokale_variable
}
```

NEXT

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

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

PRIOR

Positioniert den Cursor auf den vorhergehenden Satz der Cursortabelle.

Steht der Cursor auf dem ersten Satz der Cursortabelle, wird er vor den ersten Satz positioniert. Steht er bereits vor dem ersten Satz, bleibt die Position unverändert. PRIOR ist nur zulässig, wenn Sie den Cursor mit SCROLL vereinbart haben.

FIRST

Positioniert den Cursor auf den ersten Satz der Cursortabelle bzw. vor den ersten Satz, wenn die Cursortabelle leer ist.
FIRST ist nur zulässig, wenn Sie den Cursor mit SCROLL vereinbart haben.

LAST

Positioniert den Cursor auf den letzten Satz der Cursortabelle bzw. hinter den letzten Satz, wenn die Cursortabelle leer ist.
LAST ist nur zulässig, wenn Sie den Cursor mit SCROLL vereinbart haben.

ABSOLUTE *n*

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

Für *n* können Sie Folgendes angeben:

- eine Ganzzahl
- eine Benutzervariable (wenn die Anweisung **nicht** Bestandteil einer Prozedur ist) vom SQL-Datentyp INT oder SMALLINT
- einen Routinen-Parameter oder eine lokale Variable (wenn die Anweisung Bestandteil einer Routine ist) vom SQL-Datentyp INT oder SMALLINT

Die Cursorposition wird abhängig vom Wert für *n* wie folgt bestimmt:

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

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

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

Beispiel

```
FETCH ABSOLUTE -1
```

und

```
FETCH  LAST
```

sind äquivalent.

RELATIVE *n*

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

Für *n* können Sie Folgendes angeben:

- ein ganzzahliges Literal
- eine Benutzervariable (wenn die Anweisung **nicht** Bestandteil einer Prozedur ist) vom SQL-Datentyp INT oder SMALLINT
- einen Routinen-Parameter oder eine lokale Variable (wenn die Anweisung Bestandteil einer Routine ist) vom SQL-Datentyp INT oder SMALLINT

Die Cursorposition wird abhängig vom Wert für *n* wie folgt bestimmt:

>0	Der Cursor wird auf den Satz positioniert, der *n* Sätze hinter seiner aktuellen Position liegt. Ist die neu Position größer als die Anzahl der Sätze der Cursortabelle, wird der Cursor hinter den letzten Satz positioniert.
0	Die Cursorposition bleibt unverändert.
<0	Der Cursor wird auf den Satz positioniert, der *n* Sätze vor seiner aktuellen Position liegt. Wird die neu Position <= 1, wird der Cursor vor den ersten Satz positioniert.

FROM *cursor*

Name des Cursors.

INTO-Klausel

Angabe, wohin die gelesenen Werte gespeichert werden.

:benutzervariable, *routinenparameter*, *lokale_variable*

Name einer Benutzervariablen (wenn die Anweisung **nicht** Bestandteil einer Prozedur ist) bzw. Name eines Prozedurparameters von Typ INOUT oder OUT oder einer lokalen Variablen (wenn die Anweisung Bestandteil einer Routine ist). Der Spaltenwert des Ergebnissatzes wird dem angegebenen Ausgabeziel zugewiesen.

Der Datentyp muss mit dem Datentyp des zugehörigen Ausgabewerts verträglich sein (siehe Abschnitt „Werte in Benutzervariable oder Deskriptorbereich lesen“). Ist ein Ausgabewert ein Aggregat mit mehreren Elementen (nur bei Benutzervariable), muss die zugehörige Benutzervariable ein Vektor mit derselben Anzahl von Elementen sein.

Die Anzahl der angegebenen Elemente muss mit der Anzahl der Spalten in der SELECT-Liste der Cursorbeschreibung übereinstimmen. Der Wert der i-ten Spalte in der SELECT-Liste wird dem i-ten Ausgabeziel in der INTO-Klausel zugewiesen.

indikatorvariable

Name der Indikatorvariable, die zu der vorangehenden Benutzervariable gehört. Ist die Benutzervariable ein Vektor, muss auch die Indikatorvariable ein Vektor mit derselben Elementanzahl sein.

Die Indikatorvariable zeigt an, ob der NULL-Wert übertragen wurde oder Datenverlust aufgetreten ist:

0	Die Benutzervariable enthält den gelesenen Wert. Die Zuweisung war fehlerfrei.
-1	Der Wert, der zugewiesen werden sollte, ist der NULL-Wert.
>0	Bei alphanumerischen Werten oder National-Werten: Der Benutzervariable wurde eine verkürzte Zeichenkette zugewiesen. Der Wert der Indikatorvariable gibt die Originallänge in Code Units an.

deskriptor

Für dynamischen Cursor.

Name eines SQL-Deskriptorbereichs, der die Datentypbeschreibung für die Ausgabewerte enthält und in den die mit FETCH gelesenen Ausgabewerte geschrieben werden.

Der SQL-Deskriptorbereich muss vorher angelegt und geeignet belegt worden sein:

- Der Wert des Felds COUNT muss mit der Anzahl der Ausgabewerte übereinstimmen, die sich wie folgt berechnet: Anzahl der einfachen Ergebnisspalten plus Anzahl der Spaltenelemente von jeder multiplen Ergebnisspalte. Zusätzlich muss gelten:

  0 <= COUNT <= festgelegte Maximalanzahl von Deskriptorbereichseinträgen
- Die Ausgabewerte werden den DATA-Feldern der Deskriptorbereichseinträge in der Reihenfolge der Einträge im Deskriptorbereich zugewiesen. Die Datentypbeschreibung eines Eintrags muss mit dem Datentyp des zugehörigen Ausgabewerts verträglich sein (siehe Abschnitt „Werte in Benutzervariable oder Deskriptorbereich lesen“).

  Ist ein zu übertragender Wert NULL, wird das entsprechende INDICATOR-Feld auf den Wert -1 gesetzt. Wird eine zuzuweisende Zeichenkette abgeschnitten, gibt das entsprechende INDICATOR-Feld die Originallänge an.

  Ist für den geöffneten Cursor *cursor* der Schubmodus eingeschaltet, und wurde für *cursor* bereits eine Anweisung FETCH NEXT... ausgeführt, deren INTO-Klausel den Namen eines anderen SQL-Deskriptorbereichs enthält, so erhalten Sie eine Fehlermeldung.

Verhalten von SESAM/SQL im Fehlerfall

Bei Fehlern beim Lesen der Werte (z.B. Wert ist der NULL-Wert, aber Indikatorvariable ist nicht angegeben; numerischer Wert für Zieldatentyp zu groß) hat der Cursor die neue Position, aber die zugewiesenen Werte sind undefiniert.

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

Beispiele

Das folgende Beispiel positioniert den Cursor CUR_AUFTRAG auf einen Satz der Cursortabelle und liest die Spaltenwerte des aktuellen Satzes in die Benutzervariablen ANR, KNR, KONR, ATEXT, FERTIGSOLL und ASTNR.
Mit den Indikatorvariablen INDKONR, INDATEXT und INDFERTIGSOLL wird geprüft, ob bei der Übertragung der alphanumerischen Werte Informationsverlust aufgetreten ist und ob eine Spalte den NULL-Wert enthält.

```
FETCH cur_auftrag
INTO  :ANR,
      :KNR,
      :KONR       INDICATOR :INDKONR,
      :ATEXT      INDICATOR :INDATEXT,
      :FERTIGSOLL INDICATOR :INDFERTIGSOLL,
      :ASTNR
```

Das folgende Beispiel positioniert den Cursor CUR_ERGEBNIS auf einen Satz der Cursortabelle und liest die Spaltenwerte in den Deskriptorbereich DESKR_BEREICH.

```
FETCH cur_ergebnis INTO SQL DESCRIPTOR GLOBAL 'DESKR_BEREICH'
```

Siehe auch

CLOSE, DECLARE CURSOR, DELETE, OPEN, STORE, UPDATE

8.2.3.37 FOR - SQL-Anweisungen in einer Schleife ausführen

Die FOR-Anweisung führt SQL-Anweisungen in einer Schleife über alle Sätze eines implizit definierten Cursors aus. Cursor-Operationen (z.B. FETCH) werden dabei nicht benötigt. Sie dürfen auch nicht auf den implizit definierten Cursor angewendet werden. Der implizit definierte Cursor wird am Ende der Bearbeitung automatisch geschlossen.

Mit der ITERATE-Anweisung kann sofort zum nächsten Schleifendurchlauf gewechselt werden. Die Schleife kann über eine LEAVE-Anweisung abgebrochen werden.

Die FOR-Anweisung darf nur in einer Routine angegeben werden, d.h. im Rahmen einer CREATE PROCEDURE- oder CREATE FUNCTION-Anweisung. Routinen und ihre Verwendung in SESAM/SQL sind detailliert im Kapitel „Routinen“ beschrieben.

Die FOR-Anweisung ist eine nicht-atomare SQL-Anweisung, d.h. in ihr können weitere (atomare oder nicht-atomare) SQL-Anweisungen auftreten.

Wenn die FOR-Anweisung Teil einer COMPOUND-Anweisung ist, dann kann die Schleife bei entsprechenden Fehlerbehandlungsroutinen auch beim Auftreten eines bestimmten SQLSTATE (z.B. keine Daten, Klasse '02xxx') verlassen werden.

```
[ marke :]
FOR [ forloopname AS] [ cursor CURSOR FOR] abfrageausdruck
    DO routine_sql_anweisung; [ routine_sql_anweisung; ]...
END FOR [ marke ]

forloopname ::= einf_name
```

---

marke

Die Marke vor der FOR-Anweisung (Anfangsmarke) bezeichnet den Anfang der Schleife. Sie darf nicht identisch sein mit einer anderen Marke innerhalb der Schleife.

Die Anfangsmarke muss nur dann angegeben werden, wenn mit ITERATE zum nächsten Schleifendurchlauf gewechselt werden soll oder wenn die Schleife über eine LEAVE-Anweisung verlassen werden soll. Sie sollte aber stets verwendet werden, damit SESAM/SQL die korrekte Struktur der Routine überprüfen kann (z.B. bei geschachtelten Schleifen).

Die Marke am Ende der FOR-Anweisung (Endemarke) bezeichnet das Ende der Schleife. Wenn die Endemarke angegeben ist, dann muss auch die Anfangsmarke angegeben sein. Beide Marken müssen identisch sein.

forloopname

Name der FOR-Schleife. Er kann als Qualifikation für die Namen der Spalten der nachfolgenden Cursor-Beschreibung verwendet werden.
forloopname darf max. 31 Zeichen lang sein.

cursor

Optionaler Name für den durch *abfrageausdruck* definierten Cursor.
Dieser Name muss angegeben werden, wenn für den Cursor die Funktion UPDATE ... WHERE CURRENT OF ... bzw. DELETE ... WHERE CURRENT OF ... verwendet werden soll.

abfrageausdruck

Definition des Cursors, der durch die FOR-Anweisung abgearbeitet werden soll.Der Cursor muss eindeutig benannte Spalten haben. Dies kann durch Verwendung von Korrelationsnamen immer erreicht werden.
Die Datentypen der Ausgabewerte des Cursors dürfen nicht multipel sein. Es können aber einzelne Ausprägungen eines multiplen Feldes verwendet werden.

routine_sql_anweisung

SQL-Anweisung, die in der FOR-Anweisung ausgeführt werden soll.
Eine SQL-Anweisung wird mit „;“ (Semikolon) abgeschlossen.
Mehrere SQL-Anweisungen können nacheinander angegeben werden. Sie werden in der angegebenen Reihenfolge ausgeführt.
Vor der Durchführung einer SQL-Anweisung werden keine Privilegien geprüft.Eine SQL-Anweisung in einer Routine darf auf die Parameter der Routine und (wenn die Anweisung Teil einer COMPOUND-Anweisung ist) auf lokale Variablen, jedoch nicht auf Benutzervariablen zugreifen.

Syntax und Bedeutung von *routine_sql_anweisung* sind zentral im Abschnitt „SQL-Anweisungen in Routinen“ beschrieben. Die dort genannten SQL-Anweisungen dürfen verwendet werden.

> ¡ Die SQL-Anweisungen *update_positioned_anweisung* und *delete_positioned_anweisung* können auch für den entsprechenden Cursor ausgeführt werden, wenn *cursor* angegeben ist und der *abfrageausdruck* änderbar ist (siehe Abschnitt „Regeln für änderbare Abfrage-Ausdrücke“).

Ausführungshinweise

Die FOR-Anweisung ist eine nicht-atomare Anweisung:

- Wenn die FOR-Anweisung Teil einer COMPOUND-Anweisung ist, dann gelten die dort beschriebenen Regeln, insbesondere die dort definierten Fehler-Routinen.
- Wenn die FOR-Anweisung **nicht** Teil einer COMPOUND-Anweisung ist und eine der SQL-Anweisungen einen SQLSTATE meldet, dann werden ggf. nur die Änderungen dieser Anweisung rückgängig gemacht. Die FOR-Anweisung und die Routine, in der sie enthalten ist, werden abgebrochen. Die SQL-Anweisung, in der die Routine verwendet wurde, liefert den betreffenden SQLSTATE zurück.

Gültigkeitsbereiche bzw. Vorrangregeln für Namen:

- Bei einfachen Namen (*einf_name*) wird zunächst ein existierender Routinenparameter oder eine existierende lokale Variable mit diesem Namen verwendet. Andernfalls wird der Name in der aktuellen Anweisung gesucht. Gibt es den Namen auch dort nicht, dann wird der Name (bei geschachtelten FOR-Anweisungen) in den übergeordneten FOR-Anweisungen „von innen nach außen“ gesucht.
- Es wird empfohlen, einen Namen für die FOR-Schleife (*forloopname*) zu definieren, siehe unten. Damit werden Namensbezüge innerhalb von FOR-Schleifen deutlich gemacht. Vorrangregeln müssen dann nicht beachtet werden.

Name für eine FOR-Schleife:

In den SQL-Anweisungen der FOR-Anweisung kann über die Spaltennamen der Cursor-Beschreibung auf die aktuellen Werte Bezug genommen werden.

Deutlicher ist es aber, einen Namen für die FOR-Schleife (*forloopname*) zu definieren.
Dieser Name kann zur Qualifikation für die Spalten des aktuellen Satzes verwendet werden:

```
FOR F1 AS SELECT C001, C002 FROM T1 WHERE P < 127
DO
   UPDATE TU
   SET COLX = COLX + F1.C001 WHERE COLY = F1.C002;
END FOR
```

Anschaulich wird dies in einer geschachtelten FOR-Anweisung:

```
FOR F1 AS SELECT C001 FROM T1 WHERE P < 127
DO
    FOR F2 AS SELECT C001, C002 FROM T2 WHERE Q < 875
    DO
        UPDATE TU
               SET COLX = COLX + F1.C001 + F2.C001
               WHERE COLY < F2.C002;
    END FOR;
END FOR
```

Siehe auch

CREATE PROCEDURE, CREATE FUNCTION, ITERATE, LEAVE

8.2.3.38 GET DIAGNOSTICS - Diagnoseinformationen ausgeben

GET DIAGNOSTICS ermittelt Informationen zu einer zuvor in einer Routine ausgeführten SQL-Anweisung und trägt diese in einen Prozedurparameter vom Typ INOUT oder OUT oder eine lokale Variable ein. Die Informationen beziehen sich auf die Anweisung selbst oder auf die davon betroffenen Objekte der Datenbank.

GET DIAGNOSTICS ändert weder Inhalt noch Reihenfolge von Diagnosebereichen. D.h. aufeinander folgende GET DIAGNOSTICS-Anweisungen werten dieselbe Diagnoseinformation aus.

GET DIAGNOSTICS ist eine der Diagnoseanweisungen in Routinen, siehe „Diagnoseinformationen in Routinen“.

```
GET [CURRENT | STACKED] DIAGNOSTICS
    { anweisungs_info [, anweisungs_info ] ...] |
      CONDITION bedingungs_info [, bedingungs_info ]... }

anweisungs_info ::= name1 = ROW_COUNT

bedingungs_info ::= name2 =
{
   CONDITION_IDENTIFIER |
   RETURNED_SQLSTATE |
   MESSAGE_TEXT |
   MESSAGE_LENGTH |
   MESSAGE_OCTET_LENGTH
}

name1, name2 ::= { lokale_variable | routinenparameter }
```

CURRENT

Die Diagnoseinformation für die zuletzt ausgeführte SQL-Anweisung wird ausgegeben.

Mit dieser Anweisung wird normalerweise die Diagnoseinformation einer fehlerfrei ausgeführten SQL-Anweisung ausgegeben.

Die SQL-Anweisung kann aber auch nach einem SQLSTATE eine Fehler-Routine mit der Fehlerbehandlung CONTINUE durchlaufen haben (siehe „Lokale Fehler-Routinen“) und GET DIAGNOSTICS ist die nächste Anweisung in der Routine.Eine lokale Fehler-Routine hat einen eigenen Diagnosebereich. Mit CURRENT wird die Diagnoseinformation der zuletzt in der Fehler-Routine ausgeführten SQL-Anweisung ausgegeben. Die Diagnoseinformation der auslösenden SQL-Anweisung wird mit STACKED ausgegeben.

STACKED

Die Diagnoseinformation der SQL-Anweisung, deren SQLSTATE die Fehler-Routine ausgelöst hat, wird ausgegeben.
STACKED darf nur in einer lokalen Fehler-Routine angegeben werden.

name1, *name2*

name1 und *name2* bezeichnen lokale Variable, Prozedur- oder UDF-Parameter in die die nach dem Gleichheitszeichen beschriebene Information eingetragen wird. Der Datentyp von *name1* bzw. *name2* muss mit dem Datentyp der einzutragenden Information verträglich sein. Es gelten die Regeln im Abschnitt „Werte in Prozedurparameter (Ausgabe) oder lokale Variable eintragen“.

name1=ROW_COUNT

name1 wird die Anzahl der bearbeiteten Sätze einer der folgenden erfolgreich ausgeführten SQL-Anweisung zugewiesen: *insert_anweisung*, *update_searched_anweisung*, *delete_searched_anweisung*, *merge_anweisung* . Sonst ist der Wert undefiniert.

Datentyp: DECIMAL(31)

name2=CONDITION_IDENTIFIER

name2 wird ggf. der Name der durch eine SIGNAL- oder RESIGNAL-Anweisung gemeldeten Bedingung zugewiesen. Sonst wird eine Zeichenkette der Länge 0 zugewiesen.

Datentyp: VARCHAR(31)

name2=RETURNED_SQLSTATE

name2 wird ggf. der Wert des gemeldeten SQLSTATEs zugewiesen. Sonst wird eine Zeichenkette der Länge 0 zugewiesen.

Datentyp: VARCHAR(5)

name2=MESSAGE_TEXT

name2 wird ggf. der Meldungstext zugewiesen wenn in der SIGNAL- oder RESIGNAL-Anweisung MESSAGE_TEXT angegeben wurde. Sonst wird eine Zeichenkette der Länge 0 zugewiesen.

Datentyp: VARCHAR(120)

name2=MESSAGE_LENGTH

name2 wird ggf. die Länge des Meldungstexts zugewiesen wenn in der SIGNAL- oder RESIGNAL-Anweisung MESSAGE_TEXT angegeben wurde. Sonst wird der Wert 0 zugewiesen.

Datentyp: INTEGER

name2=MESSAGE_OCTET_LENGTH

name2 wird ggf. die Länge des Meldungstexts in Byte zugewiesen wenn in der SIGNAL- oder RESIGNAL-Anweisung MESSAGE_TEXT angegeben wurde. Sonst wird der Wert 0 zugewiesen.

Datentyp: INTEGER

Beispiele (siehe auch "Diagnoseinformationen in Routinen" **)**

Diagnoseinformationen der letzten SQL-Anweisung ausgeben:

```
GET CURRENT DIAGNOSTICS counter1=ROW_COUNT;
```

Diagnoseinformationen der SQL-Anweisung, die die Fehler-Routine ausgelöst hat, ausgeben::

```
GET STACKED DIAGNOSTICS CONDITION
    var1=RETURNED_SQLSTATE,
    var2=MESSAGE_LENGTH, var3=MESSAGE_TEXT;
```

Siehe auch

COMPOUND, CREATE FUNCTION, CREATE PROCEDURE, RESIGNAL, SIGNAL

8.2.3.39 GET DESCRIPTOR - SQL-Deskriptorbereich lesen

GET DESCRIPTOR liest die Werte von Feldern eines SQL-Deskriptorbereichs.

Der Deskriptorbereich muss mit ALLOCATE DESCRIPTOR angelegt und vor dem GET DESCRIPTOR-Aufruf belegt worden sein.

```
GET DESCRIPTOR GLOBAL deskriptor
    { : benutzervariable =COUNT |
      VALUE eintragsnummer : benutzervariable = feldbezeichner
      [, benutzervariable = feldbezeichner ]...}

eintragsnummer ::= { ganzzahl | : benutzervariable }

feldbezeichner ::=
{
   REPETITIONS |
   TYPE |
   DATETIME_INTERVAL_CODE |
   PRECISION |
   SCALE |
   LENGTH |
   INDICATOR |
   DATA |
   OCTET_LENGTH |
   NULLABLE |
   NAME |
   UNNAMED
}
```

---

deskriptor

Name des SQL-Deskriptorbereichs, dessen Einträge gelesen werden sollen.

benutzervariable=COUNT

Benutzervariable vom SQL-Datentyp SMALLINT, in die der Wert des Deskriptorbereichsfelds COUNT eingetragen wird.

eintragsnummer

Nummer des Eintrags im SQL-Deskriptorbereich, von dem Felder gelesen werden sollen. Die Einträge im Deskriptorbereich sind beginnend mit 1 durchnummeriert. Für *eintragsnummer* können Sie eine Ganzzahl oder eine Benutzervariable angeben: 1 <= *eintragsnummer* <= festgelegte Maximalanzahl von Deskriptorbereichseinträgen

Bei *eintragsnummer* > COUNT wird ein entsprechender SQLSTATE gesetzt, der mit WHENEVER NOT FOUND behandelt werden kann.

benutzervariable=feldbezeichner

Benutzervariable, in die der Wert des angegebenen Deskriptorbereichsfelds des Eintrags *eintragsnummer* eingetragen wird. Der SQL-Datentyp der Variable hängt vom angegebenen Feldbezeichner ab.

feldbezeichner

Feld des Eintrags *eintragsnummer*, das gelesen werden soll. Die Deskriptorbereichsfelder sind im Abschnitt „Deskriptorbereichsfelder“ beschrieben. Ein *feldbezeichner* darf innerhalb einer GET DESCRIPTOR-Anweisung mehrfach vorkommen.

Bei der Übertragung eines Werts von einem Deskriptorbereichsfeld in eine Benutzervariable muss die Benutzervariable bei allen Feldern außer bei NAME und DATA den SQL-Datentyp SMALLINT haben.

Zum Übertragen des Werts von NAME muss die Benutzervariable den SQL-Datentyp CHAR(*n*) oder VARCHAR(*n*) haben, mit *n* >= 128.
Zum Übertragen des Werts von DATA muss die Benutzervariable genau den SQL-Datentyp haben, der durch die Felder TYPE, DATETIME_INTERVAL_CODE, LENGTH, PRECISION, SCALE desselben Eintrags festgelegt ist (siehe Abschnitt „Werte zwischen Benutzervariablen und Deskriptorbereich übertragen“).

Außer für die Felder DATA und INDICATOR dürfen keine Vektoren angegeben werden. Wenn DATA und INDICATOR angegeben sind, müssen beide einfach oder Vektoren mit gleicher Anzahl von Elementen sein.

Ist ein Vektor mit mehreren Elementen angegeben, dann müssen bei genau so vielen nachfolgenden Einträgen die Eintragsnummern <= der festgelegten Maximalanzahl von Deskriptorbereichseinträgen sein. Sind Eintragsnummern > COUNT, wird ein entsprechender SQLSTATE gesetzt, der mit WHENEVER NOT FOUND behandelt werden kann.

GET DESCRIPTOR liest den Wert des angegebenen Felds, der zuletzt gesetzt wurde. Ist der Wert des Felds undefiniert, ist auch der zurückgelieferte Wert nicht definiert.

Für das Feld DATA gilt zusätzlich: Ist der Wert des Felds INDICATOR desselben Eintrags < 0, muss die GET DESCRIPTOR-Anweisung das INDICATOR-Feld auch enthalten und es wird nur das INDICATOR-Feld zugewiesen.

Sind Vektoren angegeben, werden entsprechend viele Einträge ab *eintragsnummer* gelesen.

Examples

Name, Typ und Länge in Byte des dritten Eintrages im SQL-Deskriptorbereich DEMO_DESC lesen:

```
GET DESCRIPTOR GLOBAL :demo_desc

VALUE 3 :desc_name = NAME :desc_type = TYPE :desc_len = OCTET_LENGTH
```

Anzahl der Einträge im SQL-Deskriptorbereich abfragen:

```
GET DESCRIPTOR GLOBAL :demo_desc :desc_count = COUNT
```

Siehe auch

ALLOCATE DESCRIPTOR, DEALLOCATE DESCRIPTOR, DESCRIBE, SET DESCRIPTOR

8.2.3.40 GRANT - Privilegien vergeben

GRANT vergibt folgende Privilegien:

- Tabellen- und Spalten-Privilegien für Basistabellen und Views
- Sonder-Privilegien für Datenbanken und Storage Groups
- EXECUTE-Privilegien für Routinen

Ist die GRANT-Anweisung in einer CREATE SCHEMA-Anweisung enthalten, darf die GRANT-Anweisung keine Sonder-Privilegien vergeben.

Der aktuelle Berechtigungsschlüssel muss die angegebenen Privilegien vergeben dürfen:

- Er ist der Berechtigungsschlüssel des universellen Benutzers.
- Er ist Eigentümer der Tabelle, der Datenbank, der Storage Group bzw. der Routine.
- Er besitzt die GRANT-Berechtigung zur Weitergabe der Privilegien.

Welcher Berechtigungsschlüssel Eigentümer ist, ist in den Views SCHEMATA, SPACES und STOGROUPS abgelegt. Ob der Berechtigungsschlüssel die GRANT-Berechtigung für ein Privileg besitzt, erfahren Sie aus den Views TABLE_PRIVILEGES, COLUMN_PRIVILEGES, USAGE_PRIVILEGES, CATALOG_PRIVILEGES und ROUTINE_PRIVILEGES (siehe Kapitel „Informationsschemata“).

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

Die GRANT-Anweisung hat mehrere Formate. Beispiele befinden sich beim jeweiligen Format.

Siehe auch

REVOKE, CREATE SCHEMA

GRANT-Format für Tabellen- und Spalten-Privilegien

```
GRANT  { ALL PRIVILEGES | tabellen_und_spalten_privileg ,...}
       ON [TABLE] tabelle
       TO { PUBLIC | berechtigungsschlüssel },...
       [WITH GRANT OPTION]
```

tabellen_und_spalten_privileg ::=

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

ALL PRIVILEGES

Alle Tabellen- und Spalten-Privilegien werden vergeben, die der aktuelle Berechtigungsschlüssel vergeben darf. ALL PRIVILEGES umfasst die Privilegien SELECT, DELETE, INSERT, UPDATE und REFERENCES.

tabellen_und_spalten_privileg

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

ON [TABLE] *tabelle*

Name der Tabelle, für die Sie Privilegien vergeben wollen.

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

Die Tabelle kann eine Basistabelle oder ein View sein. Für einen nicht änderbaren View können Sie nur das Privileg SELECT vergeben.

TO PUBLIC

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

TO *berechtigungsschlüssel*

Die Privilegien gelten für den Berechtigungsschlüssel *berechtigungsschlüssel*. Sie können mehrere Berechtigungsschlüssel angeben.

WITH GRANT OPTION

Die angegebenen Berechtigungsschlüssel erhalten zusätzlich zu den Privilegien die GRANT-Berechtigung. D. h. die Berechtigungsschlüssel sind berechtigt, die erhaltenen Privilegien an andere Berechtigungsschlüssel weiterzugeben. Die Klausel WITH GRANT OPTION dürfen Sie nicht in Verbindung mit PUBLIC verwenden.

WITH GRANT OPTION nicht angegeben:

Die angegebenen Berechtigungsschlüssel können die verliehenen Privilegien nicht weitergeben.

tabellen_und_spalten_privileg

Angabe der einzelnen Tabellen- und Spalten-Privilegien.

SELECT

Privileg, das das Lesen von Sätzen der Tabelle erlaubt.

DELETE

Privileg, das das Löschen von Sätzen der Tabelle erlaubt.

INSERT

Privileg, das das Einfügen von Sätzen in die Tabelle erlaubt.

UPDATE [(*spalte*,...)]

Privileg, das das Ändern von Sätzen der Tabelle erlaubt.

Das Ändern kann auf die angegebenen Spalten eingeschränkt werden.

spalte muss ein Spaltenname der angegebenen Tabelle sein. Sie können mehrere Spalten angeben.

(*spalte*,...) nicht angegeben:

Es ist das Ändern von allen Spalten der Tabelle erlaubt. Auch später hinzugefügte Spalten dürfen geändert werden.

REFERENCES [(*spalte*,...)]

Privileg, das die Definition von Referenzbedingungen erlaubt, die sich auf die Tabelle beziehen.

Die Referenzierung kann auf die angegebenen Spalten eingeschränkt werden.

spalte muss ein Spaltenname der angegebenen Tabelle sein. Sie können mehrere Spalten angeben.

(*spalte*,...) nicht angegeben:

Es ist das Referenzieren von allen Spalten der Tabelle erlaubt. Auch später hinzugefügte Spalten dürfen referenziert werden.

Beispiel

Das Beispiel vergibt alle Tabellen-Privilegien für BILDER an den Berechtigungsschlüssel UTIANW1 und die Tabellen-Privilegien SELECT, DELETE, INSERT, UPDATE für BESCHREIBUNG an den Berechtigungsschlüssel UTIANW2. Die Berechtigungsschlüssel müssen zuvor erzeugt werden.

```
GRANT ALL PRIVILEGES ON bilder TO utianw1
```

```
GRANT SELECT, DELETE, INSERT, UPDATE ON beschreibung TO utianw2
```

GRANT-Format für Sonder-Privilegien

```
GRANT { ALL SPECIAL PRIVILEGES | sonder_privileg ,...}
      ON { CATALOG catalog | STOGROUP stogroup }
      TO { PUBLIC | berechtigungsschlüssel },...
      [WITH GRANT OPTION]

sonder_privileg ::=
  {
    CREATE USER |
    CREATE SCHEMA |
    CREATE STOGROUP |
    UTILITY  |
    USAGE
  }
```

ALL SPECIAL PRIVILEGES

Alle Sonder-Privilegien werden vergeben, die der aktuelle Berechtigungsschlüssel vergeben darf. ALL SPECIAL PRIVILEGES umfasst die Sonder-Privilegien.

sonder_privileg

Die Sonder-Privilegien werden einzeln vergeben. Sie können mehrere Sonder-Privilegien angeben.

ON CATALOG *catalog*

Name der Datenbank, für die Sie Sonder-Privilegien vergeben wollen.

ON STOGROUP *stogroup*

Name der Storage Group, für die Sie das Privileg USAGE vergeben wollen. Der einfache Name der Storage Group kann durch einen Datenbanknamen qualifiziert werden.

TO

Siehe "GRANT - Privilegien vergeben".

WITH GRANT OPTION

Siehe "GRANT - Privilegien vergeben".

sonder_privileg

Angabe der einzelnen Sonder-Privilegien.

CREATE USER

Sonder-Privileg, das das Definieren und Löschen von Berechtigungsschlüsseln erlaubt. Sie dürfen das Privileg CREATE USER nur für eine Datenbank vergeben.

CREATE SCHEMA

Sonder-Privileg, das das Definieren von Datenbank-Schemata erlaubt. Sie dürfen das Privileg CREATE SCHEMA nur für eine Datenbank vergeben.

CREATE STOGROUP

Sonder-Privileg, das das Definieren von Storage Groups erlaubt. Sie dürfen das Privileg CREATE STOGROUP nur für eine Datenbank vergeben.

UTILITY

Sonder-Privileg, das die Verwendung von Utility-Anweisungen erlaubt. Sie dürfen das Privileg UTILITY nur für eine Datenbank vergeben.

USAGE

Sonder-Privileg, das die Verwendung der Storage Group erlaubt. Sie dürfen das Privileg USAGE nur für eine Storage Group vergeben.

Beispiele

Das Beispiel vergibt das Sonderprivileg CREATE SCHEMA an einen bereits bestehenden Berechtigungsschlüssel UTIANW.

```
GRANT CREATE SCHEMA ON CATALOG auftragkunden TO utianw
```

Das Beispiel vergibt alle Sonder-Privilegien für die Datenbank AUFTRAGKUNDEN an den Berechtigungsschlüssel UTIVERW.
Zusätzlich wird das Sonder-Privileg an UTIVERW vergeben, das die Verwendung der Storage Group STOGROUP1 erlaubt.

```
GRANT ALL SPECIAL PRIVILEGES ON CATALOG auftragkunden TO utiverw
GRANT USAGE ON STOGROUP stogroup1 TO utiverw
```

GRANT-Format für EXECUTE-Privilegien (Prozedur)

```
GRANT EXECUTE ON SPECIFIC PROCEDURE prozedur
      TO { PUBLIC | berechtigungsschlüssel },...
      [WITH GRANT OPTION]
```

prozedur ::= *routine*

EXECUTE ON SPECIFIC PROCEDURE *prozedur*

Name der Prozedur, für die das Privileg weitergegeben werden soll. Der einfache Prozedurname kann durch einen Datenbank- und Schemanamen qualifiziert werden.
Wenn Sie die GRANT-Anweisung innerhalb einer CREATE SCHEMA-Anweisung verwenden, dann dürfen Sie den Prozedurnamen nur mit den Datenbank- und Schemanamen aus der CREATE SCHEMA-Anweisung qualifizieren.

TO

Siehe "GRANT - Privilegien vergeben".

WITH GRANT OPTION

Siehe "GRANT - Privilegien vergeben".

Beispiel

Das Privileg, die Prozedur `myproc` ausführen zu dürfen, wird an die Allgemeinheit vergeben.

```
GRANT EXECUTE ON SPECIFIC PROCEDURE myproc TO PUBLIC
```

GRANT-Format für EXECUTE-Privilegien (UDF)

```
GRANT EXECUTE ON SPECIFIC FUNCTION udf
      TO { PUBLIC | berechtigungsschlüssel },...
      [WITH GRANT OPTION]
```

udf ::= *routine*

EXECUTE ON SPECIFIC FUNCTION *udf*

Name der UDF, für die das Privileg weitergegeben werden soll. Der einfache UDF-Name kann durch einen Datenbank- und Schemanamen qualifiziert werden.
Wenn Sie die GRANT-Anweisung innerhalb einer CREATE SCHEMA-Anweisung verwenden, dann dürfen Sie den UDF-Namen nur mit den Datenbank- und Schemanamen aus der CREATE SCHEMA-Anweisung qualifizieren.

TO

Siehe "GRANT - Privilegien vergeben".

WITH GRANT OPTION

Siehe "GRANT - Privilegien vergeben".

Beispiel

Das Privileg, die UDF `myudf` ausführen zu dürfen, wird an die Allgemeinheit vergeben.

```
GRANT EXECUTE ON SPECIFIC FUNCTION myudf TO PUBLIC
```

8.2.3.41 IF - SQL-Anweisungen bedingt ausführen

Die IF-Anweisung führt SQL-Anweisungen in Abhängigkeit von bestimmten Bedingungen aus. Sie darf nur in einer Routine angegeben werden, d.h. im Rahmen einer CREATE PROCEDURE- oder CREATE FUNCTION-Anweisung. Routinen und ihre Verwendung in SESAM/SQL sind detailliert im Kapitel „Routinen“ beschrieben.

Die IF-Anweisung ist eine nicht-atomare SQL-Anweisung, d.h. in ihr können weitere (atomare oder nicht-atomare) SQL-Anweisungen auftreten.

```
IF suchbedingung
   THEN routine_sql_anweisung; [ routine_sql_anweisung; ]...
   [ELSEIF suchbedingung THEN routine_sql_anweisung; [ routine_sql_anweisung; ]...]...
   [ELSE routine_sql_anweisung; [ routine_sql_anweisung; ]... ]
END IF
```

---

suchbedingung

Suchbedingung, deren Auswertung einen Wahrheitswert ergibt.
Die Suchbedingung darf Parameter von Routinen und (wenn die Anweisung Teil einer COMPOUND-Anweisung ist) lokale Variablen, jedoch keine Benutzervariablen enthalten.
Die Angabe einer Spalte ist nur erlaubt, wenn sie Teil einer Unterabfrage ist.

routine_sql_anweisung

SQL-Anweisung, die im THEN- oder ELSE-Teil der IF-Anweisung ausgeführt werden soll.
Eine SQL-Anweisung wird mit „;“ (Semikolon) abgeschlossen.
Mehrere SQL-Anweisungen können nacheinander angegeben werden. Sie werden in der angegebenen Reihenfolge ausgeführt.
Vor der Durchführung einer SQL-Anweisung werden keine Privilegien geprüft. Eine SQL-Anweisung in einer Routine darf auf die Parameter der Routine und (wenn die Anweisung Teil einer COMPOUND-Anweisung ist) auf lokale Variablen, jedoch nicht auf Benutzervariablen zugreifen.

Syntax und Bedeutung von *routine_sql_anweisung* sind zentral im Abschnitt „SQL-Anweisungen in Routinen“ beschrieben. Die dort genannten SQL-Anweisungen dürfen verwendet werden.

Ausführungshinweise

Die IF- und die ELSEIF-Klauseln werden von links nach rechts abgearbeitet. Für die erste THEN-Klausel, deren Suchbedingung Wahrheitswert wahr ergibt, werden die zugehörigen SQL-Anweisungen bearbeitet. Die IF-Anweisung ist danach beendet.

Ergibt keine der Suchbedingungen den Wahrheitswert wahr und existiert eine ELSE-Klausel, dann werden die SQL-Anweisungen der ELSE-Klausel bearbeitet.

SQL-Anweisungen werden nicht bearbeitet, wenn die zugehörige Suchbedingung den Wahrheitswert unbestimmt ergibt.

Die IF-Anweisung ist eine nicht-atomare Anweisung:

- Wenn die IF-Anweisung Teil einer COMPOUND-Anweisung ist, dann gelten die dort beschriebenen Regeln, insbesondere die dort definierten Fehler-Routinen.
- Wenn die IF-Anweisung **nicht** Teil einer COMPOUND-Anweisung ist und eine der SQL-Anweisungen einen SQLSTATE meldet, dann werden ggf. nur die Änderungen dieser SQL-Anweisung rückgängig gemacht. Die IF-Anweisung und die Routine, in der sie enthalten ist, werden abgebrochen. Die SQL-Anweisung, in der die Routine verwendet wurde, liefert den betreffenden SQLSTATE zurück.

Beispiel

Die SQL-Anweisungen werden nur durchgeführt, wenn die Tabelle `tab` nicht leer ist.

```
IF (SELECT COUNT(*) FROM tab) > 0 THEN routine_sql_anweisung END IF
```

Siehe auch

CREATE PROCEDURE, CREATE FUNCTION

8.2.3.42 INCLUDE - ESQL-Programmteile einfügen

INCLUDE fügt Programmtext, der in einem PLAM-Bibliothekselement steht, in ein ESQL-Programm ein. Der Programmtext kann eingebettete SQL-Anweisungen und Utility-Anweisungen enthalten sowie Anweisungen der Wirtssprache. Beispielsweise können Sie mit INCLUDE den Kommunikationsbereich zwischen SQL und der Wirtssprache in ein ESQL-Programm einfügen, sofern ein entsprechend vorbereitetes Element in einer BS2000-PLAM-Bibliothek vorhanden ist.

Während der Vorübersetzung durch den ESQL-Precompiler wird die INCLUDE-Anweisung durch den Text des angegebenen Bibliothekselements ersetzt. Die INCLUDE-Anweisungen werden in der Reihenfolge abgearbeitet, in der sie im Programm stehen.

`INCLUDE` *bibliothekselement*

bibliothekselement `::= {` *alphanumerisches_literal* | *regulärer_name* `}`

---

bibliothekselement

Name eines PLAM-Bibliothekselements vom Typ S. Der Name muss ein gültiger Name für ein PLAM-Bibliothekselement sein, ohne Versionsangabe (Suffix). Existieren mehrere Versionen des angegebenen Bibliothekselements in derselben PLAM-Bibliothek, verwendet SESAM/SQL die aktuelle Version.

Zuweisen von PLAM-Bibliotheken mit ESQL-Precompiler-Optionen

Jede PLAM-Bibliothek, die Bibliothekselemente enthält, muss durch eine ESQL-Precompiler-Option bekannt gemacht werden (siehe Handbuch „ ESQL-COBOL für SESAM/SQL-Server“). Mit diesen Optionen legen Sie auch fest, in welcher Reihenfolge PLAM-Bibliotheken nach Bibliothekselementen durchsucht werden. Existieren in verschiedenen PLAM-Bibliotheken Bibliothekselemente gleichen Namens, so verwendet der ESQL-Precompiler immer die PLAM-Bibliothek, die als Erste dieses Bibliothekselement enthält.

Beispiel

Das Beispiel fügt das Bibliothekselement VARIABLES in ein ESQL-Programm ein. In VARIABLES können z.B. häufig benötigte Benutzervariablen definiert werden.

 INCLUDE variables

8.2.3.43 INSERT - Sätze in Tabelle einfügen

INSERT fügt Sätze in eine bestehende Tabelle ein.

Um Sätze in die angegebene Tabelle einzufügen, müssen Sie Eigentümer dieser Tabelle sein oder das INSERT-Privileg für diese Tabelle besitzen. Zusätzlich muss der Transaktionsmodus der aktuellen Transaktion READ WRITE sein.

Die in der INSERT-Anweisung (und in Voreinstellungen) vorkommenden Spezial-Literale (siehe "Spezial-Literale") sowie die Zeitfunktionen CURRENT_DATE, CURRENT_TIME und CURRENT_TIMESTAMP werden einmal ausgewertet und die berechneten Werte gelten für alle Einfügungen.

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

```
INSERT INTO tabelle

        { [ spaltenliste ] [COUNT INTO spalte ] werte-deklaration | DEFAULT VALUES }

        [RETURN INTO parameter-deklaration ]


werte-deklaration ::= { abfrageausdruck | VALUES { zeile_2 , zeile_2 ,... | zeile_1 } }


parameter-deklaration ::=
{
    : benutzervariable [[INDICATOR] : indikatorvariable ] |
    routinenparameter |
    lokale_variable
}


spaltenliste ::= (
{
   spalte |
   spalte[posnr] |
   spalte[min .. max] |
   spalte ( posnr ) |
   spalte ( min .. max )
},...
)


zeile_2 ::= { ( insert_ausdruck_2 ,...) | insert_ausdruck_2 }
```

zeile_1 ::= { (*insert_ausdruck_1* ,...) | *insert_ausdruck_1* }

insert_ausdruck_2 ::= { *ausdruck* | NULL }

insert_ausdruck_1 ::= { *ausdruck* | NULL | DEFAULT | * | <{ *wert* | NULL},...> }

tabelle

Name der Tabelle, in die Sätze eingefügt werden sollen. Die Tabelle kann eine Basistabelle oder ein änderbarer View sein.

spaltenliste

Zählt die Spalten und die Ausprägungsbereiche von multiplen Spalten auf, deren Werte in den einzufügenden Sätzen in der INSERT-Anweisung angegeben werden, und legt die Reihenfolge dafür fest. Die Werte der restlichen Spalten in den einzufügenden Sätzen werden nicht in der INSERT-Anweisung angegeben, sondern sind DEFAULT-, NULL- oder von SESAM/SQL bestimmte Werte.

Keine *spaltenliste* angegeben:

Die INSERT-Anweisung gibt für alle Spalten der Tabelle *tabelle* die Werte in den einzufügenden Sätze an, außer für die mit COUNT INTO angegebenen Spalte, und zwar in der Spaltenreihenfolge, die mit CREATE TABLE und ALTER TABLE bzw. mit CREATE VIEW festgelegt wurde.

spalte

Einfache Spalte, deren Werte in den einzufügenden Sätze in der INSERT-Anweisung angegeben sind.

Für *spalte* geben Sie eine Spalte der angegebenen Tabelle an. Die Reihenfolge der Spalten muss nicht mit der Reihenfolge in der Tabelle übereinstimmen. Innerhalb der Spaltenliste darf eine einfache Spalte nicht doppelt vorkommen.

spalte(posnr) / spalte[posnr]

Element einer multiplen Spalte, dessen Werte in den einzufügenden Sätzen in der INSERT-Anweisung angegeben sind. Die multiple Spalte muss in der Tabelle enthalten sein.

Werden mehrere Elemente einer multiplen Spalte angegeben, so muss der angegebene Indexbereich lückenlos sein. Kein Element der multiplen Spalte darf mehfach vorkommen.

Für *posnr* geben Sie eine vorzeichenlose Ganzzahl >= 1 an.

spalte(min..max) / spalte[min..max]

Elemente einer multiplen Spalte, deren Werte in den einzufügenden Sätzen in der INSERT-Anweisung angegeben sind. Die multiple Spalte muss in der Tabelle enthalten sein.

Werden mehrere Elemente einer multiplen Spalte angegeben, so muss der angegebene Indexbereich lückenlos sein. Kein Element der multiplen Spalte darf mehrfach vorkommen.

Für *min* und *max* geben Sie vorzeichenlose Ganzzahlen >= 1 an; *max* muss >= *min* sein.

> ¡ In der Syntax kursiv gedruckte eckige Klammern sind Sonderzeichen und müssen in der Anweisung angegeben werden.

COUNT INTO *spalte*

Einfache Spalte, deren Werte in den einzufügenden Sätzen von SESAM/SQL bestimmt werden und nicht in der INSERT-Anweisung angegeben werden dürfen (Zählspalte). *spalte* darf nicht in *spaltenliste* vorkommen.

Die Spalte muss einen Ganz- oder Festpunktzahltyp (SMALLINT, INT, DECIMAL, NUMERIC) haben und zu einem Primärschlüssel gehören. Die Spalte darf weder in einer Referenzbedingung noch in einer Check-Bedingung der Tabelle *tabelle* enthalten sein.

SESAM/SQL bestimmt die Werte der zugehörigen Spalte in allen einzufügenden Sätzen, und zwar derart, dass die Primärschlüsselwerte innerhalb der Tabelle eindeutig sind.

abfrageausdruck

abfrageausdruck ist ein Abfrage-Ausdruck, dessen Ergebnistabelle die benötigten Spaltenwerte der einzufügenden Sätze angibt. Für jeden Satz der Ergebnistabelle wird ein Satz in die Tabelle *tabelle* eingefügt. Liefert *abfrageausdruck* eine leere Tabelle, so wird kein Satz eingefügt und ein entsprechender SQLSTATE gesetzt, der mit WHENEVER NOT FOUND behandelt werden kann.

VALUES-Klausel

Die benötigten Spaltenwerte sind für jeden einzufügenden Satz einzeln mit *zeile_2* bzw. *zeile_1* angegeben. Die Tabelle aus allen diesen Sätzen bzw. aus diesem einen Satz spielt die Rolle der Ergebnistabelle bei *abfrageausdruck*.

zeile_2

Es werden soviele Sätze eingefügt, wie *zeile_2* angegeben sind.

Jede *zeile_2* muss gleich viele Spalten haben. Die Datentypen der Spalten der Ergebnistabelle ergeben sich aus den Regeln, die unter „Datentyp der Ergebnisspalten bei UNION“ beschrieben sind. Enthält eine Spalte der Ergebnistabelle nur NULL, so ist ihr Datentyp der der korrespondierenden Spalte von *tabelle*.

insert_ausdruck_2

ausdruck

Der *ausdruck* von *insert_ausdruck_2* muss einfach sein.

NULL

Die zugehörige Spalte in einzufügenden Sätzen muss einfach sein. Sie wird auf den NULL-Wert gesetzt.

zeile_1

Es wird ein einziger Satz eingefügt. Die Ergebnistabelle mit den benötigten Werten für diesen einzufügenden Satz besteht aus *zeile_1*.

insert_ausdruck_1

ausdruck

Der *ausdruck* von *insert_ausdruck_1* muss entweder einfach sein, oder eine Benutzervariable, die ein Vektor mit mehreren Elementen ist. Ist eine solche Benutzervariable oder ein Aggregat angegeben, so muss die Anzahl der Elemente des Vektors bzw des Aggregats mit der Anzahl der Elemente der zugehörigen Spalte im einzufügenden Satz übereinstimmen.

NULL

Die zugehörige Spalte im einzufügenden Satz muss einfach sein. Sie wird auf den NULL-Wert gesetzt.

DEFAULT

Die zugehörige Spalte im einzufügenden Satz muss einfach sein. Sie wird auf den voreingestellten Wert gesetzt. Die Voreinstellung wird bei der Definition der Spalte festgelegt. Ist keine Voreinstellung definiert, so wird die Spalte auf den NULL-Wert gesetzt.

*

Die zugehörige Spalte im einzufügenden Satz muss einfach sein, einen Ganz- oder Festpunktzahltyp (SMALLINT, INT, DECIMAL, NUMERIC) haben und zu einem Primärschlüssel gehören. Die Spalte darf weder in einer Referenzbedingung noch in einer Check-Bedingung der Tabelle *tabelle* enthalten sein.

* darf nur einmal in der VALUES-Klausel vorkommen, und nicht zusammen mit COUNT INTO.

Der Wert der zugehörigen Spalte im einzufügenden Satz wird von SESAM/SQL bestimmt, und zwar so, dass die Primärschlüsselwerte innerhalb der Tabelle eindeutig sind.

<{*wert*, NULL},...>

Aggregat, das einer multiplen Spalte zugewiesen werden soll. Die Anzahl der Werte muss mit der Anzahl der Ausprägungen übereinstimmen.

In *abfrageausdruck*, *insert_ausdruck_2* und *insert_ausdruck_1* dürfen Sie keine Tabelle angeben, die sich auf die zu Grunde liegende Basistabelle, in die die neuen Sätze eingefügt werden, bezieht. Insbesondere dürfen Sie nicht *tabelle* angeben.

Die Anzahl der Spalten von *abfrageausdruck*, *zeile_2* oder *zeile_1* muss gleich der Anzahl der Spalten sein, die gemäß den Angaben bei *spaltenliste* und COUNT INTO anzugeben sind. Die i-te Spalte der Ergebnistabelle enthält die Werte für die i-te Spaltenangabe in *spaltenliste* (falls *spaltenliste* angegeben ist), bzw. für die i-te Spalte von *tabelle*, wobei eine mit COUNT INTO angegebene Spalte von *tabelle* nicht mitgezählt wird.

Für diese Zuweisungen gelten die im Abschnitt „Werte in Tabellenspalten eintragen“ angegebenen Zuweisungregeln.

Die restlichen Spalten der eingefügten Sätze werden folgendermaßen gesetzt:

- Die mit COUNT INTO angegebene Spalte wird auf einen von SESAM/SQL bestimmten Wert gesetzt.
- Eine Spalte mit Voreinstellung wird auf den voreingestellten Wert (DEFAULT) gesetzt.
- Eine Spalte ohne Voreinstellung wird auf den NULL-Wert gesetzt.

Ist *tabelle* ein View, so werden die Sätze in die zugrunde liegende Basistabelle eingefügt; die Spalten der Basistabelle, die nicht im View enthalten sind, werden ebenso gesetzt.

DEFAULT VALUES

Ein Satz, der nur aus den spaltenspezifischen Voreinstellungen besteht, in die Tabelle eintragen.

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

RETURN INTO

Der von SESAM/SQL bestimmte Wert für die mit COUNT INTO angegebene Spalte, bzw. für den * als *insert_ausdruck_1* wird in einem Ausgabeziel abgespeichert. Werden mehrere Sätze eingefügt, so wird der zuletzt von SESAM/SQL bestimmte Wert abgespeichert.

Die RETURN INTO-Klausel ist nur erlaubt, wenn entweder COUNT INTO angegeben ist, oder ein * als *insert_ausdruck_1* verwendet wird.

:benutzervariable, *routinenparameter*, *lokale_variable*

Name einer Benutzervariablen (wenn die Anweisung **nicht** Bestandteil einer Routine ist) bzw. Name eines Prozedurparameters vom Typ INOUT oder OUT oder einer lokalen Variablen (wenn die Anweisung Bestandteil einer Routine ist). Der Wert der Zählspalte wird dem angegebenen Ausgabeziel zugewiesen.

Das Ausgabeziel muss ein numerischer Datentyp sein.

indikatorvariable

Name der Indikatorvariablen zur vorangehenden Benutzervariablen.

Werte für multiple Spalte einfügen

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

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

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

> **!** **ACHTUNG!** Die Position eines Elements innerhalb der multiplen Spalte kann sich von der Position des entsprechenden Elements in der INSERT-Anweisung unterscheiden. Wenn ein Element einer multiplen Spalte auf den NULL-Wert gesetzt wird, so werden alle Elemente mit einer höheren Position durch verringern ihrer Position um eine Stelle „nach links“ verschoben und der NULL-Wert erhält die höchste Position.

INSERT und Integritätsbedingungen

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

INSERT und Transaktionssicherung

INSERT leitet außerhalb von Routinen eine SQL-Transaktion ein, wenn keine Transaktion offen ist. Durch die Definition eines Isolationslevels bei konkurrierenden Transaktionen können Sie steuern, welche Auswirkungen die INSERT-Anweisung auf diese Transaktionen hat (siehe Abschnitt „SET TRANSACTION - Transaktionseigenschaften festlegen“).

Tritt während der INSERT-Anweisung ein Fehler auf, so werden alle bereits eingefügten Sätze wieder gelöscht.

Beispiele

Die beiden folgenden Anweisungen nehmen jeweils drei Sätze in die Tabelle AUFTRAG auf.
In der zweiten INSERT-Anweisung wird der Wert für den Primärschlüssel von SESAM/SQL bestimmt. Der zuletzt vergebene Wert wird in der Benutzervariablen ANRRET gespeichert.

```
INSERT INTO auftrag (anr, knr, konr, atext, fertigsoll, astnr)
   VALUES (345, 101, 20, 'Netzwerk:Installation', DATE'<date>',1),
          (346, 101, 20, 'Netzwerk:Test',          DATE'<date>',1),
          (347, 101, 20, 'Netzwerk:Schulung',      DATE'<date>',1)

INSERT INTO auftrag (knr, konr, atext, fertigsoll, astnr)
   COUNT INTO anr
   VALUES  (:KNR, :KONR, :ATEXT1, :FSOLL1, :ASTNR),
           (:KNR, :KONR, :ATEXT2, :FSOLL2, :ASTNR),
           (:KNR, :KONR, :ATEXT3, :FSOLL3, :ASTNR)
   RETURN INTO :ANRRET
```

In einer Tabelle FRAUEN sind die Spalten VORNAME und NACHNAME wie in der Tabelle KONTAKT definiert. Durch die folgende INSERT-Anweisung werden alle weiblichen Kontaktpersonen in die Tabelle FRAUEN aufgenommen:

```
INSERT INTO frauen (vorname, nachname)
   SELECT vorname, nachname
```

```
FROM kontakt
WHERE anrede IN ('Frau','Fraeulein','Mrs.','Ms.')
```

Siehe auch

DELETE, MERGE, UPDATE

8.2.3.44 ITERATE - zum nächsten Schleifendurchlauf wechseln

Mit der ITERATE-Anweisung wird zum nächsten Schleifendurchlauf gewechselt.
Sie darf nur in den Kontrollanweisungen FOR, LOOP, REPEAT und WHILE einer Routine angegeben werden, d.h. im Rahmen einer CREATE PROCEDURE- oder CREATE FUNCTION-Anweisung. Routinen und ihre Verwendung in SESAM/SQL sind detailliert im Kapitel „Routinen“ beschrieben.

`ITERATE` *marke*

---

marke

Marke der FOR-, LOOP-, REPEAT- oder WHILE-Anweisung, die die ITERATE-Anweisung enthält. Der aktuelle Schleifendurchlauf wird beendet. Es wird zum nächsten Schleifendurchlauf gewechselt.

Durch Angabe einer *marke* können auch äußere Schleifendurchläufe beendet werden. Innere Schleifendurchläufe werden dann sofort beendet.

Beispiel

Wenn der Wert der Variablen i durch 3 teilbar ist, dann wird mit ITERATE sofort zum nächsten Schleifendurchlauf gewechselt.

```
DECLARE i INTEGER DEFAULT 1;
   ...
   label:
   REPEAT
       SET i = i + 1;
       IF MOD(i,3)=0 THEN ITERATE label;
       END IF;
       ...
       UNTIL i >100
   END REPEAT label;
```

Siehe auch

CREATE PROCEDURE, CREATE FUNCTION, FOR, LOOP, REPEAT, WHILE

8.2.3.45 LEAVE - Schleife oder COMPOUND-Anweisung beenden

Die LEAVE-Anweisung beendet eine Schleife oder eine COMPOUND-Anweisung.
Sie darf nur in den Kontrollanweisungen COMPOUND, FOR, LOOP, REPEAT und WHILE einer Routine angegeben werden, d.h. im Rahmen einer CREATE PROCEDURE- oder CREATE FUNCTION-Anweisung.
Routinen und ihre Verwendung in SESAM/SQL sind detailliert im Kapitel „Routinen“ beschrieben.

LEAVE *marke*

---

marke

Marke der COMPOUND-, FOR-, LOOP-, REPEAT- oder WHILE-Anweisung, die die LEAVE-Anweisung enthält. Die bezeichnete Anweisung wird beendet.

Beispiel

Siehe das Beispiel der LOOP-Anweisung auf "LOOP - SQL-Anweisungen in einer Schleife ausführen".

Siehe auch

CREATE PROCEDURE, CREATE FUNCTION, FOR, LOOP, REPEAT, WHILE

8.2.3.46 LOOP - SQL-Anweisungen in einer Schleife ausführen

Die LOOP-Anweisung führt SQL-Anweisungen in einer Schleife aus.

Mit der ITERATE-Anweisung kann sofort zum nächsten Schleifendurchlauf gewechselt werden. Die Schleife kann über eine LEAVE-Anweisung abgebrochen werden.

Die LOOP-Anweisung darf nur in einer Routine angegeben werden, d.h. im Rahmen einer CREATE PROCEDURE- oder CREATE FUNCTION-Anweisung. Routinen und ihre Verwendung in SESAM/SQL sind detailliert im Kapitel „Routinen“ beschrieben.

Die LOOP-Anweisung ist eine nicht-atomare SQL-Anweisung, d.h. in ihr können weitere (atomare oder nicht-atomare) SQL-Anweisungen auftreten.

Wenn die LOOP-Anweisung Teil einer COMPOUND-Anweisung ist, dann kann die Schleife bei entsprechenden Fehler-Routinen auch beim Auftreten eines bestimmten SQLSTATE (z.B. keine Daten, Klasse 02xxx) verlassen werden.

[*marke* :]

LOOP

routine_sql_anweisung; [*routine_sql_anweisung;*]...

END LOOP [*marke*]

---

marke

Die Marke vor der LOOP-Anweisung (Anfangsmarke) bezeichnet den Anfang der Schleife. Sie darf nicht identisch sein mit einer anderen Marke innerhalb der Schleife.

Die Anfangsmarke muss nur dann angegeben werden, wenn mit ITERATE zum nächsten Schleifendurchlauf gewechselt werden soll oder wenn die Schleife über eine LEAVE-Anweisung verlassen werden soll. Sie sollte aber stets verwendet werden, damit SESAM/SQL die korrekte Struktur der Prozedur überprüfen kann (z.B. bei geschachtelten Schleifen).

Die Marke am Ende der LOOP-Anweisung (Endemarke) bezeichnet das Ende der Schleife. Wenn die Endemarke angegeben ist, dann muss auch die Anfangsmarke angegeben sein. Beide Marken müssen identisch sein.

routine_sql_anweisung

SQL-Anweisung, die in der LOOP-Anweisung ausgeführt werden soll.

Eine SQL-Anweisung wird mit „;“ (Semikolon) abgeschlossen.
Mehrere SQL-Anweisungen können nacheinander angegeben werden. Sie werden in der angegebenen Reihenfolge ausgeführt.
Vor der Durchführung einer SQL-Anweisung werden keine Privilegien geprüft. Eine SQL-Anweisung in einer Routine darf auf die Parameter der Routine und (wenn die Anweisung Teil einer COMPOUND-Anweisung ist) auf lokale Variablen, jedoch nicht auf Benutzervariablen zugreifen.

Syntax und Bedeutung von *routine_sql_anweisung* sind zentral im Abschnitt „SQL-Anweisungen in Routinen“ beschrieben. Die dort genannten SQL-Anweisungen dürfen verwendet werden.

Ausführungshinweise

Die LOOP-Anweisung ist eine nicht-atomare Anweisung:

- Wenn die LOOP-Anweisung Teil einer COMPOUND-Anweisung ist, dann gelten die dort beschriebenen Regeln, insbesondere die dort definierten Fehler-Routinen.
- Wenn die LOOP-Anweisung **nicht** Teil einer COMPOUND-Anweisung ist und eine der SQL-Anweisungen einen SQLSTATE meldet, dann werden ggf. nur die Änderungen dieser Anweisung rückgängig gemacht. Die LOOP-Anweisung und die Routine, in der sie enthalten ist, werden abgebrochen. Die SQL-Anweisung, in der die Routine verwendetwurde, liefert den betreffenden SQLSTATE zurück.

Beispiel

Eine Schleife wird nach 1000 Durchläufen mit LEAVE abgebrochen.

```
DECLARE i INTEGER DEFAULT 0;
   ...
   label:
   LOOP
       SET i = i+1;
       IF i > 1000 THEN LEAVE label;
       ...
   END LOOP label;
```

Siehe auch

CREATE PROCEDURE, CREATE FUNCTION, ITERATE, LEAVE

8.2.3.47 MERGE - Sätze in Tabelle einfügen oder Spaltenwerte ändern

MERGE vereint die Funktionen INSERT und UPDATE in einem Arbeitsgang. Abhängig vom Ergebnis der Bedingung in der ON-Klausel ändert MERGE Spaltenwerte von bereits existierenden Sätzen (WHEN MATCHED THEN) oder fügt neue Sätze in eine bestehende Tabelle ein (WHEN NOT MATCHED THEN).

Die Bedingung kann dabei von einer einfachen Existenzabfrage bis hin zu komplexen Suchkriterien reichen. Auch triviale Bedingungen (z.B `1 <> 1`) sind möglich; sie führen dazu, dass nur Sätze geändert bzw. neu eingefügt werden.

Die in der MERGE-Anweisung (und in Voreinstellungen) vorkommenden Spezial-Literale (siehe "Spezial-Literale") sowie die Zeitfunktionen CURRENT_DATE, CURRENT_TIME und CURRENT_TIMESTAMP werden einmal ausgewertet und die berechneten Werte gelten für alle Einfügungen.

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

Zur erfolgreichen Ausführung der MERGE-Anweisung müssen bestimmte Voraussetzungen erfüllt sein:

- Zum Einfügen oder Ändern von Sätzen in *tabelle* müssen Sie
  - Eigentümer von *tabelle* sein oder
  - wenigstens das INSERT-Privileg besitzen, wenn *satz_einfuegen* angegeben ist oder
  - wenigstens das UPDATE-Privileg für alle Spalten besitzen, die in *satz_ändern* geändert werden, wenn *satz_ändern* angegeben ist.
- Außerdem müssen Sie das SELECT-Privileg für alle Tabellen besitzen, die in *tabellenangabe* angesprochen werden.
- Der Transaktionsmodus der aktuellen Transaktion muss READ WRITE sein.

```
MERGE INTO tabelle [[AS] korrelationsname ]
       USING tabellenangabe
       ON suchbedingung
       { WHEN MATCHED THEN satz_ändern |
         WHEN NOT MATCHED THEN satz_einfügen }...

satz_ändern ::= UPDATE SET spalte = spalten-wert [, spalte = spalten-wert ]...

spalten-wert ::= { ausdruck | DEFAULT | NULL}

satz_einfügen ::= INSERT [( spalte ,...)] [COUNT INTO spalte ] VALUES( wert [ ,wert ] ... )

wert ::= { ausdruck | NULL | DEFAULT | * }
```

---

tabelle

Name der Zieltabelle, in die Sätze eingefügt oder in der Sätze geändert werden sollen. Die Zieltabelle kann eine Basistabelle oder ein änderbarer View sein.
Sie darf keine multiplen Spalten besitzen (siehe Hinweis auf "MERGE - Sätze in Tabelle einfügen oder Spaltenwerte ändern").

korrelationsname

Tabellenname, der innerhalb dieser Anweisung eine Umbenennung für *tabelle* ist.

Bei jeder Spaltenangabe, die sich auf *tabelle* bezieht, müssen Sie den Spaltennamen mit dem neuen Namen *korrelationsname* qualifizieren, wenn der Spaltenname nicht eindeutig ist.

Der neue Name muss eindeutig sein, d.h. *korrelationsname* darf nur einmal in einer Tabellenangabe dieser Anweisung vorkommen.

Sie müssen eine Tabelle umbenennen, wenn die Spalten der Tabelle ohne Umbenennung nicht eindeutig angegeben werden können.

Außerdem können Sie eine Tabelle umbenennen, um durch entsprechende Namen einen Ausdruck verständlicher zu formulieren oder um lange Namen abzukürzen.

USING *tabellenangabe*

Angabe einer Quelltabelle (ungleich der Zieltabelle *tabelle*), unter deren Verwendung Sätze in die Zieltabelle *tabelle* eingefügt oder Sätze in der Zieltabelle *tabelle* geändert werden sollen. Sie darf keine multiplen Spalten besitzen (siehe Hinweis auf "MERGE - Sätze in Tabelle einfügen oder Spaltenwerte ändern"). Innerhalb der *tabellenangabe* darf auch die Zieltabelle *tabelle* referenziert werden.

ON *suchbedingung*

Angabe der Bedingung, die entscheidet, ob die UPDATE-Klausel ausgeführt werden soll (Resultat: TRUE) oder ob die INSERT-Klausel ausgeführt werden soll (Resultat: FALSE).

Genauer gesagt wird für jeden Satz der Quelltabelle geprüft, ob es einen Satz in der Zieltabelle gibt, so dass für die Kombination dieser beiden Sätze die *suchbedingung* wahr ist.
Wenn es keinen derartigen Satz in der Zieltabelle gibt, dann wird die INSERT-Klausel ausgeführt, d.h. der Satz der Quelltabelle wird in die Zieltabelle eingefügt. Wenn es einen oder mehrere derartige Sätze in der Zieltabelle gibt, dann wird für jeden dieser Sätze die UPDATE-Klausel ausgeführt, d.h. die entsprechenden Sätze in der Zieltabelle werden geändert.
Zwei verschiedene Sätze der Quelltabelle dürfen nicht zu Änderungen des gleichen Satzes in der Zieltabelle führen (Mehrfachänderung), sonst wird die MERGE-Anweisung mit SQLSTATE abgebrochen.

WHEN MATCHED THEN *satz_ändern*

Ein zu ändernder Satz wurde gefunden.

UPDATE SET ...

Angaben zum Update des zu ändernden Satzes.
Zur Beschreibung der Parameter *spalte*, *ausdruck*, DEFAULT und NULL siehe die entsprechenden Beschreibungen in der SQL-Anweisung UPDATE auf "UPDATE - Spaltenwerte ändern". Ein zu ändernder Satz darf nur einmal geändert werden. Eine weitere Änderung wird mit SQLSTATE abgewiesen.
Bei partitionierten Tabellen darf der Primärschlüsselwert nicht geändert werden.

WHEN NOT MATCHED THEN *satz_einfügen*

Es wurde kein entsprechender Satz gefunden. Der neue Satz soll eingefügt werden.

INSERT ...

Angaben zum Einfügen des neuen Satzes.

(spalte,...)

Zählt die Spalten auf, deren Werte in der INSERT-Klausel der MERGE-Anweisung angegeben werden, und legt die Reihenfolge dafür fest. Die Werte der restlichen Spalten des einzufügenden Satzes werden nicht in der MERGE-Anweisung angegeben, sondern sind DEFAULT-, NULL- oder von SESAM/SQL bestimmte Werte.

Zur Beschreibung des Parameters *spalte* siehe die entsprechende Beschreibung in der SQL-Anweisung INSERT auf "INSERT - Sätze in Tabelle einfügen".

Keine *spaltenliste* angegeben:
Die MERGE-Anweisung gibt für alle Spalten der Zieltabelle *tabelle* die Werte des einzufügenden Satzes an, außer für die mit COUNT INTO angegebenen Spalte, und zwar in der Spaltenreihenfolge, die mit CREATE TABLE und ALTER TABLE bzw. mit CREATE VIEW festgelegt wurde.

COUNT INTO *spalte*

Siehe die entsprechende Beschreibung in der SQL-Anweisung INSERT auf "INSERT - Sätze in Tabelle einfügen".

VALUES (...)

Die benötigten Spaltenwerte sind für den einzufügenden Satz angegeben.

Zur Beschreibung von *ausdruck*, NULL, DEFAULT und * siehe die Beschreibung zu *iinsert_ausdruck_1* in der SQL-Anweisung INSERT auf "INSERT - Sätze in Tabelle einfügen".

In *ausdruck* dürfen Sie keine Tabelle angeben, die sich auf die Zieltabelle, in die die neuen Sätze eingefügt werden, bezieht.
Insbesondere dürfen Sie sich auf keine Spalte der Zieltabelle beziehen.

Die Anzahl der Spalten der VALUES-Klausel muss gleich der Anzahl der Spalten sein, die gemäß den Angaben bei *(spalte,...)* und COUNT INTO anzugeben sind. Die i-te Spalte der Zieltabelle enthält die Werte für die i-te Spaltenangabe in *(spalte,...)* (falls *(spalte,...)* angegeben ist), bzw. für die i-te Spalte von *tabelle*, wobei eine mit COUNT INTO angegebene Spalte von *tabelle* nicht mitgezählt wird.

Für diese Zuweisungen gelten die im Abschnitt „Werte in Tabellenspalten eintragen“ angegebenen Zuweisungregeln.

Die restlichen Spalten des eingefügten Satzes werden folgendermaßen gesetzt:

- Die mit COUNT INTO angegebene Spalte wird auf einen von SESAM/SQL bestimmten Wert gesetzt.
- Eine Spalte mit Voreinstellung wird auf den voreingestellten Wert (DEFAULT) gesetzt.
- Eine Spalte ohne Voreinstellung wird auf den NULL-Wert gesetzt.

Wenn die Zieltabelle *tabelle* ein View ist, dann werden die Sätze in die zugrunde liegende Basistabelle eingefügt; die Spalten der Basistabelle, die nicht im View enthalten sind, werden ebenso gesetzt.

Werte für multiple Spalte einfügen

Basistabellen mit multiplen Spalten können in der MERGE-Anweisung nicht direkt bearbeitet werden.

> ¡ Wenn man aber einen (änderbaren) View mit z.B. dem Abfragausdruck SELECT *spaltenliste* FROM *tabelle* definiert ohne multiple Spalten in *spaltenliste* anzugeben, dann kann damit die MERGE-Anweisung ausgeführt werden.

MERGE und Integritätsbedingungen

Durch Angabe von Integritätsbedingungen bei der Definition der Basistabelle können Sie den Wertebereich für die entsprechenden Spalten einschränken. Die in der MERGE-Anweisung angegebenen Werte müssen den definierten Integritätsbedingungen genügen, sonst wird die MERGE-Anweisung mit SQLSTATE abgebrochen.

MERGE und Transaktionssicherung

MERGE leitet außerhalb von Routinen eine SQL-Transaktion ein, wenn keine Transaktion offen ist. Durch die Definition eines Isolationslevels bei konkurrierenden Transaktionen können Sie steuern, welche Auswirkungen die MERGE-Anweisung auf diese Transaktionen hat (siehe Abschnitt „SET TRANSACTION - Transaktionseigenschaften festlegen“).

Tritt während der MERGE-Anweisung ein Fehler auf, so werden sämtliche, durch die MERGE-Anweisung bereits durchgeführten Änderungen rückgängig gemacht.

Beispiele

Folgendes Beispiel dient der Bestandsverwaltung beim Eintreffen einer neuen Lieferung.
Bei den bereits vorhandenen Artikeln mit demselben Preis wird der Lagerbestand in der Bestandstabelle aktualisiert. Neue Artikel der Lieferungstabelle werden der Bestandstabelle hinzugefügt.

```
MERGE INTO bestand AS b USING lieferung AS l
      ON b.artikel_nr = l.artikel_nr AND b.artikel_preis = l.artikel_preis
      WHEN MATCHED THEN
             UPDATE SET artikel_anz = b.artikel_anz + l.artikel_anz
      WHEN NOT MATCHED THEN
```

```
                    INSERT (artikel_nr,artikel_preis,artikel_anz)
                    VALUES (l.artikel_nr,l.artikel_preis,l.artikel_anz)
```

Folgendes komplexe Beispiel dient ebenfalls der Bestandsverwaltung beim Eintreffen einer neuen Lieferung. Die Daten der neuen Lieferung werden z.B. in der CSV-Eingabedatei LIE-FERUNG.DATA (mit einer Überschriftszeile) geliefert:

```
Artikelnummer,Menge,Preis
1, 4, 18.50
2, 11, 19.90
3, 0, 22.95
4, 3, 84.30
5, 7, 25.90
```

Folgende MERGE-Anweisung aktualisiert die Tabelle *bestand* für die bereits vorhandenen Artikel. Neue Artikel der Lieferung werden der Bestandstabelle hinzugefügt. Die Überschriftszeile der CSV-Datei wird durch die Klausel WITH ORDINALITY in Verbindung mit WHERE übersprungen

```
MERGE INTO  bestand
      USING (SELECT CAST(artikelnummer as INT),
                    CAST(menge as INT),
                    CAST(neuerpreis as NUMERIC(10,2))
             FROM TABLE(CSV('LIEFERUNG.DATA'
                             DELIMITER ',' QUOTE '"' ESCAPE '\',
                             varchar(30), varchar(40), varchar(50)))
                    WITH ORDINALITY
              AS T(artikelnummer, menge, neuerpreis, counter)
              WHERE counter > 1)
              AS lieferung(artikelnummer, menge, neuerpreis)
      ON bestand.artikelnummer = lieferung.artikelnummer
      WHEN MATCHED THEN UPDATE SET
                        menge = bestand.menge + lieferung.menge,
                        preis = lieferung.neuerpreis
      WHEN NOT MATCHED THEN INSERT  (artikelnummer, menge, preis)
                            VALUES(lieferung.artikelnummer,
                                   lieferung.menge, lieferung.neuerpreis)
```

Siehe auch

DELETE, INSERT, UPDATE

8.2.3.48 OPEN - Cursor öffnen

OPEN öffnet einen Cursor, der mit DECLARE CURSOR vereinbart wurde.

- Die in der Cursorbeschreibung vorkommenden Benutzervariablen bzw. die Werte für Platzhalter einer dynamisch formulierten Cursorbeschreibung werden ausgewertet.
- Die in der Cursorbeschreibung vorkommenden Spezial-Literale (siehe "Spezial-Literale") sowie die Zeitfunktionen CURRENT_DATE, CURRENT_TIME und CURRENT_TIMESTAMP werden ausgewertet.

  Alle zurückgelieferten Werte enthalten gleiches Datum und/oder gleiche Uhrzeit (siehe Abschnitt „Zeitfunktionen“). Diese Werte gelten für die Cursortabelle, solange der Cursor offen ist, und auch, wenn der Cursor mit RESTORE wieder geöffnet wird.

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

Ein Cursor ist nur in der Übersetzungseinheit ansprechbar, in der er mit DECLARE CURSOR vereinbart wurde. Die Cursorvereinbarung mit DECLARE CURSOR muss im Programmtext vor der OPEN-Anweisung stehen.

Bei einem dynamischen Cursor muss die Cursorbeschreibung zum Ausführungszeitpunkt der OPEN-Anweisung vorbereitet sein.

Der Cursor muss geschlossen sein.

```
OPEN cursor
     [USING { variable [, variable ]... | SQL DESCRIPTOR GLOBAL deskriptor }]

variable ::= : benutzervariable [[INDICATOR] : indikatorvariable ]
```

---

cursor

Name des Cursors, der geöffnet werden soll.

USING-Klausel

Für dynamischen Cursor.

Angabe, woher die Eingabewerte für die dynamisch formulierte Cursorbeschreibung gelesen werden. Die USING-Klausel muss angegeben werden, wenn die Cursorbeschreibung Fragezeichen als Platzhalter für Werte enthält.

benutzervariable

Name einer Benutzervariable, deren Wert einem Platzhalter in der dynamisch formulierten Cursorbeschreibung zugewiesen wird.

Der Datentyp einer Benutzervariable muss mit dem Datentyp des zugehörigen Platzhalters verträglich sein (siehe Abschnitt „Werte für Platzhalter“). Steht der Platzhalter für ein Aggregat mit mehreren Elementen, muss die zugehörige Benutzervariable ein Vektor mit derselben Anzahl von Elementen sein.

Die Anzahl der angegebenen Benutzervariablen muss mit der Anzahl der Platzhalter in der Cursorbeschreibung übereinstimmen. Die Benutzervariablen werden in der angegebenen Reihenfolge den Platzhaltern in der dynamisch formulierten Cursorbeschreibung zugewiesen.

indikatorvariable

Name der Indikatorvariable für die vorangehende Benutzervariable. Ist die Benutzervariable ein Vektor, muss auch die Indikatorvariable ein Vektor mit derselben Elementanzahl sein.

Der Wert der Indikatorvariable zeigt an, ob der NULL-Wert übertragen werden soll:

< 0	Der NULL-Wert soll zugewiesen werden.
>= 0	Der Wert der Benutzervariable soll zugewiesen werden.

deskriptor

Name eines SQL-Deskriptorbereichs, der die Datentypen und Werte für die Platzhalter in der dynamisch formulierten Cursorbeschreibung enthält.

Der SQL-Deskriptorbereich muss vorher angelegt und geeignet belegt worden sein:

- Der Wert des Deskriptorbereichsfelds COUNT muss mit der Anzahl der benötigten Eingabewerte übereinstimmen (bei Aggregaten ein Eingabewert pro Element) und es muss gelten:

  0 <= COUNT <= festgelegte Maximalanzahl von Deskriptorbereichseinträgen
- Die Werte der DATA-Felder der Deskriptorbereichseinträge (bzw. NULL-Werte bei negativem INDICATOR) werden in der Reihenfolge der Einträge im Deskriptorbereich den Platzhaltern in der dynamisch formulierten Cursorbeschreibung zugewiesen. Die Datentypbeschreibung eines Eintrags muss mit dem Datentyp des zugehörigen Platzhalters verträglich sein (siehe Abschnitt „Werte für Platzhalter“).

Beispiel

Das Beispiel öffnet den Cursor CUR_KONTAKTE. Der Cursor definiert einen Ausschnitt aus der Tabelle KONTAKT, in dem NACHNAME, VORNAME und ABTEILUNG für Kundennummern > 103 ausgewählt werden.

```
DECLARE cur_kontakte CURSOR FOR

SELECT nachname, vorname, abteilung

FROM kontakt WHERE knr > 103

ORDER BY abteilung ASC, nachname DESC


OPEN cur_kontakte
```

Siehe auch

CLOSE, DECLARE CURSOR, FETCH, PREPARE

8.2.3.49 PERMIT - Benutzeridentifikation für SESAM/SQL V1.x angeben

Um mit SESAM/SQL V1.x erstellte Programme auch weiterhin in unveränderter Form ablaufen lassen zu können, ist die PERMIT-Anweisung weiterhin erlaubt. Die Ausführung einer PERMIT-Anweisung hat jedoch keinerlei Auswirkungen. Ein SESAM/SQL V1.x-Programm kann in der aktuellen Version von SESAM/SQL nur erfolgreich ausgeführt werden, wenn mit GRANT die benötigten Privilegien definiert wurden.

Die Anweisung PERMIT leitet keine Transaktion ein.

Siehe auch

GRANT, REVOKE

8.2.3.50 PREPARE - Dynamisch formulierte Anweisungen vorbereiten

PREPARE bereitet eine dynamisch formulierte Anweisung oder die Cursorbeschreibung eines dynamischen Cursors für die spätere Ausführung vor.

Eine mit PREPARE vorbereitete Anweisung wird mit EXECUTE ausgeführt.

Ein bei PREPARE verwendeter Anweisungsbezeichner für eine Cursorbeschreibung wird zur Vereinbarung eines dynamischen Cursors mit DECLARE CURSOR verwendet. Der dynamische Cursor wird mit OPEN geöffnet.

PREPARE *anweisungsbezeichner* FROM *anweisungsvariable*

anweisungsvariable ::= : *benutzervariable*

---

anweisungsbezeichner

Name der dynamisch formulierten Anweisung bzw. Cursorbeschreibung. Mit diesem Namen ist die Anweisung bzw. die Cursorbeschreibung in der Übersetzungseinheit ansprechbar.

anweisungsvariable

Alphanumerische Benutzervariable, die den Anweisungstext enthält. Für Benutzervariable ist auch ein Datentyp CHAR(*n*) zulässig, mit 256 <= *n* <= 32000.

Folgende Bedingungen müssen erfüllt sein:

- Innerhalb des Anweisungstexts dürfen keine Benutzervariablen verwendet werden. Platzhalter für noch unbekannte Eingabewerte werden durch ein Fragezeichen angegeben (siehe auch Abschnitt „Regeln für Platzhalter“). Die Platzhalter werden über die USING-Klausel einer EXECUTE- bzw. OPEN-Anweisung mit Werten versorgt.
- Der Anweisungstext darf keine Kommentare der Wirtssprache enthalten. Ausnahmen sind Pragmas (--% PRAGMA).
- Eine SELECT-Anweisung darf keine INTO-Klausel enthalten.
- Bei einer INSERT-Anweisung darf die RETURN INTO-Klausel nicht angegeben werden. Um die von statischen INSERT-Anweisungen bekannte Funktionalität nutzen zu können, steht der CLI-Aufruf SQL_DIAG SEQ_GET zur Verfügung. Damit kann RETURN INTO nachgebildet werden (siehe "SQL_DIAG_SEQ_GET - SQLdsg").

Ist *anweisungsbezeichner* für einen dynamischen Cursor definiert, die Anweisung aber keine Cursorbeschreibung, wird ein Fehler gemeldet. Die Anweisung wird trotzdem erfolgreich vorbereitet und kann mit EXECUTE ausgeführt werden.

Regeln für Platzhalter

Ein Platzhalter für einen Eingabewert in einer dynamisch formulierten Anweisung wird durch ein Fragezeichen dargestellt. Die Angabe eines Platzhalters ist erlaubt, wenn die mit dem Platzhalter verbundenen Operanden bzw. Operatoren den Datentyp des Platzhalters eindeutig festlegen.

Die erlaubten bzw. nicht erlaubten Positionen für Platzhalter sind im Folgenden nach der Stelligkeit der Operatoren sowie für CASE-Ausdruck, CAST-Ausdruck, numerische Funktionen, Zeichenkettenfunktionen, SELECT-Liste, INSERT, UPDATE und MERGE zusammengestellt. Bei den erlaubten Platzhaltern ist auch jeweils der Datentyp des Platzhalters angegeben.

Ist auf einer Position ein Platzhalter nicht erlaubt, dann gilt dies auch, wenn der Platzhalter in Klammern eingeschlossen wird.

Beispiel

```
nicht erlaubt: (?)+(?)
```

Einstelliger Operator

Bei einem einstelligen Operator sind keine Platzhalter erlaubt. Folgende Fälle sind daher nicht erlaubt:

- Der Operand eines einstelligen Operators darf kein Platzhalter sein (z.B. -?).
- Der Operand bei IS [NOT] NULL darf kein Platzhalter sein (z.B. ? IS NULL).
- Das Argument einer Mengenfunktion darf kein Platzhalter sein (z.B. AVG(?)).

Zweistelliger Operator

Bei einem zweistelligen Operator darf nur einer der beiden Operanden ein Platzhalter sein.

Beispiel

```
erlaubt: ?+1, ?<100, p=?

nicht erlaubt: ?=?
```

Datentyp des Platzhalters

Wenn bei der Konkatenation ein Operand ein Platzhalter (?||'...' bzw. '...'||?) ist, dann ist der Datentyp des Platzhalters VARCHAR(32000) oder NVARCHAR(16000).

Bei allen anderen zweistelligen Operatoren ist der Datentyp des Platzhalters derselbe wie der Datentyp des anderen Operanden.

Bereichsabfrage

Bei einer Bereichsabfrage, in der der erste Operand ein Platzhalter ist, darf keiner der beiden anderen Operanden ein Platzhalter sein.

Beispiel

```
erlaubt: ? BETWEEN 100 AND 500
        50 BETWEEN ? AND ?
nicht erlaubt: ? BETWEEN 100 AND ?
```

Datentyp des Platzhalters

Der Datentyp des Platzhalters ergibt sich aus den Datentypen der Werte der übrigen Operanden, die keine Platzhalter sind (siehe „Datentyp des Platzhalters bei CASE, BETWEEN und IN“).

Elementabfrage

Bei einer Elementabfrage dürfen nicht der erste Operand und alle Elemente der Liste Platzhalter sein.

Beispiel

```
erlaubt: ? IN ('Frankfurt','Muenchen','Hamburg')
         x IN (?,'Muenchen','Hamburg')
         ? IN ('Frankfurt',?,?)
         x IN (?,?,?)
         ? NOT IN (SELECT anr FROM leistung WHERE ltext = 'Schulung')
nicht erlaubt: ? IN (?,?,?)
```

Datentyp des Platzhalters

- Ist der erste Operand ein Platzhalter und der zweite Operand eine Unterabfrage, ist der Datentyp des Platzhalters derselbe wie der Datentyp der Ergebnisspalte.
- Ist der erste Operand ein Platzhalter und der zweite Operand eine Liste von Ausdrücken, dann ergibt sich der Datentyp des Platzhalters aus den Datentypen der Elemente der Liste, die keine Platzhalter sind (siehe „Datentyp des Platzhalters bei CASE, BETWEEN und IN“).
- Ist ein Element der Liste ein Platzhalter und der erste Operand kein Platzhalter, ist der Datentyp des Platzhalters derselbe wie der Datentyp des ersten Operanden.

Mustervergleich

Bei einem Mustervergleich dürfen der zweite und dritte Operand Platzhalter sein.

Beispiel

```
erlaubt: x LIKE ? ESCAPE ?
```

```
nicht erlaubt: ? LIKE y ESCAPE ?
```

Datentyp des Platzhalters

Der Datentyp des Platzhalters ist VARCHAR(32000) oder NVARCHAR(16000).

CASE-Ausdruck

In einem CASE-Ausdruck dürfen nicht alle Operanden Platzhalter sein. Enthält der CASE-Ausdruck eine oder mehrere THEN- bzw. ELSE-Klauseln, dürfen zusätzlich nicht alle Operanden dieser Klauseln Platzhalter sein. Folgende Fälle sind daher nicht erlaubt:

- In einem einfachen CASE-Ausdruck ist der erste Operand (Ausdruck nach CASE) ein Platzhalter und der Operand in der WHEN-Klausel ist ein Platzhalter bzw. - wenn es mehrere WHEN-Klauseln gibt - alle Operanden in den WHEN-Klauseln sind Platzhalter.
- In einem einfachen CASE-Ausdruck enthalten alle THEN-Klauseln und die ELSE-Klausel Platzhalter.

  Beispiel

```
erlaubt: CASE ?
          WHEN 1 THEN 10
          WHEN 2 THEN 20
          WHEN ? THEN 30
          WHEN ? THEN 30
          ELSE 50 END
nicht erlaubt: CASE ?
                WHEN ? THEN 10
                WHEN ? THEN 20
                WHEN ? THEN 30
                WHEN ? THEN 30
                ELSE 50 END
nicht erlaubt: CASE x
                WHEN 1 THEN ?
                WHEN 2 THEN ?
                ELSE ? END
```

- In einem CASE-Ausdruck mit Suchbedingung enthalten alle THEN-Klauseln und die ELSE-Klausel Platzhalter.

```
erlaubt: CASE
         WHEN astnr= 1 THEN ?
         WHEN astnr= 2 THEN ?
         WHEN astnr > 2 AND astnr < 5 THEN ?
         ELSE 50 END
nicht erlaubt: CASE
               WHEN astnr= 1 THEN ?
               WHEN astnr= 2 THEN ?
               WHEN astnr > 2 AND astnr < 5 THEN ?
               ELSE ? END
```

- In einem CASE-Ausdruck mit NULLIF sind beide Operanden Platzhalter (z.B. NULLIF (?,?))
- In einem CASE-Ausdruck mit COALESCE sind alle Operanden Platzhalter (z.B. COALESCE (?,?,?))

Datentyp des Platzhalters

Der Datentyp des Platzhalters in einem CASE-Ausdruck ist abhängig von den Datentypen der übrigen Operanden, die keine Platzhalter sind.

Ist ein Operand eines CASE-Ausdrucks mit NULLIF ein Platzhalter, so entspricht sein Datentyp dem Datentyp des anderen Operanden.

Sind mehrere der übrigen Operanden ohne Platzhalter, gelten folgende Regeln:

- Ist der erste Operand eines einfachen CASE-Ausdrucks ein Platzhalter und/oder enthält der CASE-Ausdruck in der/den WHEN-Klausel(n) Platzhalter als Operand(en), so ergibt sich ihr Datentyp aus den Datentypen der übrigen Operanden, die keine Platzhalter sind und die nicht Operanden der THEN- bzw. ELSE-Klausel(n) sind.
- Enthält ein CASE-Ausdruck mit Suchbedingung oder ein einfacher CASE-Ausdruck Platzhalter in der/den THEN-Klausel(n) und/oder der ELSE-Klausel, so ergibt sich ihr Datentyp aus den Datentypen der übrigen THEN- bzw. ELSE-Klausel-Operanden, die keine Platzhalter sind.
- Ist ein Operand eines CASE-Ausdrucks mit COALESCE ein Platzhalter, so ergibt sich sein Datentyp aus den Datentypen der übrigen Operanden, die keine Platzhalter sind.

Für die Berechnung des jeweiligen Platzhalter-Datentyps gelten die Regeln, die im Abschnitt „Datentyp des Platzhalters bei CASE, BETWEEN und IN“ beschrieben sind.

CAST-Ausdruck

Keine Einschränkungen

Datentyp des Platzhalters

Der Datentyp des Platzhalters in einem CAST-Ausdruck entspricht dem Datentyp des Ergebniswertes des CAST-Ausdrucks.

Numerische Funktionen

Bei der numerischen Funktion POSITION dürfen nicht beide Operanden Platzhalter sein (z.B. POSITION (? IN ?)).

Datentyp des Platzhalters

Der Datentyp des Platzhalters bei den numerischen Funktionen POSITION, OCTET_LENGTH und CHAR_LENGTH ist VARCHAR(32000) oder NVARCHAR(16000).

Der Datentyp des Platzhalters bei der numerischen Funktion JULIAN_DAY_OF_DATE ist DATE.

Zeichenkettenfunktionen

Bei Zeichenkettenfunktionen sind folgende Fälle nicht erlaubt:

- Bei den Zeichenkettenfunktionen LOWER und UPPER dürfen die Operanden keine Platzhalter sein
- Bei der Zeichenkettenfunktion TRIM dürfen der erste Operand (*zeichen*) und/oder der zweite Operand (*ausdruck*) keine Platzhalter sein (z.B. TRIM (TRAILING FROM ?))
- Bei der Zeichenkettenfunktion SUBSTRING darf der erste Operand kein Platzhalter sein (z.B. SUBSTRING ? FROM 1 FOR 5))

Datentyp des Platzhalters

Bei der Zeichenkettenfunktion SUBSTRING ist der Datentyp des Platzhalters NUMERIC(31,0).

Zeitfunktionen

Keine Einschränkungen

Datentyp des Platzhalters

Bei der Zeitfunktion DATE_OF_JULIAN_DAY ist der Datentyp des Platzhalters INTEGER.

SELECT-Liste

Bei einem SELECT-Ausdruck darf ein Element der SELECT-Liste nicht nur aus einem Platzhalter bestehen.

Beispiel

```
erlaubt: SELECT 3+? FROM ...

nicht erlaubt: SELECT ?,x,p FROM ...
```

INSERT, UPDATE, MERGE

Als Spaltenwert für eine einfache Spalte und für ein Element einer multiplen Spalte kann ein Platzhalter angegeben werden.

Beispiel

```
erlaubt:  INSERT INTO tab (x, ...) VALUES (?, ...)
          INSERT INTO t (x) VALUES <..., ?, ...>
          UPDATE tab SET x=?
          UPDATE t SET x=<..., ?, ...>
```

Datentyp des Platzhalters

Der Datentyp des Platzhalters ist der Datentyp der Spalte. Bei einer multiplen Spalte mit Dimension > 1 ist der Platzhalter auch multipel mit derselben Dimension. Ansonsten ist der Platzhalter einfach.

Datentyp des Platzhalters bei CASE, BETWEEN und IN

Bei CASE-Ausdrücken sowie bei Bereichs- und Elementabfragen ergibt sich in einigen Fällen der Datentyp des Platzhalters aus den Datentypen der übrigen Operanden bzw. Elemente, die keine Platzhalter sind. In diesen Fällen gelten folgende Regeln:

- Alle Werte der übrigen Operanden haben Datentyp CHAR:
  Der Wert des Platzhalters hat Datentyp CHAR mit der größten Länge.
- Mindestens ein Wert der übrigen Operanden hat Datentyp VARCHAR:Der Wert des Platzhalters hat Datentyp VARCHAR mit der größten bzw. größten maximalen Länge.
- Alle Werte der übrigen Operanden haben Datentyp NCHAR:
  Der Wert des Platzhalters hat Datentyp NCHAR mit der größten Länge.
- Mindestens ein Wert der übrigen Operanden hat Datentyp NVARCHAR:Der Wert des Platzhalters hat Datentyp NVARCHAR mit der größten bzw. größten maximalen Länge.
- Alle Werte der übrigen Operanden haben ganzzahligen Typ oder Festpunktzahl-Typ (INT, SMALLINT, NUMERIC, DEC):
  Der Wert des Platzhalters hat Datentyp Ganz- oder Festpunktzahl.
  - Die Nachkommastellenzahl ist die größte der Nachkommastellenzahlen der verschiedenen Werte der übrigen Operanden.
  - Die Gesamtstellenzahl ist die größte der Vorkommastellenzahlen plus die größte der Nachkommastellenzahlen der verschiedenen Werte der übrigen Operanden, höchstens jedoch 31.
- Mindestens ein Wert der übrigen Operanden hat Gleitpunktzahl-Typ (REAL, DOUBLE PRECISION, FLOAT), die anderen haben einen beliebigen numerischen Datentyp:Der Wert des Platzhalters hat den Datentyp DOUBLE PRECISION.
- Alle Werte der übrigen Operanden haben Zeitdatentyp:
  Der Wert des Platzhalters hat auch diesen Datentyp.

Umwandlung des Platzhalter-Datentyps durch CAST

Die Regeln für Platzhalter ergeben manchmal unerwünschte Datentypen für einen Platzhalter. Unerwünschte Datentypen können Sie mit Hilfe des CAST-Ausdrucks vermeiden (siehe Abschnitt „CAST-Ausdruck“).

Beispiel

In der folgenden dynamisch formulierten UPDATE-Anweisung steht der Platzhalter für eine einstellige Ganzzahl:

```
UPDATE t SET x=?+1
```

Wird dann für den Platzhalter in der USING-Klausel der EXECUTE-Anweisung der Wert 10 angegeben, ist die Ausführung nicht erfolgreich.

Um diese Datentypzuordnung zu vermeiden, kann die UPDATE-Anweisung wie folgt formuliert werden:

```
UPDATE t SET x=CAST(? AS DEC(5,0))
```

Prozeduren

Eine Prozedur kann über eine dynamisch formulierte CALL-Anweisung aufgerufen werden. Wenn eine Prozedur Parameter vom Typ OUT oder INOUT enthält, dann müssen in einer dynamisch formulierten CALL-Anweisung die korrespondierenden Argumente in Form von Platzhaltern angegeben werden.

Anweisungen für PREPARE

Folgende SQL-Anweisungen können mit PREPARE vorbereitet werden:

ALTER SPACE
ALTER STOGROUP
ALTER TABLE
CALL
COMMIT
CREATE INDEX
CREATE FUNCTION
CREATE PROCEDURE
CREATE SCHEMA
CREATE SPACE
CREATE STOGROUP
CREATE SYSTEM_USER
CREATE TABLE
CREATE USER
CREATE VIEW

DELETE
DROP FUNCTION
DROP INDEX
DROP PROCEDURE
DROP SCHEMA
DROP SPACE
DROP STOGROUP
DROP SYSTEM_USER
DROP TABLE
DROP USER
DROP VIEW
GRANT
INSERT (ohne RETURN INTO-Klausel)
MERGE
PERMIT
REORG STATISTICS
REVOKE
ROLLBACK
SELECT (ohne INTO-Klausel)
SET CATALOG
SET SCHEMA
SET SESSION AUTHORIZATION
SET TRANSACTION
UPDATE

Außer diesen SQL-Anweisungen können dynamisch formulierte Cursorbeschreibungen sowie alle Utility-Anweisungen mit PREPARE vorbereitet werden (siehe Handbuch „ SQL-Sprachbeschreibung Teil 2: Utilities“).

Folgende Anweisungen können nicht mit PREPARE vorbereitet werden:

ALLOCATE DESCRIPTOR
CLOSE

DEALLOCATE DESCRIPTOR
DECLARE CURSOR
DESCRIBE
EXECUTE
EXECUTE IMMEDIATE
FETCH
GET DESCRIPTOR
INCLUDE
OPEN
PREPARE
RESTORE
SET DESCRIPTOR
STORE
WHENEVER

Gültigkeitsdauer einer vorbereiteten Anweisung

Eine mit PREPARE vorbereitete SQL-Anweisung bleibt mindestens bis zum Ende der aktuellen Transaktion für die Ausführung vorbereitet. Nach Ende der Transaktion sollten Sie die Anweisung erneut vorbereiten. Enthält der Planpuffer des DBH noch den Zugriffsplan der in *anweisungsvariable* enthaltenen SQL-Anweisung, verwendet SESAM/SQL den bereits existierenden Zugriffsplan erneut.

Eine mit PREPARE vorbereitete Anweisung geht verloren, wenn PREPARE mit demselben *anweisungsbezeichner* in derselben Übersetzungseinheit und SQL-Session ausgeführt wird.

Die vorbereitete Anweisung geht auch verloren, wenn die Anweisung eine Referenz auf einen dynamischen Cursor enthält und die für diesen Cursor vorbereitete Cursorbeschreibung verloren geht.

Beispiel

Das Beispiel bereitet die Cursorbeschreibung des dynamischen Cursor CUR_LEISTUNG1 für die spätere Ausführung vor. Der Inhalt der Benutzer-Variable BESCHREIBUNG wird bearbeitet mit Aktionen der Wirtssprache des ESQL-Programms.

```
DECLARE cur_leistung1 CURSOR FOR curbeschreibung

PREPARE curbeschreibung FROM :BESCHREIBUNG
```

Siehe auch

DECLARE CURSOR, EXECUTE, FETCH, OPEN

8.2.3.51 REORG STATISTICS - Globale Statistik neu erzeugen

REORG STATISTICS erzeugt die globale Statistik über die Werteverteilung der Spalten eines Index neu. Diese Statistik wird zur Optimierung von Zugriffen mit Suchbedingungen auf Tabellen herangezogen und sollte daher nach umfangreichen Datenänderungen wieder aktualisiert werden.

Der aktuelle Berechtigungsschlüssel muss entweder Eigentümer des Schemas sein, zu dem der Index gehört oder das Sonder-Privileg UTILITY für die Datenbank besitzen, zu dem der Index gehört.

```
REORG STATISTICS FOR INDEX index
```

index

Name des Index, dessen Statistik neu erzeugt werden soll.

Der einfache Name des Index kann durch einen Datenbank- und Schemanamen qualifiziert werden.

Siehe auch

CREATE INDEX

8.2.3.52 REPEAT - SQL-Anweisungen in einer Schleife ausführen

Die REPEAT-Anweisung führt SQL-Anweisungen in einer Schleife solange aus bis die angegebene Bedingung zutrifft. Die Schleife endet mit der Prüfung der Bedingung, d.h. sie wird mindestens einmal durchlaufen.

Mit der ITERATE-Anweisung kann sofort zum nächsten Schleifendurchlauf gewechselt werden. Die Schleife kann über eine LEAVE-Anweisung abgebrochen werden.

Die REPEAT-Anweisung darf nur in einer Routine angegeben werden, d.h. im Rahmen einer CREATE PROCEDURE- oder CREATE FUNCTION-Anweisung. Routinen und ihre Verwendung in SESAM/SQL sind detailliert im Kapitel „Routinen“ beschrieben..

Die REPEAT-Anweisung ist eine nicht-atomare SQL-Anweisung, d.h. in ihr können weitere (atomare oder nicht-atomare) SQL-Anweisungen auftreten.

Wenn die REPEAT-Anweisung Teil einer COMPOUND-Anweisung ist, dann kann die Schleife bei entsprechenden Fehler-Routinen auch beim Auftreten eines bestimmten SQLSTATE (z.B. keine Daten, Klasse 02xxx) verlassen werden.

```
[ marke :]
REPEAT routine_sql_anweisung; [ routine_sql_anweisung; ]...
   UNTIL suchbedingung
END REPEAT [ marke ]
```

---

marke

Die Marke vor der REPEAT-Anweisung (Anfangsmarke) bezeichnet den Anfang der Schleife. Sie darf nicht identisch sein mit einer anderen Marke innerhalb der Schleife.

Die Anfangsmarke muss nur dann angegeben werden, wenn mit ITERATE zum nächsten Schleifendurchlauf gewechselt werden soll oder wenn die Schleife über eine LEAVE-Anweisung verlassen werden soll. Sie sollte aber stets verwendet werden, damit SESAM/SQL die korrekte Struktur der Routine überprüfen kann (z.B. bei geschachtelten Schleifen).

Die Marke am Ende der REPEAT-Anweisung (Endemarke) bezeichnet das Ende der Schleife. Wenn die Endemarke angegeben ist, dann muss auch die Anfangsmarke angegeben sein. Beide Marken müssen identisch sein.

suchbedingung

Suchbedingung, deren Auswertung einen Wahrheitswert ergibt.
Die Suchbedingung stellt das Abbruchkriterium für die Laufschleife dar.

routine_sql_anweisung

SQL-Anweisung, die in der REPEAT-Anweisung ausgeführt werden soll.Eine SQL-Anweisung wird mit „;“ (Semikolon) abgeschlossen.

Mehrere SQL-Anweisungen können nacheinander angegeben werden. Sie werden in der angegebenen Reihenfolge ausgeführt.
Vor der Durchführung einer SQL-Anweisung werden keine Privilegien geprüft.Eine SQL-Anweisung in einer Routine darf auf die Parameter der Routine und (wenn die Anweisung Teil einer COMPOUND-Anweisung ist) auf lokale Variablen, jedoch nicht auf Benutzervariablen zugreifen.

Syntax und Bedeutung von *routine_sql_anweisung* sind zentral im Abschnitt „SQL-Anweisungen in Routinen“ beschrieben. Die dort genannten SQL-Anweisungen dürfen verwendet werden.

Ausführungshinweise

Die REPEAT-Anweisung ist eine nicht-atomare Anweisung:

- Wenn die REPEAT-Anweisung Teil einer COMPOUND-Anweisung ist, dann gelten die dort beschriebenen Regeln, insbesondere die dort definierten Fehler-Routinen.
- Wenn die REPEAT-Anweisung **nicht** Teil einer COMPOUND-Anweisung ist und eine der SQL-Anweisungen einen SQLSTATE meldet, dann werden ggf. nur die Änderungendieser Anweisung rückgängig gemacht. Die REPEAT-Anweisung und die Routine, in der sie enthalten ist, werden abgebrochen. Die SQL-Anweisung, in der die Routine verwendet wurde, liefert den betreffenden SQLSTATE zurück.

Beispiel

Die Schleife wird so lange durchlaufen, bis die Variable i den Wert hat.

```
DECLARE i INTEGER DEFAULT 0;
...
label:
   REPEAT
       SET i= i+2;
       ...
   UNTIL i >1000
   END REPEAT
label;
```

Siehe auch

CREATE PROCEDURE, CREATE FUNCTION, ITERATE, LEAVE

8.2.3.53 RESIGNAL - Fehler in lokaler Fehler-Routine melden

RESIGNAL meldet in einer lokalen Fehler-Routine explizit einen Fehler oder einen SQLSTATE. Im Unterschied zu SIGNAL ist die Angabe eines Fehlernamens oder eines SQLSTATE optional.

RESIGNAL verwendet den Diagnosebereich der SQL-Anweisung, die die Fehler-Routine aktiviert hat, als aktuellen Diagnosebereich und trägt entsprechende Diagnoseinformationen in den aktuellen Diagnosebereich ein.

RESIGNAL ist eine der Diagnoseanweisungen in Routinen. Detaillierte Informationen zum Einsatz und zur Wirkung von RESIGNAL finden Sie im Abschnitt „Diagnoseinformationen in Routinen“.

`RESIGNAL [` *fehlername* `|` *sqlstate* `] [SET` *diagnose_info* `]`

sqlstate `::= SQLSTATE [VALUE]` *alphanumerisches_literal*

diagnose_info `::= MESSAGE_TEXT=` *message*

message `::={` *alphanumerisches_literal* `|` *lokale_variable* `|` *routinenparameter* `}`

---

fehlername

Name für einen Fehler oder einen SQLSTATE. *fehlername* wird in den lokalen Daten einer Routine definiert, siehe „Lokale Daten“.

sqlstate

Explizite Angabe eines selbst definierten SQLSTATE (alphanumerisches Literal der Länge 5), siehe Abschnitt „Selbst definierte SQLSTATEs“.

fehlername und *sqlstate* nicht angegeben:

Die Diagnoseinformationen CONDITION_IDENTIFIER und RETURNED_SQLSTATE bleiben unverändert.

MESSAGE_TEXT=*alphanumerisches_literal*

Beliebiger Informationstext (maximale Länge: 120 Zeichen).

MESSAGE_TEXT=*lokale_variable / routinenparameter*

Als Informationstext wird der Wert der lokalen Variablen oder des angegebenen Routinen-Parameters eingetragen.
Der Datentyp von *lokale_variable / routinenparameter* muss mit dem Datentyp VARCHAR(120) verträglich sein. Es gelten die Regeln im Abschnitt „Werte in Prozedurparameter (Ausgabe) oder lokale Variable eintragen “. Die Textlänge wird in MESSAGE_LENGTH und MESSAGE_OCTET_LENGTH eingetragen.

SET MESSAGE TEXT nicht angegeben:

Die Diagnoseinformationen MESSAGE_TEXT, MESSAGE_LENGTH und MESSAGE_OCTET_LENGTH bleiben unverändert.

Beispiele (siehe auch "Diagnoseinformationen in Routinen" **)**

Eine Bedingung mit Informationstext melden:

```
RESIGNAL SET MESSAGE_TEXT='The end is near!';
```

Siehe auch

COMPOUND, CREATE FUNCTION, CREATE PROCEDURE, GET DIAGNOSTICS, SIGNAL

8.2.3.54 RESTORE - Cursor wiederherstellen

RESTORE öffnet einen mit STORE gespeicherten Cursor.

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

Auch die in der Cursorbeschreibung vorkommenden Spezial-Literale sowie die Zeitfunktionen CURRENT_DATE, CURRENT_TIME und CURRENT_TIMESTAMP werden nicht neu ausgewertet.

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

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

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

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

Der wiederherzustellende Cursor muss mit STORE gespeichert und zum RESTORE-Zeitpunkt geschlossen sein. Die Transaktionen, die die STORE- und RESTORE-Anweisung enthalten, müssen denselben Isolationslevel haben.

Bei einem dynamischen Cursor muss die Cursorbeschreibung zum Ausführungszeitpunkt der RESTORE-Anweisung noch vorbereitet sein (siehe auch Abschnitt „Gültigkeitsdauer einer vorbereiteten Anweisung“).

RESTORE darf nicht auf Cursor angewendet werden, die mit WITH HOLD definiert wurden.

RESTORE *cursor*

---

cursor

Name des Cursors, der wiederhergestellt werden soll.

Cursor nach der RESTORE-Anweisung bearbeiten

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

Beispiel

FETCH NEXT positioniert auf den nächsten Satz in der Cursortabelle.

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

Siehe auch

DECLARE CURSOR, OPEN, STORE, FETCH, UPDATE, DELETE

8.2.3.55 RETURN - Rückgabewert einer User Defined Function (UDF) liefern

RETURN liefert den Rückgabewert einer UDF. Der Datentyp des Rückgabewertes ist durch die RETURNS-Klausel der CREATE FUNCTION-Anweisung festgelegt.

Die RETURN-Anweisung darf nur bei der Definition einer UDF mit CREATE FUNCTION angegeben werden. UDFs und ihre Verwendung in SESAM/SQL sind detailliert im Kapitel „Routinen“ beschrieben.

Eine RETURN-Anweisung beendet unmittelbar die Ausführung einer UDF. Wenn eine UDF nicht mit einer RETURN-Anweisung beendet wird, dann führt dies zu einem Fehler in der aufrufenden SQL-Anweisung.

```
RETURN { ausdruck | NULL }
```

ausdruck

Ausdruck, dessen Wert dem Rückgabewert der UDF zugeordnet wird.
Der Ausdruck darf Routinen-Parameter und (wenn die Anweisung Teil einer COMPOUND-Anweisung ist) lokale Variablen, jedoch keine Benutzervariablen enthalten. Die Angabe einer Spalte ist nur erlaubt, wenn sie Teil einer Unterabfrage ist. Der Datentyp von *ausdruck* muss mit dem Datentyp der RETURNS-Klausel aus der CREATE FUNCTION-Anweisung verträglich sein.
Es gelten die Regeln im Abschnitt „Werte in Prozedurparameter (Ausgabe) oder lokale Variable eintragen“.

NULL

Der Rückgabewert der UDF ist der NULL-Wert.

Siehe auch

CREATE FUNCTION

8.2.3.56 REVOKE - Privilegien entziehen

REVOKE entzieht Berechtigungsschlüsseln folgende Privilegien:

- Tabellen- und Spalten-Privilegien
- Sonder-Privilegien
- EXECUTE-Privilegien für Routinen

Privilegien können einem Berechtigungsschlüssel nur von dem Berechtigungsschlüssel entzogen werden, der das Privileg vergeben hat, dem sog. Grantor (siehe Abschnitt „GRANT - Privilegien vergeben“).

Welche Privilegien welchen Berechtigungsschlüsseln zugeordnet sind, erfahren Sie in den Tabellen TABLE_PRIVILEGES, COLUMN_PRIVILEGES, USAGE_PRIVILEGES, CATALOG_PRIVILEGES und ROUTINE_PRIVILEGES des INFORMATION_SCHEMA (siehe Kapitel „Informationsschemata“).

Die REVOKE-Anweisung hat mehrere Formate. Beispiele befinden sich beim jeweiligen Format.

Siehe auch

GRANT

REVOKE-Format für Tabellen- und Spalten-Privilegien

```
REVOKE  { ALL PRIVILEGES | tabellen_und_spalten_privileg ,...}
        ON [TABLE] tabelle
        FROM { PUBLIC | berechtigungsschlüssel },...
        { RESTRICT | CASCADE }

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

ALL PRIVILEGES

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

tabellen_und_spalten_privileg

Die Tabellen- und Spalten-Privilegien werden einzeln entzogen. Sie können mehrere Privilegien angeben.

ON [TABLE] *tabelle*

Name der Tabelle, für die Sie Privilegien entziehen wollen.

Die Tabelle kann eine Basistabelle oder ein View sein. Für einen nicht änderbaren View können Sie nur das Privileg SELECT entziehen.

FROM PUBLIC

Die Privilegien werden der Allgemeinheit entzogen. Individuelle Privilegien von Berechtigungsschlüsseln werden dadurch nicht berührt.

FROM *berechtigungsschlüssel*

Die Privilegien werden dem Benutzer mit dem Berechtigungsschlüssel *berechtigungsschlüssel* entzogen. Sie können mehrere Berechtigungsschlüssel angeben.

CASCADE

Der Grantor kann die von ihm vergebenen Privilegien uneingeschränkt entziehen:

- Alle angegebenen Privilegien werden entzogen.
- Wurde ein angegebenes Privileg an andere Berechtigungsschlüssel weitergegeben, werden alle direkt oder indirekt weitergegebenen Privilegien gelöscht.
- Views, die direkt oder indirekt auf Grund des angegebenen oder der weitergegebenen Privilegien definiert wurden, werden gelöscht.
- Referenzbedingungen, die auf Grund des angegebenen und der weitergegebenen Privilegien definiert wurden, werden gelöscht.
- Routinen, die direkt oder indirekt auf Grund des angegebenen und der weitergegebenen Privilegien definiert wurden, werden gelöscht.

RESTRICT

Für das Entziehen von Privilegien gelten Einschränkungen:

- Ein Privileg, das an andere Berechtigungsschlüssel weitergegeben wurde, kann nicht entzogen werden, solange ein solches weitergegebenes Privileg existiert.
- Ein Privileg, auf Grund dessen ein View oder eine Referenzbedingung definiert wurde, kann nicht entzogen werden, wenn der View bzw. die Referenzbedingung noch existiert.
- Ein Privileg, auf Grund dessen eine Routine definiert wurde, kann nicht entzogen werden, wenn die Routine noch existiert.

tabellen_und_spalten_privileg

Angabe der einzelnen Tabellen- und Spalten-Privilegien.

SELECT

Privileg, das das Lesen von Sätzen der Tabelle erlaubt.

DELETE

Privileg, das das Löschen von Sätzen der Tabelle erlaubt.

INSERT

Privileg, das das Einfügen von Sätzen in die Tabelle erlaubt.

UPDATE [(*spalte*,...)]

Privileg, das das Ändern von Sätzen der Tabelle erlaubt.

Der Entzug des Privilegs kann auf die angegebenen Spalten beschränkt werden.

spalte muss ein Spaltenname der angegebenen Tabelle sein. Sie können mehrere Spalten angeben.

(*spalte*,...) nicht angegeben:

Das Privileg zum Ändern aller Spalten der Tabelle wird entzogen.

REFERENCES [(*spalte*,...)]

Privileg, das die Definition von Referenzbedingungen erlaubt, die sich auf die Tabelle beziehen.

Der Entzug des Privilegs kann auf die angegebenen Spalten beschränkt werden.

spalte muss ein Spaltenname der angegebenen Tabelle sein. Sie können mehrere Spalten angeben.

(*spalte*,...) nicht angegeben:

Das Privileg zum Referenzieren aller Spalten der Tabelle wird entzogen.

REVOKE-Format für Sonder-Privilegien

```
REVOKE { ALL SPECIAL PRIVILEGES | sonder_privileg ,...}
       ON { CATALOG  catalog | STOGROUP stogroup }
```

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

    { RESTRICT | CASCADE }
```

sonder_privileg ::=

```
  {
    CREATE USER |
    CREATE SCHEMA |
    CREATE STOGROUP |
    UTILITY  |
    USAGE
  }
```

ALL SPECIAL PRIVILEGES

Alle Sonder-Privilegien werden entzogen, die der aktuelle Berechtigungsschlüssel entziehen darf. ALL SPECIAL PRIVILEGES umfasst die Sonderprivilegien.

sonder_privileg

Die Sonder-Privilegien werden einzeln entzogen. Sie können mehrere Sonder-Privilegien angeben.

ON CATALOG *catalog*

Name der Datenbank, für die Sie Sonder-Privilegien entziehen wollen.

ON STOGROUP *stogroup*

Name der Storage Group, für die Sie das Privileg USAGE entziehen wollen. Der einfache Name der Storage Group kann durch einen Datenbanknamen qualifiziert werden.

FROM *berechtigungsschlüssel*

Die Privilegien werden dem Benutzer mit dem Berechtigungsschlüssel *berechtigungsschlüssel* entzogen. Sie können mehrere Berechtigungsschlüssel angeben.

CASCADE

Der Grantor kann die von ihm vergebenen Privilegien uneingeschränkt entziehen:

- Alle angegebenen Privilegien werden entzogen.
- Wurde ein angegebenes Privileg an andere Berechtigungsschlüssel weitergegeben, werden alle weitergegebenen Privilegien implizit gelöscht.

RESTRICT

Für das Entziehen von Privilegien gelten Einschränkungen:

- Ein Privileg, das an andere Berechtigungsschlüssel weitergegeben wurde, kann nicht entzogen werden, solange ein solches weitergegebenes Privileg existiert.

sonder_privileg

Angabe der einzelnen Sonder-Privilegien.

CREATE USER

Sonder-Privileg, das das Definieren von Berechtigungsschlüsseln erlaubt. Sie dürfen das Privileg CREATE USER nur für eine Datenbank entziehen.

CREATE SCHEMA

Sonder-Privileg, das das Definieren von Datenbank-Schemata erlaubt. Sie dürfen das Privileg CREATE SCHEMA nur für eine Datenbank entziehen.

CREATE STOGROUP

Sonder-Privileg, das das Definieren von Storage Groups erlaubt. Sie dürfen das Privileg CREATE STOGROUP nur für eine Datenbank entziehen.

UTILITY

Sonder-Privileg, das die Verwendung von Utility-Anweisungen erlaubt. Sie dürfen das Privileg UTILITY nur für eine Datenbank entziehen.

USAGE

Sonder-Privileg, das die Verwendung der Storage Group erlaubt. Sie dürfen das Privileg USAGE nur für eine Storage Group entziehen.

Beispiel

Die folgende REVOKE-Anweisung entzieht dem Berechtigungsschlüssel UTIANW2 das UPDATE-Privileg für alle Spalten der Tabelle BESCHREIBUNG.


REVOKE UPDATE ON TABLE beschreibung FROM utianw2 RESTRICT

REVOKE-Format für EXECUTE-Privilegien (Prozedur)

```
REVOKE EXECUTE ON SPECIFIC PROCEDURE prozedur
```

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

{ RESTRICT | CASCADE }
```

prozedur ::= *routine*

EXECUTE ON SPECIFIC PROCEDURE *prozedur*

Name der Prozedur, für die das Privileg entzogen werden soll. Der einfache Prozedurname kann durch einen Datenbank- und Schemanamen qualifiziert werden.

FROM *berechtigungsschlüssel*

Die Privilegien werden dem Benutzer mit dem Berechtigungsschlüssel *berechtigungsschlüssel* entzogen. Sie können mehrere Berechtigungsschlüssel angeben.

CASCADE

Der Grantor kann die von ihm vergebenen Privilegien uneingeschränkt entziehen:

- Alle angegebenen Privilegien werden entzogen.
- Wurde ein angegebenes Privileg an andere Berechtigungsschlüssel weitergegeben, dann werden alle direkt oder indirekt weitergegebenen Privilegien gelöscht.
- Views, die direkt oder indirekt auf Grund des angegebenen oder der weitergegebenen Privilegien definiert wurden, werden gelöscht.
- Routinen, die direkt oder indirekt auf Grund des angegebenen oder der weitergegebenen Privilegien definiert wurden, werden gelöscht.

RESTRICT

Für das Entziehen von Privilegien gelten Einschränkungen:

- Ein Privileg, das an andere Berechtigungsschlüssel weitergegeben wurde, kann nicht entzogen werden, solange ein solches weitergegebenes Privileg existiert.
- Ein Privileg, auf Grund dessen ein View definiert wurde, kann nicht entzogen werden, wenn der View noch existiert.
- Ein Privileg, auf Grund dessen eine Routine definiert wurde, kann nicht entzogen werden, wenn die Routine noch existiert.

REVOKE-Format für EXECUTE-Privilegien (UDF)

```
REVOKE EXECUTE ON SPECIFIC FUNCTION udf
```

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

{ RESTRICT | CASCADE }
```

udf ::= *routine*

EXECUTE ON SPECIFIC FUNCTION *udf*

Name der UDF, für die das Privileg entzogen werden soll. Der einfache UDF-Name kann durch einen Datenbank- und Schemanamen qualifiziert werden.

FROM *berechtigungsschlüssel*

Die Privilegien werden dem Benutzer mit dem Berechtigungsschlüssel *berechtigungsschlüssel* entzogen. Sie können mehrere Berechtigungsschlüssel angeben.

CASCADE

Der Grantor kann die von ihm vergebenen Privilegien uneingeschränkt entziehen:

- Alle angegebenen Privilegien werden entzogen.
- Wurde ein angegebenes Privileg an andere Berechtigungsschlüssel weitergegeben, werden alle weitergegebenen Privilegien und alle aufgrund dieser Privilegien erzeugten Routinen und Views kaskadenförmig gelöscht.
- Views, die auf Grund des angegebenen Privilegs definiert wurden, werden kaskadenförmig gelöscht.
- Routinen, die auf Grund dieses Privilegs definiert wurden, werden kaskadenförmig gelöscht.

RESTRICT

Für das Entziehen von Privilegien gelten Einschränkungen:

- Ein Privileg, das an andere Berechtigungsschlüssel weitergegeben wurde, kann nicht entzogen werden, solange ein solches weitergegebenes Privileg existiert.
- Ein Privileg, auf Grund dessen ein View definiert wurde, kann nicht entzogen werden, wenn der View noch existiert.
- Ein Privileg, auf Grund dessen eine Routine definiert wurde, kann nicht entzogen werden, wenn die Routine noch existiert.

8.2.3.57 ROLLBACK WORK - Transaktion zurücksetzen

ROLLBACK WORK beendet eine SQL-Transaktion und setzt Änderungen zurück, die seit dem Ende der letzten SQL-Transaktion durchgeführt wurden. Einige Anweisungen, wie zum Beispiel SET SCHEMA, werden auch zurückgesetzt, wenn sie vor Beginn der aktuellen Transaktion aber nach Ende der letzten Transaktion ausgeführt wurden.

ROLLBACK WORK setzt folgende Änderungen zurück:

- geänderte Daten in SQL-Schemata
- mit STORE abgespeicherte Cursorpositionen
- mit SET CATALOG und SET SCHEMA gesetzte Datenbank- und Schemanamen
- mit SET SESSION AUTHORIZATION festgelegte Berechtigungsschlüssel
- Zuteilung (ALLOCATE) und Freigabe (DEALLOCATE) von SQL-Deskriptorbereichen
- gesetzte Werte in SQL-Deskriptorbereichen

Alle innerhalb der Transaktion geöffneten oder mit FETCH positionierten Cursor werden geschlossen. Dynamisch formulierte Anweisungen und Cursorbeschreibungen, die mit PREPARE vorbereitet wurden, gehen verloren.

Die Anweisung SET TRANSACTION sowie Utility-Anweisungen können nicht zurückgesetzt werden.

Mit der ersten fehlerfreien transaktionseinleitenden SQL-Anweisung nach ROLLBACK WORK beginnt eine neue SQL-Transaktion (siehe Abschnitt „COMMIT WORK - Transaktion beenden“).

```
ROLLBACK [WORK]
```

Implizite Ausführung von ROLLBACK WORK

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

- Innerhalb der aktuellen Transaktion tritt ein nicht behebbarer Fehler auf.
- Für zwei oder mehrere Transaktionen, die gleichzeitig auf bestimmte SQL-Daten zugreifen, kann der eingestellte Isolationslevel nicht anders gewährleistet werden (siehe auch „ Basishandbuch“).
- Eine Transaktion ist für lange Zeit unterbrochen und belegt Betriebsmittel, die von anderen Transaktionen benötigt werden (siehe auch „ Basishandbuch“).

Die Wirkung ist dieselbe wie beim expliziten Aufruf von ROLLBACK.

Transaktionen unter openUTM

Die Anweisung ROLLBACK WORK ist nicht zulässig, wenn Sie mit openUTM arbeiten. In diesem Fall wird die Transaktionssteuerung komplett mit UTM-Sprachmitteln durchgeführt. Wird eine UTM-Transaktion zurückgesetzt, dann wird auch die SQL-Transaktion zurückgesetzt..

CALL-DML-Transaktionen

Innerhalb einer CALL-DML-Transaktion ist die SQL-Anweisung ROLLBACK WORK nicht erlaubt (siehe Abschnitt „SQL-Anweisungen in CALL-DML-Transaktionen“).

Siehe auch

COMMIT

8.2.3.58 SELECT - Einzelnen Satz lesen

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

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

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

```
SELECT [ALL | DISTINCT] select_liste

   [INTO parameter-deklaration [, parameter-deklaration ]...

   FROM tabellenangabe ,...

   [WHERE suchbedingung ]

   [GROUP BY spalte ,...]

   [HAVING suchbedingung ]


parameter-deklaration ::=
{

  : benutzervariable [[INDICATOR] : indikatorvariable ] |
  routinenparameter |
  lokale_variable

}
```

Mit Ausnahme der INTO-Klausel sind die Klauseln der SELECT-Anweisung wie für den SELECT-Ausdruck definiert und im Abschnitt „SELECT-Ausdruck“ff beschrieben..

INTO

Nur für statische SELECT-Anweisung.

Bei einer statischen SELECT-Anweisung oder einer SELECT-Anweisung in einer Prozedur müssen Sie mit der INTO-Klausel das Ausgabeziel angeben, in die die Spaltenwerte des Ergebnissatzes ausgegeben werden. Angabe, wohin die gelesenen Werte gespeichert werden.

:benutzervariable, *routinenparameter*, *lokale_variable*

Name einer Benutzervariablen (wenn die Anweisung **nicht** Bestandteil einer Routine ist) bzw. Name eines Prozedurparameters vom Typ INOUT oder OUT oder einer lokalen Variablen (wenn die Anweisung

Bestandteil einer Routine ist). Der Spaltenwert des Ergebnissatzes wird dem angegebenen Ausgabeziel zugewiesen.

Der Datentyp muss mit dem Datentyp der zugehörigen Ergebnisspalte verträglich sein (siehe Abschnitt „Werte in Benutzervariable oder Deskriptorbereich lesen“). Ist eine Ergebnisspalte ein Aggregat mit mehreren Elementen, muss die zugehörige Benutzervariable ein Vektor mit derselben Anzahl von Elementen sein.

Die Anzahl der angegebenen Ausgabeziele muss mit der Anzahl der Spalten in der SELECT-Liste der SELECT-Anweisung übereinstimmen. Der Wert der i-ten Spalte in der SELECT-Liste wird dem i-ten Ausgabeziel in der INTO-Klausel zugewiesen. Ist der zuzuweisende Wert der NULL-Wert, wird das Ausgabeziel nicht gesetzt.

Gibt es keinen oder mehr als einen Ergebnissatz, wird kein Ausgabeziel gesetzt.

Gibt es keinen Ergebnissatz, wird ein SQLSTATE gesetzt, der mit WHENEVER NOT FOUND behandelt werden kann. Gibt es mehr als einen Ergebnissatz wird ein SQLSTATE gesetzt, der mit WHENEVER SQLERROR behandelt werden kann.

indikatorvariable

Name der Indikatorvariable, die zu der vorangehenden Benutzervariable gehört. Ist die Benutzervariable ein Vektor, muss auch die Indikatorvariable ein Vektor mit derselben Elementanzahl sein.

Die Indikatorvariable zeigt an, ob der NULL-Wert übertragen wurde oder Datenverlust aufgetreten ist:

0	Die Benutzervariable enthält den gelesenen Wert. Die Zuweisung war fehlerfrei.
-1	Der Wert, der zugewiesen werden sollte, ist der NULL-Wert.
> 0	Bei alphanumerischen Werten oder National-Werten: Der Benutzervariable wurde eine verkürzte Zeichenkette zugewiesen. Der Wert der Indikatorvariable gibt die Originallänge in Code Units an.

indikatorvariable nicht angegeben:
Ist der zuzuweisende Wert der NULL-Wert, wird ein entsprechender SQLSTATE gesetzt.

Dynamisch formulierte SELECT-Anweisung

Bei einer dynamisch formulierten SELECT-Anweisung dürfen Sie keine INTO-Klausel angeben. Die INTO-Klausel mit den Benutzervariablen bzw. dem SQL-Deskriptorbereich zur Aufnahme der Ergebniswerte geben Sie stattdessen in der EXECUTE-Anweisung an, mit der die dynamisch formulierte SELECT-Anweisung ausgeführt wird.

Beispiel

Das folgende Beispiel liest die Bezeichnung und den Mehrwertsteuersatz für die Leistung zu einer vorgegebenen Leistungsnummer und legt sie ab in den Benutzervariablen LTEXT und MWSATZ.

Die Leistungsnummer wird durch die Benutzer-Variable LNR definiert. Da die Leistungsnummerinnerhalb der Tabelle LEISTUNG eindeutig ist, ist sichergestellt, dass die Abfrage maximal einen Satz liefert.

```
SELECT ltext, mwsatz
INTO  :LTEXT INDICATOR :INDLTEXT :MWSATZ INDICATOR :INDMWSATZ
FROM leistung
WHERE lnr = :LNR
```

8.2.3.59 SET - Wert zuweisen

Die SET-Anweisung weist einem Parameter oder einer lokalen Variablen einer Routine einen Wert zu.
Sie darf nur in einer Routine angegeben werden, d.h. im Rahmen einer CREATE PROCEDURE- oder CREATE FUNCTION-Anweisung. Routinen und ihre Verwendung in SESAM/SQL sind detailliert im Kapitel „Routinen“ beschrieben.

```
SET { routinenparameter | lokale_variable } = { ausdruck | NULL }
```

---

routinenparameter

Prozedurparameter vom Typ INOUT oder OUT der aktuellen Prozedur, siehe "CREATE PROCEDURE - Prozedur erzeugen".

lokale_variable

Lokale Variable der aktuellen COMPOUND-Anweisung, siehe "COMPOUND - SQL-Anweisungen in gemeinsamem Kontext ausführen".

ausdruck

Ausdruck, dessen Wert dem Prozedurparameter oder der lokalen Variablen zugeordnet wird.
Der Ausdruck darf Routinen-Parameter und (wenn die Anweisung Teil einer COMPOUND-Anweisung ist) lokale Variablen, jedoch keine Benutzervariablen enthalten. Die Angabe einer Spalte ist nur erlaubt, wenn sie Teil einer Unterabfrage ist.
Der Datentyp des Ausdrucks muss mit dem Datentyp des Prozedurparameters oder der lokalen Variablen verträglich sein. Es gelten die Regeln im Abschnitt „Werte in Prozedurparameter (Ausgabe) oder lokale Variable eintragen“.

NULL

Dem Prozedurparameter oder der lokalen Variablen wird der NULL-Wert zugeordnet.

Beispiel

```
SET number_of_reads = (SELECT COUNT (*) FROM mytable)
```

Siehe auch

COMPOUND, CREATE FUNCTION, CREATE PROCEDURE

8.2.3.60 SET CATALOG - Datenbanknamen voreinstellen

SET CATALOG legt den voreingestellten Datenbanknamen für einfache Schemanamen fest, die in nachfolgenden mit PREPARE oder EXECUTE IMMEDIATE vorbereiteten Anweisungen vorkommen. Für alle anderen Anweisungen wird weiterhin der mit der Precompiler-Option voreingestellte Datenbankname für einfache Schemanamen ergänzt. Bis zur ersten Ausführung von SET CATALOG (oder SET SCHEMA) wird als voreingestellter Datenbankname für alle Anweisungen, der mit der Precompiler-Option festgelegte Datenbankname verwendet.

Die mit SET CATALOG festgelegte Voreinstellung wird rückgängig gemacht, wenn die unmittelbar folgende Transaktion - bei openUTM die aktuelle UTM-Transaktion - zurückgesetzt wird. Das gilt auch dann, wenn die auf SET CATALOG folgende Transaktion lediglich CALL-DML-Anweisungen enthält. Andernfalls gilt der mit SET CATALOG voreingestellte Datenbankname bis ein neuer Datenbankname mit SET CATALOG bzw. SET SCHEMA eingestellt wird oder bis zum Ende der SQL-Session. Die allgemeinen Regeln für implizite Datenbank- und Schemanamen finden Sie im Abschnitt „Qualifizierte Namen“.

Die Anweisung SET CATALOG leitet keine Transaktion ein.

`SET CATALOG` *voreingest_catalog*

voreingest_catalog `::= {` *alphanumerisches_literal* `| :` *benutzervariable* `}`

---

voreingest_catalog

Name der Datenbank, die für die aktuelle SQL-Session voreingestellt wird.

alphanumerisches_literal

Der Datenbankname wird als alphanumerisches Literal (nicht in der sedezimalen Form) angegeben.

benutzervariable

Der Datenbankname wird als alphanumerische Benutzervariable vom Typ CHAR oder VARCHAR angegeben. Die Benutzervariable darf kein Vektor sein und keine zugeordneteIndikatorvariable besitzen.

Beispiel

```
SET CATALOG 'auftragkunden'
```

Siehe auch

SET SCHEMA

8.2.3.61 SET DESCRIPTOR - SQL-Deskriptorbereich ändern

Mit SET DESCRIPTOR kann ein SQL-Deskriptorbereich geändert werden. Sie können den Wert des Deskriptorbereichsfelds COUNT ändern oder den Inhalt eines Eintrags.

Aufbau und Verwendung des Deskriptorbereichs sind im Abschnitt „Deskriptorbereich“ beschrieben.

Der SQL-Deskriptorbereich muss vorher angelegt worden sein.

```
SET DESCRIPTOR GLOBAL deskriptor

{ COUNT= anzahl |

  VALUE eintragsnummer feldbezeichner = feldinhalt [, feldbezeichner = feldinhalt ]... }


anzahl ::= { ganzzahl | benutzervariable }


eintragsnummer ::= { ganzzahl | benutzervariable }


feldbezeichner ::=

{
   REPETITIONS |
   TYPE |
   DATETIME_INTERVAL_CODE |
   PRECISION |
   SCALE |
   LENGTH |
   INDICATOR |
   DATA
}


feldinhalt ::= { benutzervariable | { anzzahl | benutzervariable }
```

---

deskriptor

Name des SQL-Deskriptorbereichs, dessen Einträge geändert werden sollen.

Sie können die Einträge dieses Deskriptorbereichs nicht ändern, wenn ein geöffneter Cursor mit eingeschaltetem Schubmodus existiert (siehe Abschnitt „Pragma PREFETCH“), und für diesen Cursor eine Anweisung FETCH NEXT... ausgeführt wurde, deren INTO-Klausel den Namen desselben SQL-Deskriptorbereichs enthält.

COUNT=*anzahl*

Das Deskriptorbereichsfeld COUNT wird auf den Wert von *anzahl* gesetzt.

anzahl

Für *anzahl* geben Sie eine Ganzzahl oder eine Benutzervariable vom SQL-Datentyp SMALLINT an, mit:

0 <= *anzahl* <= festgelegte Maximalanzahl von Deskriptorbereichseinträgen

Einträge, deren Eintragsnummer gößer als *anzahl* ist, werden auf undefiniert gesetzt.

VALUE-Klausel

Die angegebenen Felder des Eintrags mit der Eintragsnummer *eintragsnummer* werden auf die angegebenen Feldinhalte gesetzt.

Sind mehrere Felder angegeben, werden sie unabhängig von der Reihenfolge in der SET DESCRIPTOR-Anweisung in der folgenden Reihenfolge gesetzt:

REPETITIONS

TYPE

DATETIME_INTERVAL_CODE

PRECISION

SCALE

LENGTH

INDICATOR

DATA

eintragsnummer

Nummer des Eintrags, der geändert werden soll.

Die Einträge im Deskriptorbereich sind beginnend mit 1 durchnummeriert.

Für *eintragsnummer* können Sie eine Ganzzahl oder eine Benutzervariable vom SQL-Datentyp SMALLINT angeben, mit:

1 <= *eintragsnummer* <= COUNT und <= festgelegte Maximalanzahl von Einträgen

feldbezeichner

Feld des Eintrags *eintragsnummer*, das geändert werden soll. Derselbe Feldbezeichner darf höchstens einmal angegeben sein.

feldinhalt

Neuer Wert für das Feld *feldbezeichner*.

Ist *feldbezeichner* DATA, müssen Sie für *feldinhalt* eine Benutzervariable angeben. Ansonsten können Sie für *feldinhalt* eine Ganzzahl oder eine Benutzervariable vom SQL-Datentyp SMALLINT angeben. Außer für die Felder DATA und INDICATOR dürfen keine Aggregate bzw. Vektoren angegeben werden.

REPETITIONS

Der in *feldinhalt* angegebene Wert muss >= 1 und <= 255 sein.

Bei den Einträgen mit Eintragsnummer
eintragsnummer, *eintragsnummer*+1, ..., *eintragsnummer*+REPETITIONS-1,
werden die Felder TYPE, DATETIME_INTERVAL_CODE, PRECISION, SCALE und LENGTH mit jeweils denselben Werten gesetzt, vorausgesetzt die Eintragsnummern sind <= COUNT und <= festgelegte Maximalanzahl von Einträgen.

Beim Eintrag *eintragsnummer* wird REPETITIONS auf den Wert von *feldinhalt* gesetzt. Bei den Einträgen mit Eintragsnummer
eintragsnummer+1,...,*eintragsnummer*+REPETITIONS-1,
wird REPETITIONS auf 1 gesetzt.

Die restlichen Felder werden bei allen betroffenen Einträgen gesetzt wie angegeben oder werden undefiniert.

REPETITIONS nicht angegeben:
Beim Eintrag *eintragsnummer* wird REPETITIONS auf den Wert 1 gesetzt.

TYPE

Setzt das Feld TYPE. Zusätzlich wird das Feld DATETIME_INTERVAL_CODE desselben Eintrags auf undefiniert gesetzt. Abhängig vom Wert für TYPE werden außerdem die Felder PRECISION, SCALE und LENGTH desselben Eintrags auf Standardwerte gesetzt:

SQL-Datentyp	TYPE	PRECISION	SCALE	LENGTH
NVARCHAR	-42			1
NCHAR	-31			1
CHAR	1			1
NUMERIC	2	1	0	
DECIMAL	3	1	0	
INTEGER	4			
SMALLINT	5			
FLOAT	6	1		
REAL	7			
DOUBLE PRECISION	8			
DATE, TIME, TIMESTAMP	9	0		
VARCHAR	12			1

Tabelle 51: TYPE-Feld eines Deskriptoreintrags setzen

Nicht angegebene Werte sind undefiniert.

Außer REPETITIONS werden alle anderen Felder dieses Eintrags undefiniert.

DATETIME_INTERVAL_CODE

Abhängig vom Wert für DATETIME_INTERVAL_CODE wird zusätzlich der Wert für PRECISION gesetzt:

DATETIME_INTERVAL_CODE	PRECISION
1	0
2	0
3	6

Tabelle 52: DATETIME_INTERVAL_CODE-Feld eines Deskriptoreintrags setzen

Außer REPETITIONS und TYPE werden alle anderen Felder dieses Eintrags undefiniert.

PRECISION, SCALE, LENGTH

Die Felder werden in dieser Reihenfolge gesetzt.
Falls zuvor das Feld TYPE gesetzt wurde und PRECISION, SCALE oder LENGTH Standardwerte enthalten, werden diese überschrieben.

Der Inhalt des Felds DATA dieses Eintrags wird undefiniert.

INDICATOR

Ist ein Vektor mit mehreren Elementen angegeben, werden auch entsprechend viele INDICATOR-Felder der folgenden Einträge gesetzt, vorausgesetzt die Eintragsnummern dieser Einträge sind <= COUNT und <= festgelegte Maximalanzahl von Einträgen.

DATA

Der Datentyp der Benutzervariablen muss mit dem Datentyp übereinstimmen, der durch die Felder TYPE, LENGTH, PRECISION, SCALE und DATETIME_INTERVAL_CODE desselben Eintrags festgelegt ist (siehe Abschnitt „Werte zwischen Benutzervariablen und Deskriptorbereich übertragen“). Ist die angegebene Benutzervariable ein Vektor mit mehreren Elementen, dann müssen auch bei genau so vielen nachfolgenden Einträgen die Felder TYPE, LENGTH, PRECISION, SCALE und DATETIME_INTERVAL_CODE diesen Datentyp angeben und die Eintragsnummern dieser Einträge <= COUNT und <= festgelegte Maximalanzahl von Einträgen sein.

Wenn DATA und INDICATOR angegeben sind, müssen beide einfach oder Vektoren mit gleicher Anzahl von Elementen sein.

Das DATA-Feld wird gesetzt, wenn das zugehörige INDICATOR-Feld >= 0 ist. Ansonsten wird der Inhalt des DATA-Feld undefiniert.

Beispiele

Typ, Anzahl der Dezimalstellen und Anzahl der Nachkommastellen im 2. Eintrag des SQL-Deskriptorbereichs : DEMO_DESC werden verändert:

```
SET DESCRIPTOR GLOBAL :demo_desc

VALUE 2 TYPE = 2, PRECISIONS = 7, SCALE = 2
```

Die Anzahl der Einträge im SQL-Deskriptorbereich DEMO_DESC wird auf null gesetzt:

```
SET DESCRIPTOR GLOBAL :demo_desc COUNT = 0
```

Siehe auch

ALLOCATE DESCRIPTOR, DEALLOCATE DESCRIPTOR, DESCRIBE, GET DESCRIPTOR

8.2.3.62 SET SCHEMA - Schemanamen voreinstellen

SET SCHEMA legt den voreingestellten Schemanamen für einfache Namen von Integritätsbedingungen, Indizes und Tabellen fest, die in nachfolgenden mit PREPARE oder EXECUTE IMMEDIATE vorbereiteten Anweisungen vorkommen. Für alle anderen Anweisungen wird weiterhin der mit der Precompiler-Option voreingestellte Schemaname für einfache Namen von Integritätsbedingungen, Indizes und Tabellen ergänzt. Bis zur ersten Ausführung von SET SCHEMA wird als voreingestellter Schemaname für alle Anweisungen der mit der Precompiler-Option festgelegte Schemaname verwendet.

Die mit SET SCHEMA festgelegte Voreinstellung wird rückgängig gemacht, wenn die unmittelbar folgende Transaktion - bei openUTM die aktuelle UTM-Transaktion - zurückgesetzt wird. Das gilt auch dann, wenn die auf SET SCHEMA folgende Transaktion lediglich CALL-DML-Anweisungen enthält.
Andernfalls gilt der mit SET SCHEMA voreingestellte Schemaname bis ein neuer Schemaname mit SET SCHEMA eingestellt wird oder bis zum Ende der SQL-Session.

Die allgemeinen Regeln für implizite Datenbank- und Schemanamen finden Sie im Abschnitt „Qualifizierte Namen“.

Die Anweisung SET SCHEMA leitet keine Transaktion ein.

SET SCHEMA *voreingest_schema*

voreingest_schema ::= { *alphanumerisches_literal* | : *benutzervariable* }

---

voreingest_schema

Name des Schemas, das für die aktuelle SQL-Session voreingestellt wird. Der einfache Schemaname kann mit einem Datenbanknamen qualifiziert werden.
Wird der einfache Schemaname mit einem Datenbanknamen qualifiziert, wird dieser Datenbankname als voreingestellter Datenbankname verwendet, als wenn er mit SET CATALOG eingestellt worden wäre.

alphanumerisches_literal

Der Schemaname wird als alphanumerisches Literal (nicht in der sedezimalen Form) angegeben.

benutzervariable

Der Schemaname wird als alphanumerische Benutzervariable vom Typ CHAR oder VARCHAR angegeben. Die Benutzervariable darf kein Vektor sein und keine zugeordnete Indikatorvariable besitzen.

Beispiele

Beispiel aus der Beispieldatenbank:

```
SET SCHEMA 'auftragkunden.auftragsver'
```

Beispiel aus der dynamischen SQL:

Die Benutzervariable SOURCESTMT enthält die Anweisung:

```
CREATE TABLE aufstat (astnr INTEGER, astxt CHAR(15))
```

Mit den folgenden Anweisungen wird eine CREATE TABLE-Anweisung für die Tabelle AUFSTAT im Schema AUFTRAGSVER der Datenbank AUFTRAGKUNDEN ausgeführt:

```
SET SCHEMA 'auftragkunden.auftragsver'
EXECUTE IMMEDIATE :SOURCESTMT
```

Siehe auch

SET CATALOG

8.2.3.63 SET SESSION AUTHORIZATION - Berechtigungsschlüssel festlegen

SET SESSION AUTHORIZATION legt den aktuellen Berechtigungsschlüssel für die SQL-Session fest.

Der aktuelle Berechtigungsschlüssel einer SQL-Session wird entweder mit einer ESQL-Precompiler-Option oder mit der Anweisung SET SESSION AUTHORIZATION festgelegt. Wird an beiden Stellen kein Berechtigungsschlüssel festgelegt, so wird der voreingestellte Berechtigungsschlüssel `D0USER` als aktueller Berechtigungsschlüssel der SQL-Session verwendet.

Die mit SET SESSION AUTHORIZATION festgelegte Einstellung wird rückgängig gemacht, wenn die unmittelbar folgende Transaktion - bei openUTM die aktuelle UTM-Transaktion - zurückgesetzt wird. Das gilt auch dann, wenn die folgende Transaktion lediglich CALL-DML-Anweisungen enthält.
Andernfalls gilt der mit SET SESSION AUTHORIZATION festgelegte Berechtigungsschlüssel bis ein neuer Berechtigungsschlüssel mit SET SESSION AUTHORIZATION eingestellt wird oder bis zum Ende der SQL-Session.

Die Anweisung SET SESSION AUTHORIZATION leitet keine Transaktion ein und darf nur außerhalb einer SQL-Transaktion verwendet werden.

`SET SESSION AUTHORIZATION` *neuer_berechtigungsschlüssel*

neuer_berechtigungsschlüssel `::= {` *alphanumerisches_literal* `|` `:`*benutzervariable* `}`

---

neuer_berechtigungsschlüssel

Name des neuen Berechtigungsschlüssels, der für die SQL-Session gelten soll. Der neue Berechtigungsschlüssel gilt bis zur nächsten Anweisung SET SESSION AUTHORIZATION.

alphanumerisches_literal

Der neue Berechtigungsschlüssel wird als alphanumerisches Literal (nicht in der sedezimalen Form) vom Typ CHAR angegeben.

benutzervariable

Der neue Berechtigungsschlüssel wird als alphanumerische Benutzervariable vom Typ CHAR oder VARCHAR angegeben. Die Benutzervariable darf kein Vektor sein und keine zugeordnete Indikatorvariable besitzen.

Beispiele

Für die aktuelle SQL-Session wird ein neuer Berechtigungsschlüssel festgelegt. Der aktuelle UTM- bzw. BS2000-Benutzer muss einen Systemzugang mit diesem Berechtigungsschlüssel besitzen.

```
SET SESSION AUTHORIZATION 'utiverw'
```

Der Berechtigungsschlüssel für die aktuelle SQL-Session wird als Benutzervariable angegeben.

```
SET SESSION AUTHORIZATION :USER-NAME
```

8.2.3.64 SET TRANSACTION - Transaktionseigenschaften festlegen

Mit SET TRANSACTION können Sie Isolations- bzw. Konsistenzlevel und Transaktionsmodus der nachfolgenden SQL-Transaktion festlegen.

Der Isolations- bzw. Konsistenzlevel einer Transaktion gibt an, wie stark das Lesen von Sätzen in der Transaktion durch gleichzeitige Schreibzugriffe einer konkurrierenden Transaktion beeinflusst wird.

Mit dem Transaktionsmodus können Sie bestimmen, ob innerhalb der nachfolgenden Transaktion Tabellensätze nur gelesen oder auch geändert werden dürfen.

> **!** **ACHTUNG!** Wenn Sie einen Isolations- bzw. Konsistenzlevel festlegen, beeinflussen Sie auch den Grad an Parallelität und damit die Performanz: je mehr Phänomene ausgeschlossen werden, desto geringer ist der Grad an Parallelität.

Die mit SET TRANSACTION getroffenen Einstellungen sind nur für die SQL-Anweisungen der unmittelbar folgenden Transaktion gültig. Nach Ende oder Rücksetzen der Transaktion gelten wieder die Voreinstellungen (siehe Abschnitt „Voreinstellung“). Die Voreinstellungen gelten nach Ende der Transaktion auch dann wieder, wenn die auf SET TRANSACTION folgende Transaktion allein CALL-DML-Anweisungen, also keine SQL-Anweisungen enthält.

Die Anweisung SET TRANSACTION leitet keine Transaktion ein und darf nur außerhalb einer SQL-Transaktion verwendet werden.

```
SET TRANSACTION { level [[,] transaktionsmodus ] | transaktionsmodus [[,] level ] }


transaktionsmodus ::= { READ ONLY | READ WRITE }

level ::= { ISOLATION LEVEL isolation-level | CONSISTENCY LEVEL konsistenzlevel }

isolation-level ::=

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

Sie können das Komma zwischen den beiden Angaben weglassen. Soll Ihre Anwendung portierbar sein, müssen Sie allerdings das Komma setzen.

ISOLATION LEVEL

Isolationslevel einstellen.

Arbeiten mehrere Transaktionen gleichzeitig mit denselben Tabellen, können Phänomene eintreten, in denen Lesezugriffe in einer Transaktion durch gleichzeitige schreibende Zugriffe einer anderen Transaktion beeinflusst werden. Mit dem Isolationslevel legen Sie fest, welche dieser Phänomene in den nachfolgenden SQL-Transaktionen zugelassen sein sollen.

Folgende Phänomene sind von Bedeutung:

- dirty read:
  Eine Transaktion ändert einen Satz oder nimmt einen Satz neu auf. Eine zweite Transaktion liest diesen Satz, bevor die erste Transaktion die Änderung festgeschrieben hat. Wird die erste Transaktion zurückgesetzt, hat die zweite Transaktion einen Satz gelesen, der nie festgeschrieben worden ist.
- non-repeatable read:
  Eine Transaktion liest einen Satz. Bevor diese Transaktion beendet wird, ändert oder löscht eine zweite Transaktion denselben Satz und schreibt die Änderung fest. Versucht danach die erste Transaktion diesen Satz nochmals zu lesen, werden entweder geänderte Werte zurückgegeben oder ein Fehler, weil der Satz inzwischen gelöscht wurde. Die Leseoperation liefert also ein anderes Ergebnis als beim ersten Mal.
- phantom:
  Eine Transaktion liest Sätze, die eine bestimmte Suchbedingung erfüllen. Anschließend nimmt eine zweite Transaktion Sätze auf, die ebenfalls diese Suchbedingung erfüllen. Wiederholt die erste Transaktion danach die Abfrage, enthält die Ergebnistabelle auch die neu aufgenommenen Sätze.

READ UNCOMMITTED

Isolationslevel, der den geringsten Schutz vor konkurrierenden Transaktionen bietet. Alle oben beschriebenen Phänomene sind möglich. In der nachfolgenden SQL-Transaktion können gelesene Sätze noch nicht festgeschrieben sein und von anderen Transaktionen nach dem Lesen geändert werden.

READ UNCOMMITTED ist nicht zulässig, wenn gleichzeitig der Transaktionsmodus READ WRITE festgelegt wurde.

READ COMMITTED

Die Phänomene non-repeatable read und phantom können auftreten. In der nachfolgenden SQL-Transaktion können gelesene Sätze von anderen Transaktionen nach dem Lesen geändert werden. Es werden keine Sätze gelesen, die noch nicht festgeschrieben sind.

REPEATABLE READ

Das Phänomen phantom kann auftreten. Die Phänomene non-repeatable read und dirty read sind nicht möglich.

SERIALIZABLE

Vollständiger Schutz vor konkurrierenden Transaktionen ist gewährleistet. Die Phänomene dirty read, non-repeateable read und phantom können nicht auftreten. Die Existenz konkurrierender Transaktionen ist für die nachfolgende Transaktion nicht sichtbar.

CONSISTENCY LEVEL

Alternativ zum Isolationslevel bietet SESAM/SQL aus Gründen der Aufwärtskompatibilität zu früheren Versionen auch die Klausel CONSISTENCY LEVEL an. Damit definieren Sie einen Konsistenzlevel, der analog zum Isolationslevel festlegt, ob die Phänomene dirty read, non-repeatable read und phantom auftreten können.

konsistenzlevel

Vorzeichenlose Ganzzahl, mit: 0 <= *konsistenzlevel* <= 4.

Level	gesetzte Sperren	gelesene Sätze
0	Gelesene Sätze werden nicht gegen Ändern durch andere Transaktionen gesperrt.	alle Sätze, auch die von anderen Transaktionen gegen Änderungen gesperrten Sätze
1	Gelesene Sätze werden gegen Änderungen durch andere Transaktionen (bis Transaktionsende) gesperrt, außer wenn diese schon gesperrt sind.	wie bei 0
2	wie bei 0	nur die Sätze, die nicht von anderen Transaktionen gegen Ändern gesperrt sind
3	Gelesene Sätze werden gegen Änderungen durch andere Transaktionen (bis Transaktionsende) gesperrt.	wie bei 2
4	Gelesene Sätze werden wie bei 3 gesperrt. Bei nicht vorhandenen Sätzen wird durch Sperren gegen Änderungen von anderen Transaktionen sichergestellt, dass sie nicht von anderen Transaktionen eingefügt werden können.	wie bei 2

Tabelle 53: Konsistenzlevel

Die folgende Tabelle zeigt die Zuordnung von Konsistenz- zu Isolationslevel und welche Phänomene jeweils auftreten können.

Isolationslevel	Konsistenzlevel	dirty read	non-repeatable read	phantom
READ UNCOMMITTED	0	x	x	x

-	1	x	x [1]	x
READ COMMITTED	2	-	x	x
REPEATABLE READ	3	-	-	x
SERIALIZABLE	4	-	-	-

Tabelle 54: Zuordnung Isolationslevel, Konsistenzlevel und Phänomene

[1]das Phänomen non-repeatable read kann nur für Sätze auftreten, die vorher mit dirty read gelesen wurden.

READ ONLY

Transaktionsmodus READ ONLY einstellen.

Innerhalb der Transaktion sind nur lesende Datenbankzugriffe möglich. READ ONLY ist Voreinstellung beim Isolationslevel READ UNCOMMITTED bzw. den Konsistenzleveln 0 und 1.

READ WRITE

Transaktionsmodus READ WRITE einstellen.

Innerhalb der Transaktion sind lesende und schreibende Datenbankzugriffe möglich. READ WRITE ist Voreinstellung bei den Isolationsleveln READ COMMITTED, REPEATABLE READ und SERIALIZABLE bzw. bei den Konsistenzleveln 2, 3 und 4.

READ WRITE ist nicht zulässig, wenn gleichzeitig der Isolationslevel READ UNCOMMITED festgelegt wurde.

Voreinstellung

Existiert für Isolations- bzw. Konsistenzlevel ein Konnektionsmodul-Eintrag in der benutzerspezifischen Konfigurationsdatei (siehe „ Basishandbuch“), wird dieser Wert als Voreinstellung genommen. Andernfalls sind der Isolationslevel SERIALIZABLE, der Konsistenzlevel 4 und der Transaktionsmodus READ WRITE voreingestellt.

Über den Operanden MAX-ISOLATION-LEVEL der DBH-Option TRANSACTION-SECURITY kann für einen DBH der Isolationslevel REPEATABLE READ eingestellt werden. Arbeitet Ihre SQL-Anwendung mit einem derart eingestellten DBH, muss eine der folgenden Bedingungen erfüllt sein:

- Die Konfigurationsdatei muss den Konnektionsmodul-Parameter ISOL-LEVEL=REPEATABLE-READ (bzw. einen niedrigeren Isolationslevel) enthalten oder
- Über die SQL-Anweisung SET TRANSACTION müssen Sie vor jeder Transaktion den Isolationslevel auf REPEATABLE READ begrenzen.

Geltungsbereich unter openUTM

In einer UTM-Anwendung verliert die Anweisung SET TRANSACTION ihre Gültigkeit mit dem Ende der aktuellen UTM-Transaktion. Da in einer UTM-Transaktion jeweils nur eine Datenbank-Transaktion ablaufen kann, müssen SET TRANSACTION und die dazugehörige SQL-Transaktion in der gleichen UTM-Transaktion durchgeführt werden.

8.2.3.65 SIGNAL - Fehler in Routine melden

SIGNAL meldet in einer Routine explizit einen Fehler oder einen selbst definierten SQLSTATE.

SIGNAL löscht den aktuellen Diagnosebereich und trägt entsprechende Diagnoseinformationen in den aktuellen Diagnosebereich ein.

SIGNAL ist eine der Diagnoseanweisungen in Routinen. Detaillierte Informationen zum Einsatz und zur Wirkung von SIGNAL finden Sie im Abschnitt „Diagnoseinformationen in Routinen“.

`SIGNAL { `*fehlername*` | `*sqlstate*` } [SET `*diagnose_info*` ]`

sqlstate `::= SQLSTATE [VALUE]` *alphanumerisches_literal*

diagnose_info `::= MESSAGE_TEXT=` *message*

message `::= {` *alphanumerisches_literal* | *lokale_variable* | *routinenparameter* `}`

---

fehlername

Name für einen Fehler oder einen SQLSTATE. *fehlername* wird in den lokalen Daten einer Routine definiert, siehe „Lokale Daten“.

sqlstate

Explizite Angabe eines selbst definierten SQLSTATEs (alphanumerisches Literal der Länge 5), siehe Abschnitt „Selbst definierte SQLSTATEs“.

MESSAGE_TEXT=*alphanumerisches_literal*

Beliebiger Informationstext (maximale Länge: 120 Zeichen). Die Textlänge wird in MESSAGE_LENGTH und MESSAGE_OCTET_LENGTH eingetragen.

MESSAGE_TEXT=*lokale_variable / routinenparameter*

Als Informationstext wird der Wert der lokalen Variablen oder des angegebenen Routinen-Parameters eingetragen.
Der Datentyp von *lokale_variable / routinenparameter* muss mit dem Datentyp VARCHAR(120) verträglich sein. Es gelten die Regeln im Abschnitt „Werte in Prozedurparameter (Ausgabe) oder lokale Variable eintragen “. Die Textlänge wird in MESSAGE_LENGTH und MESSAGE_OCTET_LENGTH eingetragen.

SET MESSAGE TEXT nicht angegeben:

Die Diagnoseinformationen MESSAGE_TEXT, MESSAGE_LENGTH und MESSAGE_OCTET_LENGTH werden mit den entsprechenden NULL-Werten versorgt.

Beispiele (siehe auch "Diagnoseinformationen in Routinen" **)**

Einen selbstdefinierten SQLSTATE melden:

```
SIGNAL SQLSTATE VALUE '46SA5';
```

Eine Bedingung mit Informationstext melden:

```
SIGNAL end_job SET MESSAGE_TEXT='The end is near!';
```

Siehe auch

COMPOUND, CREATE FUNCTION, CREATE PROCEDURE, GET DIAGNOSTICS, RESIGNAL

8.2.3.66 STORE - Cursorposition speichern

STORE speichert die aktuelle Cursorposition.

Am Ende einer Transaktion werden alle geöffneten Cursor geschlossen. Um in der folgenden Transaktion trotzdem wieder auf den Inhalt der Ergebnistabelle zugreifen zu können, muss vor Transaktionsende mit STORE die aktuelle Cursorposition gespeichert werden. Ein gespeicherter Cursor kann mit RESTORE wiederhergestellt werden.

Nach STORE ist kein FETCH mehr möglich.

Der Cursor muss geöffnet sein.

STORE ist nicht erlaubt, wenn für den geöffneten Cursor der Schubmodus eingeschaltet ist (siehe Abschnitt „Pragma PREFETCH“).

STORE darf nicht auf Cursor angewendet werden, die mit WITH HOLD definiert wurden.

`STORE` *cursor*

---

cursor

Name des Cursors, dessen Position gespeichert wird.

Der Aufruf überschreibt eine zuvor mit STORE gespeicherte Cursorposition für denselben Cursor.

Siehe auch

DECLARE CURSOR, OPEN, RESTORE

8.2.3.67 UPDATE - Spaltenwerte ändern

UPDATE ändert Spaltenwerte von Sätzen in einer Tabelle.
Bei partitionierten Tabellen darf der Primärschlüsselwert nicht geändert werden.

Die in der UPDATE-Anweisung (und in Voreinstellungen) vorkommenden Spezial-Literale (siehe "Spezial-Literale") sowie die Zeitfunktionen CURRENT_DATE, CURRENT_TIME und CURRENT_TIMESTAMP werden einmal ausgewertet, und die berechneten Werte gelten für alle Änderungen.

Um einen Satz in der angegebenen Tabelle zu ändern, müssen Sie Eigentümer dieser Tabelle sein oder das UPDATE-Privileg für jede zu ändernde Spalte besitzen. Zusätzlich muss der Transaktionsmodus der aktuellen Transaktion READ WRITE sein.

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

```
UPDATE tabelle [[AS] korrelationsname ]
    SET spaltenangabe = spaltenwert [, spaltenangabe = spaltenwert ]...
  [WHERE { suchbedingung | CURRENT OF cursor }]

spaltenangabe ::= { spalte | spalte ( posnr ) | spalte ( min..max ) }
spaltenwert ::= { ausdruck | <{ wert | NULL},...> | DEFAULT | NULL }
```

---

tabelle

Name der Tabelle, in der Sätze geändert werden sollen. Die Tabelle kann eine Basistabelle oder ein änderbarer View sein.

korrelationsname

Tabellenname, der innerhalb dieser Anweisung eine Umbenennung für *tabelle* ist.

Bei jeder Spaltenangabe, die sich auf *tabelle* bezieht, müssen Sie den Spaltennamen mit dem neuen Namen *korrelationsname* qualifizieren, wenn der Spaltenname nicht eindeutig ist.

Der neue Name muss eindeutig sein, d.h. *korrelationsname* darf nur einmal in einer Tabellenangabe dieser Anweisung vorkommen.

Sie müssen eine Tabelle umbenennen, wenn die Spalten der Tabelle ohne Umbenennung nicht eindeutig angegeben werden können.

Außerdem können Sie eine Tabelle umbenennen, um durch entsprechende Namen einen Ausdruck verständlicher zu formulieren oder um lange Namen abzukürzen.

spalte

Name einer einfachen Spalte, deren Inhalt geändert werden soll. Die Spalte muss in der Tabelle enthalten sein. Eine Spalte darf nur einmal innerhalb einer UPDATE-Anweisung vorkommen.

spalte(*posnr)*

Element einer multiplen Spalte, dessen Wert geändert werden soll.

Die multiple Spalte muss in der Tabelle enthalten sein. Werden mehrere Elemente einer multiplen Spalte angegeben, muss der Teilbereich aus den angegebenen Spaltenelementen lückenlos sein. Jedes Element darf genau einmal vorkommen.

Für *posnr* geben Sie eine vorzeichenlose Ganzzahl >= 1 an.

spalte(*min..max*)

Teilbereich von Spaltenelementen einer multiplen Spalte, die mit Werten belegt werden sollen. Die multiple Spalte muss in der Tabelle enthalten sein. Werden mehrere Elemente einer multiplen Spalte angegeben, muss der Teilbereich aus den angegebenen Spaltenelementen lückenlos sein.
Jedes Element darf genau einmal vorkommen.

Für *min* und *max* geben Sie vorzeichenlose Ganzzahlen >= 1 an; *max* muss >= *min* sein.

ausdruck

Ausdruck, dessen Wert der vorangehenden einfachen Spalte zugeordnet wird. Der Wert des Ausdrucks muss mit dem Datentyp der Spalte verträglich sein (siehe Abschnitt „Werte in Tabellenspalten eintragen“).

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

Für *ausdruck* gelten folgende Einschränkungen:

- Weder die *tabelle* zu Grunde liegende Basistabelle noch ein View auf diese Basistabelle dürfen in der FROM-Klausel einer Unterabfrage stehen, die in *ausdruck* vorkommt.
- Mengenfunktionen (AVG, MAX, MIN, SUM, COUNT) sind nicht erlaubt.

<{ *wert* , NULL},...>

Aggregat, das einer multiplen Spalte zugewiesen werden soll.

Die Anzahl der Ausprägungen muss mit der Anzahl der Spaltenelemente übereinstimmen. Der Datentyp von *wert* muss mit dem Datentyp der Zielspalte verträglich sein (siehee Abschnitt „Werte in Tabellenspalten eintragen“).

DEFAULT

Nur für einfache Spalte.

Die zugehörige Spalte wird mit dem voreingestellten Wert belegt, wenn eine Voreinstellung für die Spalte definiert ist. Sonst wird sie mit dem NULL-Wert belegt.

NULL

Der vorangehenden Spalte wird der NULL-Wert zugewiesen.

WHERE-Klausel

Die WHERE-Klausel gibt an, welche Sätze geändert werden.

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

suchbedingung

Bedingung, die die zu ändernden Sätze erfüllen müssen. Ein Satz wird nur geändert, wenn er die angegebene Suchbedingung erfüllt.

Für *suchbedingung* gelten folgende Einschränkungen:

- Spaltenangaben in *suchbedingung* außerhalb von Unterabfragen dürfen sich nur auf die angegebene Tabelle beziehen.
- Weder die *tabelle* zu Grunde liegende Basistabelle noch ein View auf diese Basistabelle dürfen in der FROM-Klausel einer Unterabfrage stehen, die in *suchbedingung* vorkommt.

Wenn kein Satz die Suchbedingung erfüllt, wird kein Satz verändert und ein SQLSTATE gesetzt, der mit WHENEVER NOT FOUND behandelt werden kann.

CURRENT OF *cursor*

Name des Cursors, über den der zu ändernde Satz bestimmt wird. *tabelle* muss die in der ersten FROM-Klausel der Cursorbeschreibung angegebene Tabelle sein.

Für den Cursor gelten folgende Bedingungen:

- Der Cursor muss sich auf die Tabelle *tabelle* beziehen.
- Der Cursor muss änderbar sein.
- Der Cursor muss zum Ausführungszeitpunkt der UPDATE-Anweisung geöffnet und mit FETCH auf einen Satz der Tabelle positioniert sein. Zusätzlich muss die FETCH-Anweisung in derselben Transaktion erfolgt sein wie die UPDATE-Anweisung..

UPDATE ändert den Satz, auf den der Cursor *cursor* zeigt.

UPDATE ist nicht erlaubt, wenn für den Cursor *cursor* der Schubmodus eingeschaltet ist (siehe Abschnitt „Pragma PREFETCH“).

Wurde der Cursor *cursor* mit der FOR UPDATE-Klausel und Spaltenangaben vereinbart, können nur die dort angegebenen Spalten geändert werden.

Die UPDATE-Anweisung beeinflusst die Position des Cursors nicht. Soll der nächste Satz der Ergebnistabelle geändert werden, müssen Sie den Cursor mit FETCH auf diesen Satz positionieren.

Werte einer multiplen Spalte ändern

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

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

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

> **!** **ACHTUNG!** Die Position eines Elements innerhalb der multiplen Spalte kann sich ändern. Wenn ein Element mit einer niedrigeren Position auf den NULL-Wert gesetzt wird, werden alle nachfolgenden Elemente nach links verschoben und der NULL-Wert hinten angehängt.

UPDATE und Integritätsbedingungen

Durch Angabe von Integritätsbedingungen bei der Definition der Basistabelle können Sie den möglichen Inhalt von *tabelle* einschränken. Nach allen Änderungen durch die UPDATE-Anweisung muss der Inhalt von *tabelle* den definierten Integritätsbedingungen genügen.

UPDATE und änderbarer View

Ist bei der Definition eines änderbaren View CHECK OPTION angegeben, können nur Sätze in den View eingefügt werden, die den Abfrage-Ausdruck der View-Definition erfüllen.

UPDATE und Transaktionssicherung

UPDATE leitet außerhalb von Routinen eine Transaktion ein, wenn keine Transaktion offen ist. Durch die Definition eines Isolationslevels bei konkurrierenden Transaktionen können Sie steuern, welche Auswirkungen die UPDATE-Anweisung auf konkurrierende Transaktionen hat (siehe Abschnitt „SET TRANSACTION - Transaktionseigenschaften festlegen“).
Tritt während der UPDATE-Anweisung ein Fehler auf, so werden sämtliche bereits durchgeführten Änderungen rückgängig gemacht.

Beispiele

Mindestbestand aller Artikel auf 20 erhöhen.

```
UPDATE artikel
SET    minbestand = 20
WHERE  minbestand < 20
```

Änderung des Mindestbestands unter Verwendung eines Cursors:

```
DECLARE cur_artikel CURSOR FOR
   SELECT minbestand FROM artikel WHERE minbestand < 20
   FOR UPDATE
```

```
OPEN cur_artikel
```

Mit einer Folge von FETCH- und UPDATE-Anweisungen können die betroffenen Sätze geändert werden.

```
FETCH cur_artikel INTO :MINBESTAND

UPDATE artikel SET minbestand = 20 WHERE CURRENT OF cur_artikel
```

Der Cursor CUR_MWST wählt alle Leistungen aus, für die keine Mehrwertsteuer berechnet wird. Mit einer Folge von FETCH- und UPDATE-Anweisungen können die betroffenen Sätzegeändert werden.

```
DECLARE cur_mwsatz CURSOR WITH HOLD FOR
   SELECT lnr, ltext, mwsatz
   FROM leistung WHERE mwsatz=0.00
   FOR UPDATE

OPEN cur_mwsatz

FETCH NEXT cur_mwsatz
      INTO :LNR, :LTEXT  INDICATOR :INDLTEXT, :MWSATZ INDICATOR :INDMWSATZ

UPDATE leistung SET mwsatz =0.15 WHERE CURRENT OF cur_mwst
...
```

In der Tabelle FARBTAB wird für die Farbe orange die Intensität der einzelnen Farbkomponentengeändert. Die Spalte RGB für die Farbintensität ist eine multiple Spalte:

```
UPDATE farbtab SET rgb(1..3) = <0.8, 0.4, 0> WHERE farbname = 'orange'
```

Siehe auch

DELETE, INSERT, MERGE

8.2.3.68 WHENEVER - Fehlerbehandlung definieren

WHENEVER legt die Reaktion auf Anweisungen fest, die mit einem SQLSTATE '00000' und '01*xxx*' beendet wurden.

WHENEVER ist keine ausführbare Anweisung.

Die WHENEVER-Anweisung kann mehrmals in einem Programm vorkommen. Die Angaben einer WHENEVER-Anweisung gelten für alle im Programmtext (nach Einfügen aller Includes) folgenden SQL- und Utility-Anweisungen bis zur nächsten WHENEVER-Anweisung für dieselbe Fehlerklasse.

Vor der ersten WHENEVER-Anweisung gilt WHENEVER...CONTINUE.

```
WHENEVER
{ SQLERROR | NOT FOUND }
{ CONTINUE | { GOTO | GO TO }[:] sprungziel }
```

SQLERROR

Reaktion festlegen für:

SQLSTATE '00000', '01*xxx*' und '02*xxx*'.

NOT FOUND

Reaktion festlegen für:

SQLSTATE = '02*xxx*'.

CONTINUE

Programm wird nach SQLERROR bzw. NOT FOUND mit der folgenden Anweisung fortgesetzt. Mit CONTINUE kanns eine zuvor vereinbarte Aktion für dieselbe Fehlerklasse wieder aufgehoben werden.

Wenn ein Programmabschnitt für die Fehlerbehandlung auch SQL-Anweisungen enthält, sollte dieser Abschnitt mit einer WHENEVER-Anweisung mit CONTINUE-Klausel eingeleitet werden. Damit kann eine Endlosschleife bei erneutem Auftreten eines Fehlers vermieden werden.

sprungziel

Sprungziel in einem ESQL-Programm.

Die Klausel entspricht einer Sprunganweisung in der Wirtssprache (z.B. GO TO in COBOL).

sprungziel muss die Namenskonventionen für Sprungmarken der jeweiligen Wirtssprache erfüllen (siehe Handbuch „ ESQL-COBOL für SESAM/SQL-Server“).

Programm wird nach SQLERROR bzw. NOT FOUND an der durch *sprungziel* festgelegten Stelle fortgesetzt.

Der Doppelpunkt : wird nur noch wegen Aufwärtskompatibilität unterstützt.

Beispiele

Das Beispiel setzt das Programm fort mit dem Paragrafen SQLERR nach einer Anweisung, die beendet wurde mit einem SQLSTATE '00000', '01*xxx*' und '02*xxx*'.

 WHENEVER SQLERROR GOTO SQLERR

Das nächste Beispiel zeigt den Einsatz der WHENEVER-Anweisung bei dem Lesen einer Cursortabelle mit FETCH. Vor dem Positionieren des Cursors wird ein Sprungziel vereinbart für den Fall, dass die angegebene Cursor-Position nicht existiert. Der Cursor wird innerhalb einer Schleife auf den jeweils nächsten Satz positioniert. Die Sätze werden gelesen, bis das Ende der Cursortabelle erreicht ist. Existiert die angegebene Position nicht, wird ein entsprechender SQLSTATE gesetzt und das Programm an dem Sprungziel fortgesetzt, dass durch die WHENEVER-Anweisung festgelegt wurde.

Am Sprungziel wird die vereinbarte Aktion für die Fehlerbehandlung wieder aufgehoben.

```
      WHENEVER NOT FOUND GOTO F-4
F-2.
      FETCH cur_kontakte
            INTO :NACHNAME,
                 :VORNAME INDICATOR :INDVORNAME,
                 :ABTEILUNG INDICATOR :INDABTEILUNG
F-3.
      Ausgeben des gelesenen Satzes, zu F-2. gehen
F-4.
      WHENEVER NOT FOUND CONTINUE
```

8.2.3.69 WHILE - SQL-Anweisungen in einer Schleife ausführen

Die WHILE-Anweisung führt SQL-Anweisungen in einer Schleife aus solange die angegebene Suchbedingung zutrifft. Die Schleife beginnt mit einer Prüfung, d.h. sie kann schon vor dem ersten Durchlauf beendet sein.

Mit der ITERATE-Anweisung kann sofort zum nächsten Schleifendurchlauf gewechselt werden. Die Schleife kann über eine LEAVE-Anweisung abgebrochen werden.

Die WHILE-Anweisung darf nur in einer Routine angegeben werden, d.h. im Rahmen einer CREATE PROCEDURE- oder CREATE FUNCTION-Anweisung. Routinen und ihre Verwendung in SESAM/SQL sind detailliert im Kapitel „Routinen“ beschrieben.

Die WHILE-Anweisung ist eine nicht-atomare SQL-Anweisung, d.h. in ihr können weitere (atomare oder nicht-atomare) SQL-Anweisungen auftreten.

Wenn die WHILE-Anweisung Teil einer COMPOUND-Anweisung ist, dann kann die Schleife bei entsprechenden Fehler-Routinen auch beim Auftreten eines bestimmten SQLSTATE (z.B. keine Daten, Klasse 02xxx) verlassen werden.

```
[ marke :]
WHILE suchbedingung
    DO routine_sql_anweisung; [ routine_sql_anweisung; ]...
END WHILE [ marke ]
```

---

marke

Die Marke vor der WHILE-Anweisung (Anfangsmarke) bezeichnet den Anfang der Schleife. Sie darf nicht identisch sein mit einer anderen Marke innerhalb der Schleife.

Die Anfangsmarke muss nur dann angegeben werden, wenn mit ITERATE zum nächsten Schleifendurchlauf gewechselt werden soll oder wenn die Schleife über eine LEAVE-Anweisung verlassen werden soll. Sie sollte aber stets verwendet werden, damit SESAM/SQL die korrekte Struktur der Routine überprüfen kann (z.B. bei geschachtelten Schleifen).

Die Marke am Ende der WHILE-Anweisung (Endemarke) bezeichnet das Ende der Schleife. Wenn die Endemarke angegeben ist, dann muss auch die Anfangsmarke angegeben sein. Beide Marken müssen identisch sein.

suchbedingung

Suchbedingung, deren Auswertung einen Wahrheitswert ergibt.

routine_sql_anweisung

SQL-Anweisung, die in der WHILE-Anweisung ausgeführt werden soll.
Eine SQL-Anweisung wird mit „;“ (Semikolon) abgeschlossen.
Mehrere SQL-Anweisungen können nacheinander angegeben werden. Sie werden in der angegebenen

Reihenfolge ausgeführt.
Vor der Durchführung einer SQL-Anweisung werden keine Privilegien geprüft.Eine SQL-Anweisung in einer Routine darf auf die Parameter der Routine und (wenn die Anweisung Teil einer COMPOUND-Anweisung ist) auf lokale Variablen, jedoch nicht auf Benutzervariablen zugreifen.

Syntax und Bedeutung von *routine_sql_anweisung* sind zentral im Abschnitt „SQL-Anweisungen in Routinen“ beschrieben. Die dort genannten SQL-Anweisungen dürfen verwendet werden.

Ausführungshinweise

Die WHILE-Anweisung ist eine nicht-atomare Anweisung:

- Wenn die WHILE-Anweisung Teil einer COMPOUND-Anweisung ist, dann gelten die dort beschriebenen Regeln, insbesondere die dort definierten Fehler-Routinen.
- Wenn die WHILE-Anweisung **nicht** Teil einer COMPOUND-Anweisung ist und eine der SQL-Anweisungen einen SQLSTATE meldet, dann werden ggf. nur die Änderungen dieser SQL-Anweisung rückgängig gemacht. Die WHILE-Anweisung und die Routine, in der sie enthalten ist, werden abgebrochen. Die SQL-Anweisung, in der die Routine verwendet wurde, liefert den betreffenden SQLSTATE zurück.

Beispiel

Die Schleife wird solange durchlaufen, wie die Variable i einen Wert < 100 hat.

```
DECLARE i INTEGER DEFAULT 0;
   label:
   WHILE i < 100
   DO
       SET i = i+1;
       ...
   END WHILE label;
```

Siehe auch

CREATE PROCEDURE, CREATE FUNCTION, ITERATE, LEAVE

9 SESAM-CLI

Dieses Kapitel beschreibt die Schnittstelle SESAM-CLI (**C**all **L**evel **I**nterface) und gliedert sich in zwei Teile:

- Abschnitt „Konzept des SESAM-CLI“, in dem die Struktur der CLI-Aufrufe, die verwendeten Datentypen und das Verhalten bei Transaktionen erklärt werden.
- Abschnitt „Aufrufe des SESAM-CLI“, der die Funktion und Syntax der Aufrufe in alphabetischer Reihenfolge beschreibt.

9.1 Konzept des SESAM-CLI

Das SESAM-CLI (**C**all **L**evel **I**nterface) ist eine prozedurale Schnittstelle, mit der in erster Linie auf BLOBs (**B**inary **L**arge **Ob**jects) zugegriffen wird. Daneben existiert zur Zeit ein weiterer CLI-Aufruf, mit dem Sie Attributwerte bei dynamischen INSERT-Anweisungen bestimmen können.

BLOBs sind Folgen von Bytes variabler Länge, die bis zu 2^{31}-1 Bytes groß sein können. Sie können mit SESAM /SQL als BLOB-Objekte persistent in Datenbanken gespeichert werden. Ihre Darstellung und Änderung erfolgt außerhalb von SESAM/SQL mit speziellen Programmen, wie zum Beispiel mit Microsoft ™ Word.

Der Wert eines solchen BLOB-Objekts wird als BLOB-Wert bezeichnet. BLOB-Objekte können Sie ausschließlich in besonderen Tabellen, den BLOB-Tabellen, speichern. Diese erzeugen Sie mit der SQL-Anweisung CREATE TABLE *tabelle* OF BLOB (siehe "CREATE TABLE - Basistabelle erzeugen"). Die BLOB-Werte einer BLOB-Tabelle bilden die Objekte einer Klasse.

Wird ein BLOB-Objekt erzeugt, werden sein Wert und die zugeordneten Attribute in einer BLOB-Tabelle gespeichert. Zusätzlich wird ein eindeutiger REF-Wert erzeugt, der das BLOB-Objekt bezeichnet, solange es existiert. Dieser REF-Wert kann in REF-Spalten beliebiger Basistabellen gespeichert werden. Der BLOB-Wert selbst wird stückweise in mehreren Zeilen der BLOB-Tabelle gespeichert. Diese Speichermethode ermöglicht einen effizienten sequenziellen Zugriff auf die BLOB-Werte.

Die Attribute eines BLOB-Objekts sind Eigenschaften, die einerseits der Anwender selbst festlegt und andererseits SESAM/SQL dem Objekt zuordnet
(siehe Abschnitt „CREATE TABLE - Basistabelle erzeugen“).

BLOB-Objekte, Klassen und Attribute von BLOB-Objekten, BLOB-Werte und Sequenzen von BLOB-Werten werden mit Hilfe der Schnittstelle SESAM-CLI angesprochen. Die einzelnen CLI-Aufrufe werden ausführlich im Abschnitt „Aufrufe des SESAM-CLI“ beschrieben.

Die inhaltliche Bearbeitung der BLOB-Werte erfolgt nicht in SESAM/SQL sondern in den objektspezifischen Programmen, wie zum Beispiel in MS Word bei Worddokumenten. Für den Transfer der BLOB-Werte aus SESAM /SQL zum Beispiel in eine BS2000-Datei stehen zwei verschiedene Möglichkeiten zur Verfügung:

- Der BLOB-Wert kann mit SQL_BLOB_VAL_GET komplett in einen Speicherbereich geschrieben werden.
- Mit der Befehlsfolge SQL_BLOB_VAL_OPEN, SQL_BLOB_VAL_FETCH und SQL_BLOB_VAL_CLOSE kann der BLOB-Wert stückweise ausgelesen werden (siehe im Abschnitt „Alphabetischer Nachschlageteil“).

> **i** Für die Funktionen SQL_BLOB_VAL_OPEN, SQL_BLOB_VAL_FETCH, SQL_BLOB_VAL_STOW und SQL_BLOB_VAL_CLOSE gibt es jeweils eine Variante mit dem Suffix _STATELESS. Mit diesem Satz von Funktionen kann die stückweise Bearbeitung von BLOBs auch über UTM-Dialogschrittwechsel hinweg erfolgen. Die Schnittstellen dieser Funktionen sind in den Dateien *sqlblox.h* (für C) und *SQLBLOX* (für COBOL) beschrieben.

Die Beispieldatenbank von SESAM/SQL (siehe „ Basishandbuch“) enthält Tabellen zur Verwaltung von BLOB-Objekten. Dort finden Sie auch ein ESQL-Programm mit C-Funktionen zur Bearbeitung dieser BLOB-Objekte.

9.1.1 Struktur von SESAM-CLI-Aufrufen

CLI-Aufrufe können aus C- oder COBOL-Programmen erfolgen.

Alle Aufrufe des SESAM-CLI haben folgende Grundstruktur:

---

cli_aufruf `::=` *cli_prozedur_name* `(` *cli_parameter* `[,` *cli_parameter* `]...)`

cli_parameter `::=` *cli_parameter_name* `{ IN | IN OUT | OUT }` *cli_parameter_datentyp*

cli_parameter_datentyp `::= { BOOLEAN | INTEGER | CHAR(` *max* `) | POINTER | SQLda }`

---

cli_prozedur_name

Name der aufrufenden CLI-Prozedur. Für jede Prozedur gibt es einen langen und einen kurzen Namen. In C sollte die Langform verwendet werden. In COBOL muss die Kurzform der Namen verwendet werden (siehe alphabetischer Nachschlageteil).

cli_parameter_name

Name des Parameters (siehe alphabetischer Nachschlageteil).

`IN`, `IN OUT` bzw. `OUT`

unterscheidet, ob es sich um einen Ein- oder Ausgabeparameter handelt. Ist `IN OUT` angegeben, so handelt es sich um einen Ein- und Ausgabeparameter.

cli_parameter_datentyp

Name des Parameterdatentyps.

Die sprachspezifische Syntax von C und COBOL für die einzelnen CLI-Aufrufe wird im Abschnitt „Aufrufe des SESAM-CLI“ beschrieben.

Korrespondierende Datentypen

Die folgende Tabelle zeigt, welche Datentypen aus C und COBOL den SQL-Datentypen der CLI-Routine entsprechen. Die Spalte „Art“ gibt an, ob es sich um einen Ein- oder Ausgabeparameter handelt. Die Spalte „Größe“ gibt die Länge der Werte in Byte an.

SQL-Datentyp	Art	COBOL-Datentyp	C-Datentyp	Größe
BOOLEAN	IN	PIC S9(p) mit der USAGE-Klausel COMP	long int const *	4
	OUT		long int *	

INTEGER	IN	PIC S9(p) mit der USAGE-Klausel COMP	long int const *	4
	OUT		long int *	
	IN OUT			
CHAR(n)	IN	PIC X(n)	char const *	n
	OUT		char *	
	IN OUT			
POINTER	IN	PIC X(n) oder COBOL group item	char *	4
CHAR(ddd)	OUT	SQLda	SQLda_t *	ddd

Tabelle 55: Korrespondierende Datentypen

Der Wert „p“ im Datentyp PIC S9(p) muss im Bereich zwischen 5 und 9 liegen.

Die SYNCHRONIZED-Klausel brauchen Sie in COBOL für die Argumente von CLI-Aufrufen nicht anzugeben. Der Datentyp POINTER dient zur Übergabe einer Adresse eines Pufferbereichs, dessen Länge durch einen anderen Parameter bestimmt wird.

Der COBOL-Datentyp SQLda

SQLda ist der Diagnosebereich in SESAM/SQL. Dieser Diagnosebereich hat in COBOL folgenden Aufbau:

```
01  SQLda
    .
    05      SQLda01             PIC     S9(4)       BINARY.
            88 SQLda01val                 value 910.
    05      SQLda02             PIC     S9(4)       BINARY.
    05      SQLda03             PIC     S9(4)       BINARY.
    05      SQLda04             PIC     S9(4)       BINARY.
    05      SQLerrline          PIC     S9(4)       BINARY.
    05      SQLerrcol           PIC     S9(4)       BINARY.
    05      SQLda07             PIC     S9(4)       BINARY.
    05      SQLda08             PIC     X(5).
    05      SQLCLI-SQLSTATE redefines SQLda08 PIC X(5).
    05      SQLerrm             PIC     X(240)
                                        .
    05      SQLda10             PIC     X.
    05      SQLda21             PIC     S9(9)       BINARY.
    05      SQLda22             PIC     X(4).
```

	05	SQLda23	PIC	S9(4)	BINARY.
	05	SQLda24	PIC	X(2).	
	05	SQLrowcount	PIC	S9(9)	BINARY.
	05	SQLda99	PIC	X(634) .	

`SQLda`

Diagnosebereich für SESAM/SQL.

`SQLda01, SQLda02, ... SQLda99`

für interne Zwecke reserviert.

`SQLerrline`

enthält im Fehlerfall die Zeilennummer der Stelle im Text einer vorbereiteten Anweisung, in der ein Fehler aufgetreten ist. Enthält 0 (Null), wenn die Stelle nicht bestimmt werden konnte oder nicht sinnvoll ist.

`SQLerrcol`

enthält im Fehlerfall die Spaltennummer der Stelle im Text einer vorbereiteten Anweisung, in der ein Fehler aufgetreten ist. Enthält 0 (Null), wenn die Stelle nicht bestimmt werden konnte oder nicht sinnvoll ist.

`SQLerrm`

enthält nach Ausführung einer SQL-Anweisung einen Meldungstext, falls die SQL-Anweisung in SQLSTATE einen anderen Rückgabewert als 00000 liefert. Der Text enthält die Fehlerklasse (W für WARNING oder E für ERROR), die Meldungsnummer SQL *nnnn* und den Meldungstext.

`SQLrowcount`

enthält nach Ausführung der entsprechenden Anweisungen folgende Information:

- nach einer INSERT-Anweisung die Anzahl der eingefügten Sätze
- nach einer UPDATE- oder DELETE-Anweisung mit einer Suchbedingung die Anzahl der Sätze, für die die Suchbedingung erfüllt ist
- nach einer UPDATE- oder DELETE-Anweisung ohne WHERE-Klausel die Anzahl der Sätze in der referenzierten Tabelle
- nach einer MERGE-Anweisung die Summe aus der Anzahl geänderter und Anzahl eingefügter Sätze
- nach einer UNLOAD-Anweisung die Anzahl der ausgebenen Sätze
- nach einer LOAD-Anweisung die Anzahl der neu zugeladenen Sätze
- nach einer EXPORT-Anweisung die Anzahl der in die Export-Datei kopierten Sätze

- nach einer IMPORT-Anweisung die Anzahl der aus der Export-Datei kopierten Sätze

In allen anderen Fällen ist der Inhalt nicht definiert.

Der C-Datentyp SQLda_t

Analog zu SQLda in COBOL gibt es in C den Datentyp SQLda_t. Der Diagnosebereich hat in C folgenden Aufbau:

```
typedef struct {
        short           SQLda01;        /*length*/
        short           SQLda02;        /*reaction_code*/
        short           SQLda03;        /*error_code*/
        short           SQLda04;        /*errm_significant*/
        short           SQLerrline;     /*sqlrow*/
        short           SQLerrcol;      /*sqlcolumn*/
        short           SQLda07;        /*sqlcode*/
        char            SQLda08[5];     /*sqlstate*/
        char            SQLerrm[240];   /*sqlerrm*/
        signed char     SQLda10;        /*SQLda10*/
        long            SQLda21;        /*check field*/
        char            SQLda22[4];     /*tag*/
        unsigned short  SQLda23;        /*internal*/
        char            SQLda24[2];     /*slack*/
        long            SQLrowcount;    /*row_count*/
        char            SQLda99[634];   /*internal area*/ }
SQLda_t;
```

Die einzelnen Parameter entsprechen jeweils den Parametern des COBOL-Datentyps SQLda.

9.1.2 Transaktionseinleitende Anweisungen in CLI-Aufrufen

Die meisten CLI-Aufrufe enthalten transaktionseinleitende SQL-Anweisungen. Mit Ausnahme von SQL_BLOB_CLS_REF werden in allen CLI-Aufrufen SQL-Anweisungen zur Datenmanipulation (abfragen, ändern) aufgerufen.

> **i** In einer SQL-Transaktion dürfen entweder nur SQL-Anweisungen zur Datenmanipulation oder nur SQL-Anweisungen zur Schemadefinition und -verwaltung ausgeführt werden (siehe „Anweisungen innerhalb von Transaktionen“). Aus diesem Grund können CLI-Aufrufe in einer SQL-Transaktion nicht erfolgreich durchgeführt werden, wenn auch SQL-Anweisungen zur Schemadefinition und -verwaltung in dieser Transaktion aufgerufen werden.

Isolationslevel

Mit dem Isolationslevel kann die Parallelität der Transaktionen beeinflusst werden. Die einzelnen Level und die möglichen Phänomene bei konkurrierenden Transaktionen sind im Abschnitt „SET TRANSACTION - Transaktionseigenschaften festlegen“ beschrieben. Für die Bearbeitung von BLOB-Objekten mit CLI-Funktionen ergeben sich folgende Auswirkungen:

- Wird ein BLOB-Wert in einer Transaktion mit dem Isolationslevel SERIALIZABLE oder REPEATABLE READ gelesen, so werden Änderungen durch konkurrierende Transaktionen so lange verzögert, bis das Lesen des Wertes abgeschlossen ist. Änderungen konkurrierender Transaktionen sind also entweder vollständig sichtbar oder unsichtbar. Das Phänomen phantom kann hier nicht auftreten, da REF-Werte nicht wieder verwendet werden.
- Wird ein BLOB-Wert in einer Transaktion mit dem Isolationslevel READ COMMITTED oder READ UNCOMMITTED gelesen, so kann er innerhalb der Transaktion durch konkurrierende Transaktionen geändert werden. Es ist dann möglich, dass teilweise alte und teilweise neue Werte gelesen werden. Es könnte sogar vorkommen, dass während des Lesens zweier Stücke eines BLOB-Werts das BLOB-Objekt gelöscht wird und danach durch ein anderes mit der gleichen Objektnummer ersetzt wird. Am Attribut UPDATED können Sie erkennen, wann das Objekt zuletzt geändert wurde und somit sicherstellen, dass es sich noch um dasselbe handelt.

Konsistenz bei Updates

Der BLOB-Wert eines BLOB-Objekts wird in mehreren Zeilen einer BLOB-Tabelle gespeichert. Zum Ersetzen eines existierenden BLOB-Werts ist deshalb im Allgemeinen mehr als eine DML-Anweisung notwendig. Das Ändern eines BLOB-Objekts ist deshalb keine atomare Operation.

Aus diesem Grund kann es vorkommen, dass das Ändern nur teilweise erfolgreich ist. Beispielsweise kann beim Schreiben des neuen Werts eines BLOB-Objekts der erste Aufruf von BLOB_VAL_STOW (siehe "SQL_BLOB_VAL_STOW - SQLbvst") erfolgreich sein, der zweite aber nicht. In solchen Fällen sollten Sie mit ROLLBACK alle Änderungen zurücksetzen.

Das Ändern der Attribute von BLOB-Objekten und das Löschen von BLOB-Objekten sind jedoch atomare Operationen. Schlagen sie fehl, wird der ursprüngliche Zustand hergestellt.

9.2 Aufrufe des SESAM-CLI

- Übersicht
- Alphabetischer Nachschlageteil
  - SQL_BLOB_CLS_ISBTAB - SQLbcis
  - SQL_BLOB_CLS_REF - SQLbcre
  - SQL_BLOB_OBJ_CLONE - SQLbocl
  - SQL_BLOB_OBJ_CREATE - SQLbocr
  - SQL_BLOB_OBJ_CREAT2 - SQLboc2
  - SQL_BLOB_OBJ_DROP - SQLbodr
  - SQL_BLOB_TAG_GET - SQLbtge
  - SQL_BLOB_TAG_PUT - SQLbtpu
  - SQL_BLOB_VAL_CLOSE - SQLbvcl
  - SQL_BLOB_VAL_FETCH - SQLbvfe
  - SQL_BLOB_VAL_GET - SQLbvge
  - SQL_BLOB_VAL_LEN - SQLbvle
  - SQL_BLOB_VAL_OPEN - SQLbvop
  - SQL_BLOB_VAL_PUT - SQLbvpu
  - SQL_BLOB_VAL_STOW - SQLbvst
  - SQL_DIAG_SEQ_GET - SQLdsg

9.2.1 Übersicht

Folgende Aufrufe des SESAM-CLI stehen dem Anwender zur Verfügung:

Operationen mit Klassen von BLOB-Objekten

CLI-Aufruf	Kurzform	Funktion
SQL_BLOB_CLS_REF	SQLbcre	REF-Wert der Klasse bilden und ausgeben
SQL_BLOB_CLS_ISBTAB	SQLbcis	Prüfung, ob BLOB-Tabelle vorliegt

Tabelle 56: CLI-Aufrufe für Operationen mit Klassen von BLOB-Objekten

BLOB-Objekte erzeugen und löschen

CLI-Aufruf	Kurzform	Funktion
SQL_BLOB_OBJ_CLONE	SQLbocl	Objekt klonen
SQL_BLOB_OBJ_CREATE	SQLbocr	Objekt erzeugen (Objektnummer sequenziell)
SQL_BLOB_OBJ_CREAT2	SQLboc2	Objekt erzeugen (Objektnummer bereichspezifisch)
SQL_BLOB_OBJ_DROP	SQLbodr	Objekt löschen

Tabelle 57: CLI-Aufrufe für BLOB-Objekte

Attribut eines BLOB-Objekts lesen und setzen

CLI-Aufruf	Kurzform	Funktion
SQL_BLOB_TAG_GET	SQLbtge	Attributwert lesen
SQL_BLOB_TAG_PUT	SQLbtpu	Attributwert setzen

Tabelle 58: CLI-Aufrufe für Attribute von BLOB-Objekten

BLOB-Werte lesen und schreiben

CLI-Aufruf	Kurzform	Funktion
SQL_BLOB_VAL_GET	SQLbvge	BLOB-Wert lesen
SQL_BLOB_VAL_PUT	SQLbvpu	BLOB-Wert setzen
SQL_BLOB_VAL_LEN	SQLbvle	Länge des BLOB-Werts ausgeben

Tabelle 59: CLI-Aufrufe für BLOB-Werte

Sequenzielle Bearbeitung von BLOB-Werten

CLI-Aufruf	Kurzform	Funktion
SQL_BLOB_VAL_OPEN	SQLbvop	Access-Handle öffnen
SQL_BLOB_VAL_CLOSE	SQLbvcl	Access-Handle schließen
SQL_BLOB_VAL_FETCH	SQLbvfe	Sequenzielles Lesen eines BLOB-Werts

SQL_BLOB_VAL_STOW	SQLbvst	Sequenzielles Setzen eines BLOB-Werts

Tabelle 60: CLI-Aufruf für einzelne Sequenzen der BLOB-Werte

Attributwerte bei dynamischen INSERT-Anweisungen bestimmen

CLI-Aufruf	Kurzform	Funktion
SQL_DIAG_SEQ_GET	SQLdsg	Bereitstellung der Funktionalität des RETURN INTO aus statischen INSERT-Anweisungen für dynamische INSERT-Anweisungen

Tabelle 61: CLI-Aufruf zur Bestimmung von Attributwerten bei dynamischen INSERT-Anweisungen

Beispiel

Ein Beispielprogramm zur Bearbeitung von BLOB-Werten mit SESAM-CLI finden Sie in der Bibliothek SIPANY.SESAM-SQL.090.MAN-DB. Bei diesem Programm handelt es sich um ein ESQL-COBOL-Programm, aus dem C-Funktionen zur Ausführung der CLI-Aufrufe aufgerufen werden.

Im Folgenden sind die Schritte skizziert, die notwendig sind, um ein BLOB-Objekt zu erzeugen.

1. Der REF-Wert *ref_value* wird ausgegeben für die Klasse der BLOB-Objekte, zu der das neue BLOB-Objekt gehören soll. Das BLOB-Objekt soll in der Tabelle *table* im Schema *schema* liegen.

   ```
   SQL_BLOB_CLS_REF(table, schema, ref_value, &SQLDA)
   ```

2. Das BLOB-Objekt wird erzeugt, wobei der REF-Wert *ref_value* der Klasse und der Name der Datenbank *catalog* eingegeben werden. Ausgegeben wird der REF-Wert *ref_value* des neuen BLOB-Objekts.

   ```
   SQL_BLOB_OBJ_CREATE(ref_value, catalog, &SQLDA)
   ```

3. Ein Access-Handle wird zum Schreiben geöffnet mit ForWriteAccess=1. Es werden der REF-Wert *ref_value* des BLOB-Objekts und der Datenbankname *catalog* angegeben. Das Access-Handle wird im Folgenden durch den zurückgelieferten Wert *access_handle* identifiziert.

   ```
   SQL_BLOB_VAL_OPEN (ref_value, catalog,
                      (long int const *)&ForWriteAccess,
                      access_handle, &SQLDA)
   ```

4. Innerhalb des Access-Handle wird der BLOB-Wert sequenziell gesetzt. Zur Identifizierung des Access-Handle wird *access_handle* angegeben. Dieser Schritt wird so oft wiederholt, bis der gesamte BLOB-Wert aus dem Puffer gelesen wurde.

   ```
   SQL_BLOB_VAL_STOW  (access_handle, input_buffer,
                      (long int const *)&n, &SQLDA)
   ```

5. Der Access-Handle wird geschlossen. Zur Identifizierung des Access-Handle wird *access_handle* angegeben.

   ```
   SQL_BLOB_VAL_CLOSE (access_handle, &SQLDA)
   ```

9.2.2 Alphabetischer Nachschlageteil

In diesem Abschnitt sind die CLI-Aufrufe in einem einheitlichen Stil beschrieben. Die Aufrufe sind alphabetisch angeordnet. Pro Aufruf gibt es einen Eintrag, der den Namen des Aufrufs und seine Kurzform als Kopfzeile hat.

Jeder Eintrag besteht aus mehreren Abschnitten:

Name des Aufrufs - Kurzform

Nach der Überschrift wird die Wirkungsweise des Aufrufs beschrieben.

Dieser Abschnitt beschreibt auch die notwendigen Zugriffsrechte, die für die erfolgreiche Ausführung des Aufrufs erfüllt sein müssen.

```
Funktionsdeklaration in C
Funktionsdeklaration in COBOL

parameter
   Erklärungen zum Parameter.
```

Die Parameter sind in der Reihenfolge beschrieben, in der sie in der Funktionsdeklaration vorkommen.

9.2.2.1 SQL_BLOB_CLS_ISBTAB - SQLbcis

SQL_BLOB_CLS_ISBTAB prüft, ob eine Basistabelle eine BLOB-Tabelle ist. Es wird bei Eingabe des Datenbank-, des Tabellen- und des Schemanamens der Wert 1 bzw. 0 ausgegeben. Falls 1 ausgegeben wird, liegt eine BLOB-Tabelle vor. Liegen Syntaxfehler oder keine BLOB-Tabelle vor, so wird der Wert 0 zurückgeliefert.

Dieser CLI-Aufruf erfordert das SELECT-Privileg für die BLOB-Tabelle.

CLI-Deklaration in C:

```
void SQL_BLOB_CLS_ISBTAB( char const *TableName
          ,char const *SchemaName
          ,char const *CatalogId
          ,long int *IsBlobTable
          ,struct SQLda_t *SQLda);
```

CLI-Deklaration in COBOL:

```
IDENTIFICATION DIVISION.
PROGRAM-ID. SQLbcis IS PROTOTYPE.
DATA DIVISION.
LINKAGE SECTION.
   01 TableName PIC X(31).
   01 SchemaName PIC X(31).
   01 CatalogId PIC X(31).
   01 IsBLOBtable PIC S9(9)COMP.
   COPY SQLCA.          *> for group item SQLda.
PROCEDURE DIVISION USING TableName, SchemaName, CatalogId, IsBLOBtable,
          SQLda.
END PROGRAM SQLbcis.
```

`TableName`

Name einer Basistabelle. `TableName` muss der einfache Tabellenname ohne Datenbank und Schemanamen (siehe Abschnitt „Einfache Namen“) sein. Groß- und Kleinschreibung muss beachtet werden. Der Name muss gegebenenfalls mit Leerzeichen auf 31 Zeichen verlängert oder mit einem Null-Byte abgeschlossen werden.

`SchemaName`

Name des Schemas, in dem die Basistabelle liegt. `SchemaName` muss der einfache Schemaname ohne Datenbankname (siehe Abschnitt „Einfache Namen“) sein. Groß- und Kleinschreibung muss beachtet werden. Der Name muss gegebenenfalls mit Leerzeichen auf 31 Zeichen verlängert oder mit einem Null-Byte abgeschlossen werden.

`CatalogId`

Name der Datenbank, in der die Tabelle liegt. `CatalogId` ist ein einfacher Name (siehe Abschnitt „Einfache Namen“). Der Name muss gegebenenfalls mit Leerzeichen auf 31 Zeichen verlängert oder mit einem Null-Byte abgeschlossen werden. Soll der Name der voreingestellten Datenbank verwendet werden, so geben Sie statt des Datenbanknamens ein Null-Byte, ein oder mehrere Leerzeichen an.

`IsBLOBtable`

Boolscher Wert. Falls 1 ausgegeben wird, liegt eine BLOB-Tabelle vor. Liegen Syntaxfehler oder keine BLOB-Tabelle vor, so wird der Wert 0 zurückgeliefert.

`SQLda`

Diagnosebereich.

9.2.2.2 SQL_BLOB_CLS_REF - SQLbcre

SQL_BLOB_CLS_REF gibt den REF-Wert der Klasse der BLOB-Objekte einer BLOB-Tabelle aus. Eingabe ist der Tabellen- und Schemaname.

Dieser CLI-Aufruf erfordert keine Privilegien.

CLI-Deklaration in C:

```
void SQL_BLOB_CLS_REF( char const *BlobTableName
          ,char const *BlobSchemaName
          ,char *REFvalue
          ,struct SQLda_t *SQLda);
```

CLI-Deklaration in COBOL:

```
IDENTIFICATION DIVISION.
PROGRAM-ID. SQLbcre IS PROTOTYPE.
DATA DIVISION.
LINKAGE SECTION.
   01 BlobTableName PIC X(31).
   01 BlobSchemaName PIC X(31).
   01 REFvalue PIC X(237).
   COPY SQLCA.       *> for group item SQLda.
PROCEDURE DIVISION USING BlobTableName, BlobSchemaName, REFvalue, SQLda.
END PROGRAM SQLbcre.
```

`BlobTableName`

Name der BLOB-Tabelle. `BlobTableName` muss der einfache Tabellenname ohne Datenbank- und Schemaname (siehe Abschnitt „Einfache Namen“) sein. Groß- und Kleinschreibung muss beachtet werden. Der Name muss gegebenenfalls mit Leerzeichen auf 31 Zeichen verlängert oder mit einem Null-Byte abgeschlossen werden.

`BlobSchemaName`

Name des Schemas, in der die BLOB-Tabelle liegt. `BlobSchemaName` muss der einfache Schemaname ohne Datenbankname (siehe Abschnitt „Einfache Namen“) sein. Groß- und Kleinschreibung muss beachtet werden. Der Name muss gegebenenfalls mit Leerzeichen auf 31 Zeichen verlängert oder mit einem Null-Byte abgeschlossen werden.

`REFvalue`

Bei erfolgreicher Ausführung des CLI-Aufrufs wird der REF-Wert der Klasse ausgegeben. Dieser REF-Wert wird, falls er kürzer als 237 Zeichen ist, mit Leerzeichen auf diese Länge ergänzt. Die genaue Struktur des REF-Werts ist auf "Spaltendefinition" beschrieben.

`SQLda`

Diagnosebereich.

9.2.2.3 SQL_BLOB_OBJ_CLONE - SQLbocl

Der SESAM/SQL CLI-Aufruf SQL_BLOB_OBJ_CLONE erzeugt einen Klon eines existierenden BLOB-Objekts in einer anderen Datenbank. Der Klon hat denselben REF-Wert wie das Original und wird in einer BLOB-Tabelle mit demselben Schemanamen und Tabellennamen erzeugt wie das Original. Die Attribute des Klons werden gemäß den Defaultwerten seiner BLOB-Tabelle gesetzt. Sein BLOB-Wert hat die Länge 0.

Den Aufruf SQL_BLOB_OBJ_CLONE können Sie zur Replikation einer BLOB-Tabelle in einer anderen Datenbank verwenden. Bisher war das mit erneutem Ausführen von SQL_BLOB_OBJ_CREATE auf der anderen Datenbank möglich. Dies hatte aber den Nachteil, dass sich unterschiedliche REF-Werte für Original und kopiertes Objekt ergeben, so dass die Referenzen in beiden Datenbanken unterschiedlich sind. Mit dem Aufruf SQL_BLOB_OBJ_CLONE kann man einen Klon mit demselben REF-Wert erzeugen, so dass die Referenzen in beiden Datenbanken gleich bleiben.

Mit SQL_BLOB_OBJ_CLONE können Sie auch ein versehentlich gelöschtes BLOB-Objekt wieder erzeugen.

CLI-Deklaration in C:

```
#define SQL_BLOB_OBJ_CLONE SQLBOCL
extern void SQL_BLOB_OBJ_CLONE( char const *REFvalue /* in */
          ,char const *CatalogId                     /* in */
          ,SQLda_t *sqlda);                          /* out */
```

CLI-Deklaration in Cobol:

```
IDENTIFICATION DIVISION.
PROGRAM-ID. SQLbocl IS PROTOTYPE.
DATA DIVISION.
LINKAGE SECTION.
   01 REFvalue PIC X(237).
   01 CatalogId PIC X(31).
   COPY SQLCA.
PROCEDURE DIVISION USING REFvalue, CatalogId, SQLda.
END PROGRAM SQLbocl.
```

`REFValue`

Der eingegebene `REFValue` muss ein korrekter REF-Wert sein, und er darf kein Klassenobjekt bezeichnen. Die durch `REFvalue` bezeichnete BLOB-Tabelle muss in der angegebenen Datenbank existieren. Sie darf das mit `REFvalue` referenzierte BLOB-Objekt nicht enthalten.Nach erfolgreichem Aufruf referenziert der ausgegebene `REFvalue` ein neues BLOB-Objekt in der Datenbank `CatalogId`. Der BLOB-Wert hat die Länge 0. Die Attribute CREATED und UPDATED enthalten denselben Timestamp, und andere Attribute haben den Default-Wert aus der BLOB-Tabelle.

`CatalogID`

Name der Datenbank mit der neuen BLOB-Tabelle. `CatalogID` ist ein einfacher Name (siehe Abschnitt „Einfache Namen“). Der Name muss gegebenenfalls mit Leerzeichen auf 31 Zeichen verlängert oder mit einem Null-Byte abgeschlossen werden. Soll der Name der voreingestellten Datenbank verwendet werden, so müssen Sie statt des Datenbanknamens ein Null-Byte, ein oder mehrere Leerzeichen angeben.

9.2.2.4 SQL_BLOB_OBJ_CREATE - SQLbocr

SQL_BLOB_OBJ_CREATE erzeugt ein neues BLOB-Objekt und gibt den generierten REF-Wert aus. Eingegeben werden der Datenbankname und der REF-Wert der Klasse, in der das neue BLOB-Objekt enthalten sein soll. Es kann auch der REF-Wert eines bereits existierenden Objekts dieser Klasse angegeben werden. Folgende Werte werden bei der Erzeugung des neuen Objekts in die BLOB-Tabelle eingetragen:

- Das BLOB-Objekt erhält eine innerhalb der Klasse eindeutige Objektnummer. Diese Objektnummer wird sequenziell vergeben.
- Die Attribute des BLOB-Objekts mit den Namen (Tags) UPDATED und CREATED erhalten beide den aktuellen Zeitstempel. Die anderen Attribute werden aus den Voreinstellungen übernommen.

Der BLOB-Wert des neu erzeugten BLOB-Objekts hat die Länge 0.

Das Erzeugen eines BLOB-Objekts erfordert die Privilegien INSERT und SELECT auf BLOB-Tabellen und das UPDATE-Privileg auf die Spalte obj_ref der BLOB-Tabelle.

CLI-Deklaration in C:

```
void SQL_BLOB_OBJ_CREATE( char *REFvalue
          ,char const *CatalogId
          ,struct SQLda_t *SQLda);
```

CLI-Deklaration in COBOL:

```
IDENTIFICATION DIVISION.
PROGRAM-ID. SQLbocr IS PROTOTYPE.
DATA DIVISION.
LINKAGE SECTION.
   01 REFvalue PIC X(237).
   01 CatalogId PIC X(31).
   COPY SQLCA.      *> for group item SQLda.
PROCEDURE DIVISION USING REFvalue, CatalogId, SQLda.
END PROGRAM SQLbocr.
```

`REFvalue`

Der REF-Wert der Klasse oder eines bereits existierenden Objekts derselben Klasse. Die genaue Struktur des REF-Werts ist auf "Spaltendefinition" beschrieben.

`CatalogId`

Name der Datenbank, in der die Tabelle liegt. `CatalogId` ist ein einfacher Name (siehe Abschnitt „Einfache Namen“). Der Name muss gegebenenfalls mit Leerzeichen auf 31 Zeichen verlängert oder mit einem Null-Byte abgeschlossen werden. Soll der Name der voreingestellten Datenbank verwendet werden, so müssen Sie statt des Datenbanknamens ein Null-Byte, ein oder mehrere Leerzeichen angeben.

`SQLda`

Diagnosebereich.

9.2.2.5 SQL_BLOB_OBJ_CREAT2 - SQLboc2

SQL_BLOB_OBJ_CREAT2 erzeugt ein neues BLOB-Objekt und gibt den generierten REF-Wert aus. Eingegeben werden der Datenbankname, der REF-Wert der Klasse, in der das neue BLOB-Objekt enthalten sein soll, sowie ein Intervall für die Objektnummer. Es kann auch der REF-Wert eines bereits existierenden Objekts dieser Klasse angegeben werden. Folgende Werte werden bei der Erzeugung des neuen Objekts in die BLOB-Tabelle eingetragen:

- Das BLOB-Objekt erhält eine innerhalb der Klasse eindeutige Objektnummer. Diese Objektnummer wird innerhalb des angegebenen Intervalls nach einem bestimmten Algorithmus vergeben. Dieser gewährleistet, dass die Objektnummern bei mehreren SQL_BLOB_OBJ_CREAT2-Aufrufen gleichmäßig auf das angegebene Intervall verteilt werden.
- Die Attribute des BLOB-Objekts mit den Namen (Tags) UPDATED und CREATED erhalten beide den aktuellen Zeitstempel. Die anderen Attribute werden aus den Voreinstellungen übernommen.

Der BLOB-Wert des neu erzeugten BLOB-Objekts hat die Länge 0.

Das Erzeugen eines BLOB-Objekts erfordert die Privilegien INSERT und SELECT auf BLOB-Tabellen und das UPDATE-Privileg auf die Spalte obj_ref der BLOB-Tabelle.

CLI-Deklaration in C:

```
void SQL_BLOB_OBJ_CREAT2( char *REFvalue
          ,char const *CatalogId
          ,long int *MinObjectNmbr
          ,long int *MaxObjectNmbr
          ,struct SQLda_t *SQLda);
```

CLI-Deklaration in Cobol:

```
IDENTIFICATION DIVISION.
PROGRAM-ID. SQLboc2 IS PROTOTYPE.
DATA DIVISION.
LINKAGE SECTION.
   01 REFvalue PIC X(237).
   01 CatalogId PIC X(31).
   01 MinObjectNmbr PIC S9(9) COMP.
   01 MaxObjectNmbr PIC S9(9) COMP.
   COPY SQLCA.     *> for group item SQLda.
PROCEDURE DIVISION USING REFvalue, MinObjectNmbr, MaxObjectNmbr, CatalogId,
          SQLda.
END PROGRAM SQLboc2.
```

`REFvalue`

Der REF-Wert der Klasse oder eines bereits existierenden Objekts derselben Klasse. Die genaue Struktur des REF-Werts ist auf "Spaltendefinition" beschrieben.

`CatalogId`

Name der Datenbank, in der die Tabelle liegt. `CatalogId` ist ein einfacher Name (siehe Abschnitt „Einfache Namen“). Der Name muss gegebenenfalls mit Leerzeichen auf 31 Zeichen verlängert oder mit einem Null-Byte abgeschlossen werden. Soll der Name der voreingestellten Datenbank verwendet werden, so müssen Sie statt des Datenbanknamens ein Null-Byte, ein oder mehrere Leerzeichen angeben.

`MinObjectNmbr`

Minimalwert für die Objektnummer (muss >= 1 sein).

`MaxObjectNmbr`

Maximalwert für die Objektnummer (muss größer als oder gleich dem Minimalwert sein).

`SQLda`

Diagnosebereich.

9.2.2.6 SQL_BLOB_OBJ_DROP - SQLbodr

SQL_BLOB_OBJ_DROP löscht ein existierendes BLOB-Objekt bei Eingabe des Datenbanknamens und des REF-Werts. Mit dem BLOB-Objekt werden auch der BLOB-Wert und alle seine Attribute gelöscht. Das Löschen eines BLOB-Objekts wird durch das Löschen von einem oder mehreren Sätzen in der BLOB-Tabelle realisiert. Falls dabei ein Fehler auftritt, wird dieser zurückgemeldet und das BLOB-Objekt bleibt unverändert. (Das Attribut UPDATED kann jedoch verändert sein.) Die Synchronisation von konkurrierenden Transaktionen erfolgt wie in SESAM/SQL üblich.

Das Löschen eines BLOB-Objekts erfordert die Privilegien DELETE und SELECT für die BLOB-Tabelle und das UPDATE-Privileg auf die Spalte slice_val der BLOB-Tabelle.

CLI-Deklaration in C:

```
void SQL_BLOB_OBJ_DROP( char const *REFvalue
          ,char const *CatalogId
          ,struct SQLda_t *SQLda);
```

CLI-Deklaration in COBOL:

```
IDENTIFICATION DIVISION.
PROGRAM-ID. SQLbodr IS PROTOTYPE.
DATA DIVISION.
LINKAGE SECTION.
   01 REFvalue PIC X(237).
   01 CatalogId PIC X(31).
   COPY SQLCA.     *> for group item SQLda.
PROCEDURE DIVISION USING REFvalue, CatalogId, SQLda.
END PROGRAM SQLbodr.
```

`REFvalue`

Der REF-Wert des BLOB-Objekts. Die genaue Struktur des REF-Werts ist auf "Spaltendefinition" beschrieben.

`CatalogId`

Name der Datenbank, in der die Tabelle liegt. `CatalogId` ist ein einfacher Name (siehe Abschnitt „Einfache Namen“). Der Name muss gegebenenfalls mit Leerzeichen auf 31 Zeichen verlängert oder mit einem Null-Byte abgeschlossen werden. Soll der Name der voreingestellten Datenbank verwendet werden, so müssen Sie statt des Datenbanknamens ein Null-Byte, ein oder mehrere Leerzeichen angeben.

`SQLda`

Diagnosebereich.

9.2.2.7 SQL_BLOB_TAG_GET - SQLbtge

SQL_BLOB_TAG_GET gibt einen Attributwert eines existierenden BLOB-Objekts aus. Eingegeben wird der REF-Wert des BLOB-Objekts, der Datenbankname und der Name des Attributs (Tag). Mögliche Tags sind CREATED, UPDATED, MIME und USAGE. Zusätzlich müssen zum Auslesen des Werts der Puffers und dessen Größe zur Verfügung gestellt werden. Hat das BLOB-Objekt kein Attribut mit dem angegebenen Tag, so wird eine Fehlermeldung ausgegeben.

Dieser CLI-Aufruf erfordert das SELECT-Privileg für die BLOB-Tabelle.

CLI-Deklaration in C:

```
void SQL_BLOB_TAG_GET( char const *REFvalue
          ,char const *CatalogId
          ,char const *TagName
          ,char *Buffer
          ,long int const *BufferLength
          ,long int *ValueLength
          ,struct SQLda_t *SQLda);
```

CLI-Deklaration in COBOL:

```
IDENTIFICATION DIVISION.
PROGRAM-ID. SQLbtge IS PROTOTYPE.
DATA DIVISION.
LINKAGE SECTION.
   01 REFvalue PIC X(237).
   01 CatalogId PIC X(31).
   01 TagName PIC X(31).
   01 Buffer.     *> of any length
       02 PIC X(1).
   01 BufferLength PIC S9(9) COMP.
   01 ValueLength PIC S9(9) COMP.
   COPY SQLCA.     *> for group item SQLda.
PROCEDURE DIVISION USING REFvalue, CatalogId, TagName, Buffer, BufferLength,
          ValueLength, SQLda.
END PROGRAM SQLbtge.
```

`REFvalue`

Der REF-Wert des BLOB-Objekts. Die genaue Struktur des REF-Werts ist auf "Spaltendefinition" beschrieben.

`CatalogId`

Name der Datenbank, in der die Tabelle liegt. `CatalogId` ist ein einfacher Name (siehe Abschnitt „Einfache Namen“). Der Name muss gegebenenfalls mit Leerzeichen auf 31 Zeichen verlängert oder mit einem Null-Byte abgeschlossen werden. Soll der Name der voreingestellten Datenbank verwendet werden, so müssen Sie statt des Datenbanknamens ein Null-Byte, ein oder mehrere Leerzeichen angeben.

`TagName`

Name des Attributs (Tag). Der Name muss gegebenenfalls mit Leerzeichen auf 31 Zeichen verlängert oder mit einem Null-Byte abgeschlossen werden. `TagName` darf nicht nur aus Leerzeichen bestehen.

`Buffer`

Puffer, in den der Attributwert geschrieben werden soll.

`BufferLength`

Größe des Puffers in Byte. `BufferLength` muss eine Zahl >= 0 sein. Ist die Puffergröße kleiner als die Länge des Attributwerts nach dem Abschneiden der Leerzeichen am Ende, werden soviele Werte in den Puffer geschrieben, wie dessen Größe erlaubt. Zusätzlich wird in diesem Fall eine Meldung ausgegeben.

`ValueLength`

Bei erfolgreichem Lesen des Attributwerts wird dessen Länge in Byte ausgegeben. Ist die Länge größer als der Wert bei `BufferLength`, dann wurde nur ein Teil des Attributwerts in den Puffer übertragen.

`SQLda`

Diagnosebereich.

9.2.2.8 SQL_BLOB_TAG_PUT - SQLbtpu

SQL_BLOB_TAG_PUT ersetzt einen Attributwert eines existierenden BLOB-Objekts. Die Eingabeparameter sind der REF-Wert, der Datenbankname und der Name des Attributs (Tag). Der neue Attributwert muss in einem Puffer stehen. Dessen Adresse und die Länge des neuen Werts sind weitere Eingabeparameter.

Dieser CLI-Aufruf erfordert das SELECT-Privileg auf die BLOB-Tabelle und das UPDATE-Privileg auf die Spalte slice_val der BLOB-Tabelle.

CLI-Deklaration in C:

```
void SQL_BLOB_TAG_PUT( char const *REFvalue
          ,char const *CatalogId
          ,char const *TagName
          ,char *Buffer
          ,long int const *ValueLength
          ,struct SQLda_t *SQLda);
```

CLI-Deklaration in COBOL:

```
IDENTIFICATION DIVISION.
PROGRAM-ID. SQLbtpu IS PROTOTYPE.
DATA DIVISION.
LINKAGE SECTION.
   01 REFvalue PIC X(237).
   01 CatalogId PIC X(31).
   01 TagName PIC X(31).
   01 Buffer.    *> of any length
       02 PIC X(1).
   01 ValueLength PIC S9(9) COMP.
   COPY SQLCA.     *> for group item SQLda.
PROCEDURE DIVISION USING REFvalue, CatalogId, TagName, Buffer, ValueLength,
          SQLda.
END PROGRAM SQLbtpu.
```

`REFvalue`

Der REF-Wert des BLOB-Objekts. Die genaue Struktur des REF-Werts ist auf "Spaltendefinition" beschrieben.

`CatalogId`

Name der Datenbank, in der die Tabelle liegt. `CatalogId` ist ein einfacher Name (siehe Abschnitt „Einfache Namen“). Der Name muss gegebenenfalls mit Leerzeichen auf 31 Zeichen verlängert oder mit einem Null-Byte abgeschlossen werden. Soll der Name der voreingestellten Datenbank verwendet werden, so müssen Sie statt des Datenbanknamens ein Null-Byte, ein oder mehrere Leerzeichen angeben.

`TagName`

Name der Attribute (Tags). Der Name muss gegebenenfalls mit Leerzeichen auf 31 Zeichen verlängert oder mit einem Null-Byte abgeschlossen werden. `TagName` darf nicht nur aus Leerzeichen bestehen.

`Buffer`

Puffer, in dem der neue Attributwert steht.

`ValueLength`

Länge des neuen Attributwerts. `ValueLength` muss eine Zahl >= 0 sein.

`SQLda`

Diagnosebereich als Ausgabeparameter.

9.2.2.9 SQL_BLOB_VAL_CLOSE - SQLbvcl

SQL_BLOB_VAL_CLOSE beendet die Verwendung eines Access-Handles, das mit SQL_BLOB_VAL_OPEN (siehe "SQL_BLOB_VAL_OPEN - SQLbvop") eröffnet wird.

Mit einem Access-Handle ist es möglich, BLOB-Werte sequenziell zu bearbeiten. Dabei werden die Aufrufe SQL_BLOB_VAL_FETCH (siehe "SQL_BLOB_VAL_FETCH - SQLbvfe") zum sequenziellen Lesen und SQL_BLOB_VAL_STOW (siehe "SQL_BLOB_VAL_STOW - SQLbvst") zum sequenziellen Schreiben angeboten.

Falls Sie versuchen ein Access-Handle zu beenden, das bereits beendet wurde, erhalten Sie eine Fehlermeldung. SQL_BLOB_VAL_CLOSE kann das INSERT-Privileg auf die BLOB-Tabelle erfordern.

CLI-Deklaration in C:

```
void SQL_BLOB_VAL_CLOSE( char *AccessHandle
                        ,struct SQLda_t *SQLda);
```

CLI-Deklaration in COBOL:

```
IDENTIFICATION DIVISION.
PROGRAM-ID. SQLbvcl IS PROTOTYPE.
DATA DIVISION.
   01 AccessHandle PIC X(32).
   COPY SQLCA.     *> for group item SQLda.
LINKAGE SECTION.
PROCEDURE DIVISION USING AccessHandle, SQLda.
END PROGRAM SQLbvcl.
```

`AccessHandle`

Der bei SQL_BLOB_VAL_OPEN gelieferte Wert für das zu beendende Access-Handle muss hier eingegeben werden. Dieser Wert darf vom Aufrufer nicht modifiziert werden.

`SQLda`

Diagnosebereich.

9.2.2.10 SQL_BLOB_VAL_FETCH - SQLbvfe

Mit SQL_BLOB_VAL_FETCH wird ein BLOB-Wert sequenziell gelesen. Im Gegensatz dazu wird bei dem CLI-Aufruf SQL_BLOB_VAL_GET (siehe "SQL_BLOB_VAL_GET - SQLbvge") der gesamte Wert in einem Stück gelesen. SQL_BLOB_VAL_FETCH besitzt daher gegenüber SQL_BLOB_VAL_GET den Vorteil, dass die Größe des Puffers, in den ausgelesen wird, kleiner sein kann als der BLOB-Wert.

Um einen BLOB-Wert mit SQL_BLOB_VAL_FETCH sequenziell zu lesen, benötigen Sie ein Access-Handle zum Lesen. Dieses Access-Handle erzeugen Sie mit SQL_BLOB_VAL_OPEN. Dabei legen Sie mit dem Parameter `ForWriteAccess` dieses Aufrufs fest, dass Sie dieses Access-Handle zum Lesen benötigen (siehe "SQL_BLOB_VAL_OPEN - SQLbvop"f).
SESAM/SQL liefert nach dem Aufruf von SQL_BLOB_VAL_OPEN eine eindeutige Identifikation für das Access-Handle zurück.

Diese Identifikation müssen Sie bei jedem Aufruf von SQL_BLOB_VAL_FETCH angeben. Zusätzlich übergeben Sie noch den Puffer, in den das gelesene Stück übertragen werden soll, und die Länge dieses Puffers.

Beim ersten Aufruf von SQL_BLOB_VAL_FETCH innerhalb eines Access-Handles wird der Puffer mit dem ersten Teil des BLOB-Werts gefüllt. Beim nächsten Aufruf mit demselben Access-Handle wird der Puffer mit dem nächsten Teil des BLOB-Werts gefüllt, usw.. Wurde der gesamte BLOB-Wert gelesen, wird dies mit einer Meldung (SQLSTATE 02000) angezeigt.

Nach dem vollständigen Lesen des BLOB-Werts ist es nicht mehr möglich, mit dem verwendeten Access-Handle SQL_BLOB_VAL_FETCH aufzurufen. Das Access-Handle muss mit SQL_BLOB_VAL_CLOSE geschlossen werden (siehe "SQL_BLOB_VAL_CLOSE - SQLbvcl").

Die gesamte Folge von Operationen (SQL_BLOB_VAL_OPEN, wiederholtes SQL_BLOB_VAL_FETCH und SQL_BLOB_VAL_CLOSE) muss innerhalb einer Transaktion stattfinden.

Dieser CLI-Aufruf erfordert keine Privilegien.

CLI-Deklaration in C:

```
void SQL_BLOB_VAL_FETCH( char *AccessHandle
          ,char *Buffer
          ,long int const *BufferLength
          ,long int *ValueLength
          ,struct SQLda_t * SQLda);
```

CLI-Deklaration in Cobol:

```
IDENTIFICATION DIVISION.
PROGRAM-ID. SQLbvfe IS PROTOTYPE.
DATA DIVISION.
LINKAGE SECTION.
   01 AccessHandle PIC X(32).
   01 Buffer.     *> of any length
       02 PIC X(1).
   01 BufferLength PIC S9(9) COMP.
   01 ValueLength PIC S9(9) COMP.
   COPY SQLCA.     *> for group item SQLda.
PROCEDURE DIVISION USING AccessHandle, Buffer, BufferLength, ValueLength,
          SQLda.
END PROGRAM SQLbvfe.
```

`AccessHandle`

Der bei SQL_BLOB_VAL_OPEN gelieferte Wert für das Access-Handle muss hier eingegeben werden. Dieser Wert darf vom Aufrufer nicht modifiziert werden.

`Buffer`

Puffer, in den der Wert übertragen werden soll.

`BufferLength`

Größe des Puffers in Byte. `BufferLength` muss eine Zahl >= 0 sein.

`ValueLength`

Größe des in den Puffer geschriebenen Teils des BLOB-Werts. Ist der Wert von `ValueLength` kleiner als der Wert von `BufferLength`, so ist der BLOB-Wert komplett gelesen worden.

`SQLda`

Diagnosebereich.

9.2.2.11 SQL_BLOB_VAL_GET - SQLbvge

SQL_BLOB_VAL_GET liest einen BLOB-Wert. Eingegeben werden der REF-Wert, der Datenbankname, der Puffer, in den der gelesene Wert übertragen wird und die Größe dieses Puffers.

Für die Ausgabe des BLOB-Werts wird das SELECT-Privileg auf die BLOB-Tabelle benötigt.

CLI-Deklaration in C:

```
void SQL_BLOB_VAL_GET( char const *REFvalue
          ,char const *CatalogId
          ,char *Buffer
          ,long int const *BufferLength
          ,long int *ValueLength
          ,struct SQLda_t *SQLda);
```

CLI-Deklaration in COBOL:

```
IDENTIFICATION DIVISION.
PROGRAM-ID. SQLbvge IS PROTOTYPE.
DATA DIVISION.
LINKAGE SECTION.
   01 REFvalue PIC X(237).
   01 CatalogId PIC X(31).
   01 Buffer.     *> of any length
       02 PIC X(1).
   01 BufferLength PIC S9(9) COMP.
   01 ValueLength PIC S9(9) COMP.
   COPY SQLCA.     *> for group item SQLda.
PROCEDURE DIVISION USING REFvalue, CatalogId, Buffer, BufferLength,
          ValueLength, SQLda.
END PROGRAM SQLbvge.
```

`REFvalue`

Der REF-Wert des BLOB-Objekts. Die genaue Struktur des REF-Werts ist auf "Spaltendefinition" beschrieben.

`CatalogId`

Name der Datenbank, in der die Tabelle liegt. `CatalogId` ist ein einfacher Name (siehe Abschnitt „Einfache Namen“). Der Name muss gegebenenfalls mit Leerzeichen auf 31 Zeichen verlängert oder mit einem Null-Byte abgeschlossen werden. Soll der Name der voreingestellten Datenbank verwendet werden, so müssen Sie statt des Datenbanknamens ein Null-Byte, ein oder mehr Leerzeichen angeben.

`Buffer`

Puffer, in den der Wert übertragen wird.

`BufferLength`

Größe des Puffers in Byte. `BufferLength` muss eine Zahl >= 0 sein.

`ValueLength`

Länge des BLOB-Werts. Ist der Wert von `ValueLength` größer als der Wert von `BufferLength`, so wurden nur die ersten Bytes des BLOB-Werts (bis zur Größe `BufferLength`) in den Puffer übertragen. Anderenfalls steht am Ende des Aufrufs der ganze BLOB-Wert in den ersten Bytes des Puffers.

`SQLda`

Diagnosebereich.

9.2.2.12 SQL_BLOB_VAL_LEN - SQLbvle

SQL_BLOB_VAL_LEN bestimmt bei Eingabe des REF-Werts und des Datenbanknamens die Länge eines BLOB-Werts und gibt diese aus.

Für den Aufruf SQL_BLOB_VAL_LEN wird das SELECT-Privileg auf die BLOB-Tabelle benötigt.

CLI-Deklaration in C:

```
void SQL_BLOB_VAL_LEN( char const *REFvalue
          ,char const *CatalogId
          ,long int *ValueLength
          ,struct SQLda_t *SQLda);
```

CLI-Deklaration in COBOL:

```
IDENTIFICATION DIVISION.
PROGRAM-ID. SQLbvle IS PROTOTYPE.
DATA DIVISION.
LINKAGE SECTION.
   01 REFvalue PIC X(237).
   01 CatalogId PIC X(31).
   01 ValueLength PIC S9(9) COMP.
   COPY SQLCA.     *> for group item SQLda.
PROCEDURE DIVISION USING REFvalue, CatalogId, ValueLength, SQLda.
END PROGRAM SQLbvle.
```

`REFvalue`

Der REF-Wert des BLOB-Objekts. Die genaue Struktur des REF-Werts ist auf "Spaltendefinition" beschrieben.

`CatalogId`

Name der Datenbank, in der die Tabelle liegt. `CatalogId` ist ein einfacher Name (siehe Abschnitt „Einfache Namen“). Der Name muss gegebenenfalls mit Leerzeichen auf 31 Zeichen verlängert oder mit einem Null-Byte abgeschlossen werden. Soll der Name der voreingestellten Datenbank verwendet werden, so müssen Sie statt des Datenbanknamens ein Null-Byte, ein oder mehr Leerzeichen angeben.

`ValueLength`

Länge des BLOB-Werts.

`SQLda`

Diagnosebereich.

9.2.2.13 SQL_BLOB_VAL_OPEN - SQLbvop

Der CLI-Aufruf SQL_BLOB_VAL_OPEN öffnet ein Access-Handle. Ein Access-Handle benötigen Sie zur sequenziellen Bearbeitung von BLOB-Werten. In SESAM/SQL können Sie BLOB-Werte mit SQL_BLOB_VAL_FETCH (siehe "SQL_BLOB_VAL_FETCH - SQLbvfe") sequenziell lesen und mit SQL_BLOB_VAL_STOW (siehe "SQL_BLOB_VAL_STOW - SQLbvst") sequenziell schreiben.

Ist das sequenzielle Bearbeiten mit dem Access-Handle abgeschlossen, beenden Sie dieses mit dem Aufruf SQL_BLOB_VAL_CLOSE (siehe "SQL_BLOB_VAL_CLOSE - SQLbvcl"). Das Access-Handle muss innerhalb derselben Transaktion geöffnet und wieder geschlossen werden.

Durch wiederholtes Anwenden der oben genannten CLI-Aufrufe zum Schreiben oder Lesen, wird der BLOB-Wert stückweise bearbeitet. Damit dies korrekt erfolgen kann, wird ein Access-Handle benötigt, das die folgenden internen Informationen verwaltet:

- Informationen darüber welcher BLOB-Wert angesprochen wird und
- über den Fortschritt des Lesens oder des Schreibens (welches Teilstück beispielsweise als nächstes gelesen bzw. geschrieben werden muss).

Als Parameter geben Sie den REF-Wert, den Datenbanknamen und `ForWriteAccess` an. Mit `ForWriteAccess` legen Sie fest, ob in dem Access-Handle gelesen oder geschrieben wird.
Das Access-Handle erhält von SESAM/SQL eine eindeutige Identifikation, die Sie beim jeweiligen Aufruf von SQL_BLOB_VAL_FETCH und SQL_BLOB_VAL_STOW angeben müssen.

Sie können bis zu 10 Access-Handle gleichzeitig öffnen, d.h. 10 sequenzielle Bearbeitungen von BLOB-Objekten parallel durchführen. Falls Sie eine 11. sequenzielle Bearbeitung parallel durchführen wollen, bekommen Sie eine entsprechende Meldung.

Versäumen Sie es, am Ende einer sequenziellen Bearbeitung SQL_BLOB_VAL_CLOSE abzusetzen, bleibt das Access-Handle als „belegt“ gekennzeichnet. Eine optimale Ausnutzung der 10 gleichzeitig zu öffnenden Access-Handle ist dann nicht mehr möglich.

Öffnen Sie ein Access-Handle zum Schreiben und hat das zu bearbeitende BLOB-Objekt bereits einen BLOB-Wert, wird dieser alte BLOB-Wert durch den Aufruf von SQL_BLOB_VAL_OPEN gelöscht, d.h. er hat die Länge 0. Das BLOB-Objekt selbst bleibt erhalten.

Eine parallele sequenzielle Bearbeitung des gleichen BLOB-Objektes durch mehrere schreibende Sequenzen sollte vermieden werden, da die schreibenden Sequenzen gegenseitig den BLOB-Wert beeinflussen können.

SQL_BLOB_VAL_OPEN zum Lesen erfordert das SELECT-Privileg auf die BLOB-Tabelle.
SQL_BLOB_VAL_OPEN zum Schreiben erfordert die Privilegien SELECT und DELETE auf die BLOB-Tabelle und das UPDATE-Privileg auf die Spalte slice_val der BLOB-Tabelle.

CLI-Deklaration in C:

```
void SQL_BLOB_VAL_OPEN( char const *REFvalue
          ,char const *CatalogId
          ,long int *ForWriteAccess
          ,char *AccesHandle
          ,struct SQLda_t *SQLda);
```

CLI-Deklaration in COBOL:

```
IDENTIFICATION DIVISION.
PROGRAM-ID. SQLbvop IS PROTOTYPE.
```

```
DATA DIVISION.
LINKAGE SECTION.
   01 REFvalue PIC X(237).
   01 CatalogId PIC X(31).
   01 ForWriteAccess PIC S9(9) COMP.
   01 AccesHandle PIC X(32).
   COPY SQLCA.     *> for group item SQLda.
PROCEDURE DIVISION USING REFvalue, CatalogId, ForWriteAccess, AccesHandle,
          SQLda.
END PROGRAM SQLbvop.
```

`REFvalue`

Der REF-Wert des BLOB-Objekts. Die genaue Struktur des REF-Werts ist auf "Spaltendefinition" beschrieben.

`CatalogId`

Name der Datenbank, in der die Tabelle liegt. `CatalogId` ist ein einfacher Name (siehe Abschnitt „Einfache Namen“). Der Name muss gegebenenfalls mit Leerzeichen auf 31 Zeichen verlängert oder mit einem Null-Byte abgeschlossen werden. Soll der Name der voreingestellten Datenbank verwendet werden, so müssen Sie statt des Datenbanknamens ein Null-Byte, ein oder mehrere Leerzeichen angeben.

`ForWriteAccess`

Als Eingabe sind hier die Werte 1 (=TRUE) oder 0 (=FALSE) möglich. Die Werte haben folgende Bedeutung:

0: Es wird ein Access-Handle zum Lesen (mit SQL_BLOB_VAL_FETCH) geliefert.
.

1: Es wird ein Access-Handle zum Schreiben (mit SQL_BLOB_VAL_STOW) geliefert.

`AccessHandle`

Der ausgebene Wert für `AccessHandle` dient zur Identifikation des Access-Handles. Sie dürfen diesen Wert nicht modifizieren, da er in allen nachfolgenden Operationen bis zum abschließenden Aufruf SQL_BLOB_VAL_CLOSE verwendet wird.

`SQLda`

Diagnosebereich.

9.2.2.14 SQL_BLOB_VAL_PUT - SQLbvpu

SQL_BLOB_VAL_PUT ersetzt den BLOB-Wert eines BLOB-Objekts durch einen neuen BLOB-Wert aus einem Puffer. Dazu müssen der REF-Wert des BLOB-Objekts, der Datenbankname, der Puffer, in dem der neue Wert steht, und die Länge des neuen Werts angegeben werden.

Das Ersetzen eines BLOB-Werts erfordert die Privilegien INSERT, SELECT und DELETE auf die BLOB-Tabelle. Zusätzlich benötigen Sie noch das UPDATE-Privileg auf die Spalte slice_val der BLOB-Tabelle.

CLI-Deklaration in C:

```
void SQL_BLOB_VAL_PUT( char const *REFvalue
          ,char const *CatalogId
          ,char *Buffer
          ,long int *ValueLength
          ,struct SQLda_t *SQLda);
```

CLI-Deklaration in COBOL:

```
IDENTIFICATION DIVISION.
PROGRAM-ID. SQLbvpu IS PROTOTYPE.
DATA DIVISION.
LINKAGE SECTION.
   01 REFvalue PIC X(237).
   01 CatalogId PIC X(31).
   01 Buffer.     *> of any length
       02 PIC X(1).
   01 ValueLength PIC S9(9) COMP.
   COPY SQLCA.     *> for group item SQLda.
PROCEDURE DIVISION USING REFvalue, CatalogId, Buffer, ValueLength, SQLda.
END PROGRAM SQLbvge.
```

`REFvalue`

Der REF-Wert des BLOB-Objekts. Die genaue Struktur des REF-Werts ist auf "Spaltendefinition" beschrieben.

`CatalogId`

Name der Datenbank, in der die Tabelle liegt. `CatalogId` ist ein einfacher Name (siehe Abschnitt „Einfache Namen“). Der Name muss gegebenenfalls mit Leerzeichen auf 31 Zeichen verlängert oder mit einem Null-Byte abgeschlossen werden. Soll der Name der voreingestellten Datenbank verwendet werden, so müssen Sie statt des Datenbanknamens ein Null-Byte, ein oder mehr Leerzeichen angeben.

`Buffer`

Puffer, der den neuen BLOB-Wert enthält.

`ValueLength`

Länge des BLOB-Werts. `ValueLength` muss eine Zahl >= 0 sein.

`SQLda`

Diagnosebereich.

9.2.2.15 SQL_BLOB_VAL_STOW - SQLbvst

Mit SQL_BLOB_VAL_STOW wird ein neuer BLOB-Wert sequenziell in ein BLOB-Objekt geschrieben. Im Gegensatz dazu wird bei dem CLI-Aufruf SQL_BLOB_VAL_PUT (siehe "SQL_BLOB_VAL_PUT - SQLbvpu") der gesamte Wert in einem Stück geschrieben. SQL_BLOB_VAL_STOW besitzt daher gegenüber SQL_BLOB_VAL_PUT den Vorteil, dass der Puffer nicht die Größe des gesamten neuen BLOB-Werts haben muss. Der neue BLOB-Wert kann in kleinen Teilstücken übergeben werden.

Um einen BLOB-Wert mit SQL_BLOB_VAL_STOW sequenziell zu schreiben, benötigen Sie ein Access-Handle zum Schreiben. Dieses Access-Handle erzeugen Sie mit SQL_BLOB_VAL_OPEN. Dabei legen Sie mit dem Parameter `ForWriteAccess` dieses Aufrufs fest, dass Sie dieses Access-Handle zum Schreiben benötigen (siehe "SQL_BLOB_VAL_OPEN - SQLbvop"f). SESAM/SQL liefert nach dem Aufruf von SQL_BLOB_VAL_OPEN eine eindeutige Identifikation für das Access-Handle zurück.

Diese Identifikation müssen Sie bei jedem Aufruf von SQL_BLOB_VAL_STOW angeben. Zusätzlich übergeben Sie noch den Puffer, in dem das neue Stück des BLOB-Werts steht, und die Länge dieses Puffers.

Nachdem durch mehrmaliges Aufrufen von SQL_BLOB_VAL_STOW alle Stücke des BLOB-Werts geschrieben wurden, muss das verwendete Access-Handle mit SQL_BLOB_VAL_CLOSE geschlossen werden (siehe "SQL_BLOB_VAL_CLOSE - SQLbvcl"). Erst dabei wird das letzte Stück des neuen BLOB-Werts in die BLOB-Tabelle eingefügt.

Die gesamte Folge von Operationen (SQL_BLOB_VAL_OPEN, wiederholtes SQL_BLOB_VAL_STOW und SQL_BLOB_VAL_CLOSE) muss innerhalb einer Transaktion stattfinden.

Dieser CLI-Aufruf erfordert das INSERT-Privileg für die BLOB-Tabelle.

CLI-Deklaration in C:

```
void SQL_BLOB_VAL_STOW( char *AccessHandle
          ,char *Buffer
          ,long int const *ValueLength
          ,struct SQLda_t *SQLda);
```

CLI-Deklaration in Cobol:

```
IDENTIFICATION DIVISION.
PROGRAM-ID. SQLbvst IS PROTOTYPE.
DATA DIVISION.
LINKAGE SECTION.
   01 AccessHandle PIC X(32).
   01 Buffer.    *> of any length
       02 PIC X(1).
   01 ValueLength PIC S9(9) COMP.
   COPY SQLCA.   *> for group item SQLda.
PROCEDURE DIVISION USING AccessHandle, Buffer, ValueLength, SQLda.
END PROGRAM SQLbvst.
```

`AccessHandle`

Der bei SQL_BLOB_VAL_OPEN gelieferte Wert für das Access-Handle muss hier eingegeben werden. Dieser Wert darf vom Aufrufer nicht modifiziert werden.

`Buffer`

Puffer, der den neuen Wert enthält.

`ValueLength`

Länge des Werts. `ValueLength` muss eine Zahl >= 0 sein.

`SQLda`

Diagnosebereich.

9.2.2.16 SQL_DIAG_SEQ_GET - SQLdsg

Mit SQL_DIAG_SEQ_GET kann die von statischen INSERT-Anweisungen bekannte Funktionalität RETURN INTO für dynamische INSERT-Anweisungen nachgebildet werden.

SQL_DIAG SEQ_GET liefert den Wert, der von SESAM/SQL in einer INSERT-Anweisung für einen '*' in der VALUES-Klausel bzw. bei der Angabe von COUNT INTO vergeben wurde..

Die Funktion kann auch für statische INSERT-Anweisungen genutzt werden.

Sie wird als LLM 'SQLDSG' in der Bibliothek SIPLIB.SESAM-SQL.090.CLI zur Verfügung gestellt. Wenn sie in einem Anwenderprogramm genutzt werden soll, muss entweder das LLM explizit hinzugebunden werden oder die Bibliothek zur Laufzeit als BLSLIBxx angegeben werden. Die Schnittstellen dieser Funktion werden in der Bibliothek als S-Elemente *sqldsg.h* (für C) und *sqldsg* (für COBOL) bereitgestellt.

CLI-Deklaration in C:

```
extern void SQL_DIAG_SEQ_GET( struct SQLda_t *SQLda
          ,char *SequenceValue
          ,signed short *RC);
```

CLI-Deklaration in COBOL:

```
IDENTIFICATION DIVISION.
PROGRAM-ID. SQLdsg IS PROTOTYPE.
DATA DIVISION.
LINKAGE SECTION.
   COPY SQLCA .     *> for group item SQLda
   01 SequenceValue PIC X(34).
   01 RC PIC S9(4) BINARY.
PROCEDURE DIVISION USING SQLda, SQL_SequenceValue, RC.
END PROGRAM SQLdsg.
```

`SQLda`

SQL-Diagnosebereich, mit dem die INSERT-Anweisung ausgeführt wurde.Falls die SQLda eines COBOL /ESQL-Programms aus einem C-Programm heraus referiert werden soll, so muss sie in COBOL als EXTERNAL angegeben werden.

`SequenceValue`

Speicherbereich, in den das Ergebnis übertragen werden soll. Der Bereich muss mindestes 34 Byte groß sein.

Der Wert hat bei erfolgreicher Übertragung (RC=0) 34 Zeichen und folgendes Format, wobei mindestens eine Ziffer geliefert wird:
[<Leerzeichen>...]{+|-}[Ziffer...][.][Ziffer...]<Leerzeichen>

Das Format ist so gewählt,

- dass der Wert mit den C-Funktionen `strtol()`, `atol()`, `srtod()`, `strtof()` und `atof()` in entsprechende numerische oder Gleitpunktvariablen von C übertragen werden kann.

- dass der Wert mit der COBOL-Funktion NUMVAL in entsprechende numerische COBOL-Variablen übertragen werden kann.

Bei RC 0 ist der Inhalt von `SequenceValue` ungeändert.

`RC`

Rückgabewert:

0	Der gewünschte Wert ist in den Speicherbereich übertragen worden, auf den sich SequenceValue bezieht.
-1	Die Eingabeparameter waren nicht korrekt oder die SQLda wurde nicht als Diagnosebereich erkannt.
100	Es konnte kein von SESAM/SQL vergebener Wert ermittelt werden. Mögliche Ursachen:
	• Die INSERT-Anweisung enthielt keinen '*' in der VALUES-Klausel und keine Angabe COUNT INTO.
	• Die Ausführung der INSERT-Anweisung war nicht erfolgreich und führte zu einem SQLSTATE der Klasse „Fehler“.
	• Die zuletzt mit der gleichen `SQLda` ausgeführte SQL-Anweisung war keine INSERT-Anweisung.

10 Informationsschemata

Dieses Kapitel beschreibt die Informationsschemata, die über die Struktur der Datenbank informieren.

Es beschreibt die Views des INFORMATION_SCHEMA und des SYS_INFO_SCHEMA.

10.1 Views des INFORMATION_SCHEMA

Im INFORMATION_SCHEMA finden Sie Informationen über Objekte der Datenbank. Jeder Berechtigungsschlüssel hat nur Zugriff auf die Objekte, für die er berechtigt ist. Die Views des INFORMATION_SCHEMA entsprechen dem SQL-Standard, so weit sie Objekte betreffen, die in SESAM/SQL und im SQL-Standard definiert sind. Das INFORMATION_SCHEMA enthält zusätzliche Views für SESAM/SQL-Erweiterungen.

Die folgende Tabelle zeigt, in welchen Views des INFORMATION_SCHEMA Informationen über welche Datenbankobjekte enthalten sind.

Die Views des INFORMATION_SCHEMA sind in den folgenden Abschnitten alphabetisch beschrieben.

Objekt	**Viewname**	**Information über**
Schema	SCHEMATA	Schemata der Datenbank
Tabelle	TABLES	Tabellen der Datenbank
	BASE_TABLES	Basistabellen der Datenbank
	PARTITIONS	Partitionen von Basistabellen
	VIEW_TABLE_USAGE	Tabellen, auf denen Views basieren
	CONSTRAINT_TABLE_USAGE	Tabellen, auf denen Integritätsbedingungen basieren
View	VIEWS	Views der Datenbank
Spalte	COLUMNS	Spalten der Datenbank
	BASE_TABLE_COLUMNS	Spalten in Basistabellen
	VIEW_COLUMN_USAGE	Spalten, auf denen Views basieren
	CONSTRAINT_COLUMN_USAGE	Spalten, auf denen Integritätsbedingungen basieren
	INDEX_COLUMN_USAGE	Spalten, auf denen Indizes basieren
	KEY_COLUMN_USAGE	Spalten, für die eine Primärschlüssel- oder Eindeutigkeitsbedingung definiert ist
Privileg	TABLE_PRIVILEGES	Tabellen-Privilegien
	COLUMN_PRIVILEGES	Spalten-Privilegien
	CATALOG_PRIVILEGES	Sonder-Privilegien
	USAGE_PRIVILEGES	USAGE-Privilegien
	ROUTINE_PRIVILEGES	Privilegien für Routinen
Index	INDEXES	Indizes der Datenbank

Integritätsbedingung	TABLE_CONSTRAINTS	Integritätsbedingungen
	REFERENTIAL_CONSTRAINTS	Referenz-Bedingungen
	CHECK_CONSTRAINTS	Check-Bedingungen
Storage Group	STOGROUPS	Storage Groups der Datenbank
Volume	STOGROUP_VOLUME_USAGE	für Storage Groups verwendete Datenträger
Space	SPACES	Spaces
Routinen	PARAMETERS	Parameter von Routinen
	ROUTINES	Routinen
	ROUTINE_ROUTINE_USAGE	Routinen in anderen Routinen
	ROUTINE_TABLE_USAGE	Tabellen in Routinen
	ROUTINE_COLUMN_USAGE	Spalten in Routinen
	VIEW_ROUTINE_USAGE	Routinen in Views
Benutzer	USERS	Berechtigungsschlüssel
	SYSTEM_ENTRIES	Systemzugänge
DA-LOG-Datei	DA_LOGS	DA-LOG-Dateien
Medientabelle	MEDIA_DESCRIPTIONS MEDIA_RECORDS	Mediensätze der datenbankspezifischen Dateien
Recovery Einheit	RECOVERY_UNITS	Recovery-Einheiten für Spaces
Zeichensatz	CHARACTER_SETS	Zeichensatz
Sortierreihenfolge	COLLATIONS	Sortierreihenfolge
Transliteration	TRANSLATIONS	Transliterationen
Features und Conformance	SQL_FEATURES SQL_IMPL_INFO SQL_LANGUAGES_S SQL_SIZING	Features, Subfeatures, Implementationen, implementierte Hostsprachen, Einbettungen und implementationsspezifische Maximalwerte

Tabelle 62: Views des INFORMATION_SCHEMA

10.1.1 BASE_TABLES

Information über Basistabellen. Der aktuelle Berechtigungsschlüssel muss mindestens ein Tabellen-Privileg für die Basistabelle oder das UTILITY-Privileg für die Datenbank besitzen.

Spaltenname	Datentyp	Inhalt
TABLE_CATALOG	CHAR (18)	Datenbankname
TABLE_SCHEMA	CHAR (31)	Name des Schemas, zu dem die Tabelle gehört
TABLE_NAME	CHAR (31)	Name der Basistabelle
SPACE_NAME	CHAR (18)	Name des Space, in dem die Basistabelle gespeichert ist. Wenn die Tabelle partitioniert ist, wird als Spacename “_PARTITIONS_” ausgegeben
TABLE_STYLE	VARCHAR (6)	OLDEST Nur-CALL-DML-Tabelle OLD CALL-DML/SQL-Tabelle NEW SQL-Tabelle

Tabelle 63: View BASE_TABLES des INFORMATION_SCHEMA

10.1.2 BASE_TABLE_COLUMNS

Information über Spalten von Basistabellen. Der aktuelle Berechtigungsschlüssel muss mindestens ein Spalten-Privileg für die Spalte oder das UTILITY-Privileg für die Datenbank besitzen.

Spaltenname	Datentyp	Inhalt
TABLE_CATALOG	CHAR (18)	Datenbankname
TABLE_SCHEMA	CHAR (31)	Name des Schemas, zu dem die Tabelle gehört
TABLE_NAME	CHAR (31)	Name der Basistabelle
COLUMN_NAME	CHAR (31)	Spaltenname
ORDINAL_POSITION	SMALLINT	laufende Nummer der Spalte in der Tabelle
COLUMN_DEFAULT	VARCHAR (256)	voreingestellter Wert, wie er bei der Spaltendefinition angegeben wurde (z.B. CHAR-Literal in Hochkommas), falls der aktuelle Berechtigungsschlüssel Schema Eigentümer ist TRUNCATED, wenn die Darstellung des voreingestellten Werts mehr als 256 Zeichen umfasst und der aktuelle Berechtigungsschlüssel Schema-Eigentümer ist. Der voreingestellte Wert kann nicht angezeigt werden. NULL-Wert sonst
IS_NULLABLE	VARCHAR (3)	NO Spalte darf mit Sicherheit keinen NULL-Wert annehmen YES sonst
DATA_TYPE	VARCHAR (24)	Datentyp der Spalte: CHARACTER CHARACTER VARYING NATIONAL CHAR NATIONAL CHAR VARYING REAL DOUBLE PRECISION FLOAT INTEGER SMALLINT NUMERIC

		DECIMAL DATE TIME TIMESTAMP OLDEST
CHARACTER _MAXIMUM_LENGTH	SMALLINT	max. Länge der Spalte in Code Units, falls Datentyp CHARACTER, CHARACTER VARYING, NATIONAL CHAR, NATIONAL CHAR VARYING oder OLDEST NULL-Wert sonst
NUMERIC_PRECISION	SMALLINT	Gesamtstellenzahl, falls numerischer Datentyp NULL-Wert sonst
NUMERIC_PRECISION _RADIX	SMALLINT	Basis der Stellenzahl, falls numerischer Datentyp NULL-Wert sonst
NUMERIC_SCALE	SMALLINT	Nachkommastellenzahl, falls exakter numerischer Datentyp NULL-Wert sonst
DATETIME_PRECISION	SMALLINT	Nachkommastellenzahl, falls Datentyp TIME oder TIMESTAMP NULL-Wert sonst
Die Spalten OLDEST_DESCRIPTOR* sind belegt, falls DATA_TYPE ist OLDEST:		
OLDEST_DESCRIPTOR1	CHAR (1)	Y linksbündig N nicht linksbündig NULL-Wert, wenn DATATYP nicht OLDEST
OLDEST_DESCRIPTOR2	CHAR (1)	Y Füllzeichen N keine Füllzeichen NULL-Wert, wenn DATATYPE nicht OLDEST
OLDEST_DESCRIPTOR3	CHAR (1)	Y Null (0) als Wert erlaubt N Null (0) nicht erlaubt

		NULL-Wert, wenn DATATYPE nicht OLDEST
OLDEST_DESCRIPTOR4	CHAR (1)	Y Wert hat arithmetisches Ergebnis N Wert hat kein arithmetisches Ergeb nis NULL-Wert, wenn DATATYPE nicht OLDEST
COLUMN_DESCRIPTOR1	CHAR (1)	Y Spalte besitzt genau einen einfachen Index und kommt nicht in zusammengesetztem Index vor N Spalte besitzt keinen oder mehr als einen einfachen Index oder kommt in zusammengesetztem Index vor
COLUMN_DESCRIPTOR2	CHAR (1)	Y Spalte besitzt genau einen zusammengesetzten Index und keinen einfachen Index N Spalte besitzt keinen Index, mehr als einen Index oder nur einen einfachen Index
COLUMN_DESCRIPTOR3	CHAR (1)	Y Spalte besitzt mehr als einen Index N Spalte besitzt höchstens einen Index
COLUMN_DESCRIPTOR4	CHAR (1)	Y Spalte besitzt eine CALL-DML-Voreinstellung N Spalte besitzt keine CALL-DML-Voreinstellung
COLUMN_DESCRIPTOR5	CHAR (1)	Y Spalte ist eine multiple Spalte N Spalte ist eine einfache Spalte
PK_DISTANCE	SMALLINT	Abstand der Spalte zum Anfang des Primärschlüssels NULL-Wert, falls Spalte nicht im Primärschlüssel
SESAM_SAN	CHAR (3)	Symbolischer Attributname der Spalte NULL-Wert, falls Spalte in SQL-Tabelle definiert
SESAM_DEFAULT	CHAR (2)	CALL-DML-Voreinstellung (ggf. mit Vorzeichen, falls numerischer Datentyp) NULL-Wert, falls Spalte in SQL-Tabelle definiert
FIRST_OCCURRENCE	SMALLINT	erste mögliche Ausprägung einer multiplen Spalte (= 1) NULL-Wert, falls Spalte nicht multipel

LAST_OCCURRENCE	SMALLINT	letzte mögliche Ausprägung einer multiplen Spalte NULL-Wert, falls Spalte nicht multipel

Tabelle 64: View BASE_TABLE_COLUMNS des INFORMATION_SCHEMA

10.1.3 CATALOG_PRIVILEGES

Information über die Privilegien für die Datenbank, die für den aktuellen Berechtigungsschlüssel verfügbar sind oder die von ihm vergeben wurden.

Spaltenname	Datentyp	Inhalt
GRANTOR	CHAR (18)	Berechtigungsschlüssel, der das Privileg vergab, bzw. _SYSTEM
GRANTEE	CHAR (18)	Berechtigungsschlüssel, dem das Privileg verliehen wurde, bzw. PUBLIC
CATALOG_NAME	CHAR (18)	Datenbankname
PRIVILEGE_TYPE	CHAR (18)	Privileg-Typ: CREATE USER CREATE SCHEMA CREATE STOGROUP UTILITY
IS_GRANTABLE	VARCHAR (3)	YES der Berechtigungsschlüssel besitzt GRANT-Berechtigung für das Privileg NO keine GRANT-Berechtigung

Tabelle 65: View CATALOG_PRIVILEGES des INFORMATION_SCHEMA

10.1.4 CHARACTER_SETS

Information über Zeichensätze, die für den aktuellen Berechtigungsschlüssel verfügbar sind.

Spaltenname	Datentyp	Inhalt
CHARACTER_SET_CATALOG	CHAR (18)	Datenbankname
CHARACTER_SET_SCHEMA	CHAR (31)	INFORMATION_SCHEMA
CHARACTER_SET_NAME	CHAR (18)	UTF16, EBCDIC, SQL_TEXT SQL_CHARACTER, SQL_IDENTIFIER
FORM_OF_USE	CHAR (18)	EBCDIC, UTF16
NUMBER_OF_CHARACTERS	INTEGER	bei FORM_OF_USE= EBCDIC die Anzahl Zeichen des Zeichensatzes, NULL-Wert sonst
DEFAULT_COLLATE_CATALOG	CHAR (18)	Datenbankname
DEFAULT_COLLATE_SCHEMA	CHAR (31)	INFORMATION_SCHEMA
DEFAULT_COLLATE_NAME	CHAR (18)	EBCDIC_BINARY UTF16_BINARY

Tabelle 66: View CHARACTER_SETS des INFORMATION_SCHEMA

10.1.5 CHECK_CONSTRAINTS

Information über Check-Bedingungen, die dem aktuellen Berechtigungsschlüssel gehören, sowie die zugehörige Check-Suchbedingung.

Spaltenname	Datentyp	Inhalt
CONSTRAINT_CATALOG	CHAR (18)	Datenbankname
CONSTRAINT_SCHEMA	CHAR (31)	Name des Schemas, zu dem die Tabelle mit der Check-Bedingung gehört
CONSTRAINT_NAME	CHAR (31)	Name der Check-Bedingung
CHECK_CLAUSE	VARCHAR (32000)	Suchbedingung

Tabelle 67: View CHECK_CONSTRAINTS des INFORMATION_SCHEMA

10.1.6 COLLATIONS

Information über Sortierreihenfolgen, die für den aktuellen Berechtigungsschlüssel verfügbar sind.

Spaltenname	Datentyp	Inhalt
COLLATION_CATALOG	CHAR (18)	Datenbankname
COLLATION_SCHEMA	CHAR (31)	INFORMATION_SCHEMA
COLLATION_NAME	CHAR (18)	DUCET_NO_VARS DUCET_WITH_VARS
CHARACTER_SET_CATALOG	CHAR (18)	Datenbankname
CHARACTER_SET_SCHEMA	CHAR (31)	INFORMATION_SCHEMA
CHARACTER_SET_NAME	CHAR (18)	UTF16
PAD_ATTRIBUTE	CHAR (9)	NO PAD

Tabelle 68: View COLLATIONS des INFORMATION_SCHEMA

10.1.7 COLUMNS

Information über alle Spalten von Basistabellen und Views, für die der aktuelle Berechtigungsschlüssel Privilegien besitzt.

Spaltenname	Datentyp	Inhalt
TABLE_CATALOG	CHAR (18)	Datenbankname
TABLE_SCHEMA	CHAR (31)	Name des Schemas, zu dem die Tabelle gehört
TABLE_NAME	CHAR (31)	Name der Basistabelle oder des View
COLUMN_NAME	CHAR (31)	Spaltenname
ORDINAL_POSITION	SMALLINT	laufende Nummer der Spalte in der Tabelle
COLUMN_DEFAULT	VARCHAR (256)	nur bei Basistabellen: voreingestellter Wert, wie er bei der Spaltendefinition angegeben wurde (z.B. CHAR-Literal in Hochkommas), falls der aktuelle Berechtigungsschlüssel Schema-Eigentümer ist. TRUNCATED, wenn die Darstellung des voreingestelltenWerts mehr als 256 Zeichen umfasst und der aktuelle Berechtigungsschlüssel Schema-Eigentümer ist. Der voreingestellte Wert kann nicht angezeigt werden. NULL-Wert sonst
IS_NULLABLE	VARCHAR (3)	NO Basistabellen-Spalte darf mit Sicherheit keinen NULL-Wert annehmen YES sonst
DATA_TYPE	VARCHAR (24)	Datentyp der Spalte: CHARACTER CHARACTER VARYING NATIONAL CHAR NATIONAL CHAR VARYING REAL DOUBLE PRECISION FLOAT INTEGER SMALLINT NUMERIC DECIMAL DATE

		TIME TIMESTAMP OLDEST
CHARACTER _MAXIMUM_LENGTH	SMALLINT	max. Länge der Spalte in Code Units, falls Datentyp CHARACTER, CHARACTER VARYING, NATIONAL CHAR, NATIONAL CHAR VARYING oder OLDEST NULL-Wert sonst
CHARACTER_OCTET _LENGTH	SMALLINT	max. Länge der Spalte in Bytes, falls Datentyp CHARACTER, CHARACTER VARYING, NATIONAL CHAR, NATIONAL CHAR VARYING oder OLDEST NULL-Wert sonst
NUMERIC_PRECISION	SMALLINT	Gesamtstellenzahl, falls numerischer Datentyp NULL-Wert sonst
NUMERIC_PRECISION _RADIX	SMALLINT	Basis der Stellenzahl, falls numerischer Datentyp NULL-Wert sonst
NUMERIC_SCALE	SMALLINT	Nachkommastellenzahl, falls exakter numerischer Datentyp NULL-Wert sonst
DATETIME _PRECISION	SMALLINT	Nachkommastellenzahl, falls Datentyp TIME oder TIMESTAMP NULL-Wert sonst
CHARACTER_SET _CATALOG	CHAR (18)	Datenbankname, falls Datentyp CHARACTER, CHARACTER VARYING, NATIONAL CHAR oder NATIONAL CHAR VARYING NULL-Wert sonst
CHARACTER_SET _SCHEMA	CHAR (31)	INFORMATION_SCHEMA, falls Datentyp CHARACTER, CHARACTER VARYING, NATIONAL CHAR oder NATIONAL CHAR VARYING

		NULL-Wert sonst
CHARACTER_SET_NAME	CHAR (18)	EBCDIC, falls Datentyp CHARACTER oder CHARACTER VARYING UTF16, falls Datentyp NATIONAL CHAR oder NATIONAL CHAR VARYING NULL-Wert sonst
COLLATION_CATALOG	CHAR (18)	Datenbankname, falls Datentyp CHARACTER, CHARACTER VARYING, NATIONAL CHAR oder NATIONAL CHAR VARYING NULL-Wert sonst
COLLATION_SCHEMA	CHAR (31)	INFORMATION_SCHEMA, falls Datentyp CHARACTER, CHARACTER VARYING, NATIONAL CHAR oder NATIONAL CHAR VARYING NULL-Wert sonst
COLLATION_NAME	CHAR (18)	EBCDIC_BINARY, falls Datentyp CHARACTER oder CHARACTER VARYING UTF16_BINARY, falls Datentyp NATIONAL CHAR oder NATIONAL CHAR VARYING NULL-Wert sonst
DOMAIN_CATALOG	CHAR (18)	NULL-Wert
DOMAIN_SCHEMA	CHAR (31)	NULL-Wert
DOMAIN_NAME	CHAR (31)	NULL-Wert
FIRST_OCCURRENCE	SMALLINT	erste mögliche Ausprägung einer multiplen Spalte (bei Basistabelle = 1) NULL-Wert, falls Spalte nicht multipel
LAST_OCCURRENCE	SMALLINT	letzte mögliche Ausprägung einer multiplen Spalte NULL-Wert, falls Spalte nicht multipel

Tabelle 69: View COLUMNS des INFORMATION_SCHEMA

10.1.8 COLUMN_PRIVILEGES

Information über alle Spalten-Privilegien, die für den aktuellen Berechtigungsschlüssel verfügbar sind oder die von ihm vergeben wurden.

Spaltenname	Datentyp	Inhalt
GRANTOR	CHAR (18)	Berechtigungsschlüssel, der das Privileg vergab bzw. _SYSTEM
GRANTEE	CHAR (18)	Berechtigungsschlüssel, dem das Privileg verliehen wurde, oder PUBLIC
TABLE_CATALOG	CHAR (18)	Datenbankname
TABLE_SCHEMA	CHAR (31)	Name des Schemas, zu dem die Spalte gehört, für die das Privileg gilt
TABLE_NAME	CHAR (31)	Name der Tabelle, für deren Spalte das Privileg gilt
COLUMN_NAME	CHAR (31)	Name der Spalte, auf die das Privileg eingeschränkt wurde
PRIVILEGE_TYPE	CHAR (18)	Privileg-Typ: SELECT INSERT UPDATE REFERENCES
IS_GRANTABLE	VARCHAR (3)	YES der Berechtigungsschlüssel besitzt GRANT-Berechtigung für das Privileg NO keine GRANT-Berechtigung

Tabelle 70: View COLUMN_PRIVILEGES des INFORMATION_SCHEMA

10.1.9 CONSTRAINT_COLUMN_USAGE

Information über Spalten, die dem aktuellen Berechtigungsschlüssel gehören und in Integritätsbedingungen angesprochen werden (außer Spalten, die in Referenz-Bedingungen referenziert werden).

Spaltenname	Datentyp	Inhalt
TABLE_CATALOG	CHAR (18)	Datenbankname
TABLE_SCHEMA	CHAR (31)	Name des Schemas, zu dem die Tabelle gehört
TABLE_NAME	CHAR (31)	Name der Tabelle, die in der Integritätsbedingung angesprochen wird
COLUMN_NAME	CHAR (31)	Spaltenname
CONSTRAINT_CATALOG	CHAR (18)	Datenbankname
CONSTRAINT_SCHEMA	CHAR (31)	Name des Schemas, zu dem die Tabelle mit der Integritätsbedingung gehört
CONSTRAINT_NAME	CHAR (31)	Name der Integritätsbedingung

Tabelle 71: View CONSTRAINT_COLUMN_USAGE des INFORMATION_SCHEMA

10.1.10 CONSTRAINT_TABLE_USAGE

Information über Tabellen, die dem aktuellen Berechtigungsschlüssel gehören und in Check- oder Referenzbedingungen angesprochen werden. Bei Referenz-Bedingungen werden nur die referenzierten Tabellen angezeigt.

Spaltenname	Datentyp	Inhalt
TABLE_CATALOG	CHAR (18)	Datenbankname
TABLE_SCHEMA	CHAR (31)	Name des Schemas, zu dem die Tabelle gehört
TABLE_NAME	CHAR (31)	Name der Tabelle, die in der Integritätsbedingung angesprochen wird
CONSTRAINT_CATALOG	CHAR (18)	Datenbankname
CONSTRAINT_SCHEMA	CHAR (31)	Name des Schemas, zu dem die Tabelle mit der Integritätsbedingung gehört
CONSTRAINT_NAME	CHAR (31)	Name der Integritätsbedingung

Tabelle 72: View CONSTRAINT_TABLE_USAGE des INFORMATION_SCHEMA

10.1.11 DA_LOGS

Information über DA-LOG-Dateien einer Datenbank. Der aktuelle Berechtigungsschlüssel muss das UTILITY-Privileg für die Datenbank besitzen oder Eigentümer mindestens eines Anwender-Space der Datenbank sein.

Spaltenname	Datentyp	Inhalt
DALOG_CATALOG	CHAR (18)	Datenbankname
DALOG_VERSION	INTEGER	Versionsnummer der DA-LOG-Datei
DALOG_SUBNUMBER	INTEGER	laufende Nummer der DA-LOG-Datei innerhalb der Version
DALOG_INIT	TIMESTAMP (3)	Erstellungszeitpunkt

Tabelle 73: View DA_LOGS des INFORMATION_SCHEMA

10.1.12 INDEXES

Information über die Indizes, die dem aktuellen Berechtigungsschlüssel gehören. Der aktuelle Berechtigungsschlüssel muss das UTILITY-Privileg für die Datenbank besitzen oder Eigentümer des Schemas sein, in dem der Index definiert ist.

Spaltenname	Datentyp	Inhalt
INDEX_CATALOG	CHAR (18)	Datenbankname
INDEX_SCHEMA	CHAR (31)	Name des Schemas, zu dem der Index gehört
INDEX_NAME	CHAR (18)	Name des Index
TABLE_NAME	CHAR (31)	Name der Basistabelle, zu der der Index gehört
SPACE_NAME	CHAR (18)	Name des Space, in dem der Index gespeichert ist
LENGTH_I	SMALLINT	Gesamtlänge des Index
CONSTRAINT_NAME	CHAR (31)	Name der Eindeutigkeitsbedingung, wenn der Index von einer Eindeutigkeitsbedingung verwendet wird NULL-Wert sonst
STATE	VARCHAR (9)	Status: GENERATED DEFECT
GENERATE_TYPE	VARCHAR (8)	wie erzeugt: EXPLICIT IMPLICIT
STATISTICS_INFO	VARCHAR (3)	YES Statistik-Information vorhanden NO nicht vorhanden
INDEX_TYPE	VARCHAR (8)	Index-Typ: SINGLE COMPOUND

Tabelle 74: View INDEXES des INFORMATION_SCHEMA

10.1.13 INDEX_COLUMN_USAGE

Information über die Spalten der Indizes, die dem aktuellen Berechtigungsschlüssel gehören..

Spaltenname	Datentyp	Inhalt
INDEX_CATALOG	CHAR (18)	Datenbankname
INDEX_SCHEMA	CHAR (31)	Name des Schemas, zu dem der Index gehört
INDEX_NAME	CHAR (18)	Name des Index
TABLE_NAME	CHAR (31)	Name der Basistabelle, zu der der Index gehört
COLUMN_NAME	CHAR (31)	Name der Spalte des Index
ORDINAL_POSITION	SMALLINT	Position der Spalte im Index
LENGTH_C	SMALLINT	gibt an, bis zu welcher Länge (in Bytes) die Spalte in den Index einbezogen ist
INDEX_DISTANCE	SMALLINT	Abstand der Spalte zum Indexanfang
DATE_TYPE_C	VARCHAR (24)	Datentyp der Spalte CHARACTER CHARACTER VARYING NATIONAL CHAR NATIONAL CHAR VARYING REAL DOUBLE PRECISION FLOAT INTEGER SMALLINT NUMERIC DECIMAL DATE TIME TIMESTAMP OLDEST

Tabelle 75: View INDEX_COLUMN_USAGE des INFORMATION_SCHEMA

10.1.14 KEY_COLUMN_USAGE

Information über Primärschlüssel- und Eindeutigkeitsbedingungen, die dem aktuellen Berechtigungsschlüssel gehören, sowie die Namen der entsprechenden Spalten.

Zusätzlich wird über die Referenzbedingungen, die dem aktuellen Berechtigungsschlüssel gehören, sowie die Namen der referenzierenden Spalten informiert.

Spaltenname	Datentyp	Inhalt
CONSTRAINT_CATALOG	CHAR (18)	Datenbankname
CONSTRAINT_SCHEMA	CHAR (31)	Name des Schemas, zu dem die Tabelle mit der Integritätsbedingung gehört
CONSTRAINT_NAME	CHAR (31)	Name der Integritätsbedingung
TABLE_CATALOG	CHAR (18)	Datenbankname
TABLE_SCHEMA	CHAR (31)	Name des Schemas, zu dem die Tabelle gehört
TABLE_NAME	CHAR (31)	Name der Tabelle, zu dem die Integritätsbedingung gehört
COLUMN_NAME	CHAR (31)	Name einer Spalte der Integritätsbedingung
ORDINAL_POSITION	SMALLINT	Position der Spalte in der Integritätsbedingung

Tabelle 76: View KEY_COLUMN_USAGE des INFORMATION_SCHEMA

10.1.15 MEDIA_DESCRIPTIONS

Information über Dateieigenschaften für datenbankspezifische Dateien. Der aktuelle Berechtigungsschlüssel muss das UTILITY-Privileg für die Datenbank besitzen.

Spaltenname	Datentyp	Inhalt
MEDIA_CATALOG	CHAR (18)	Datenbankname
FILE_TYPE	CHAR (6)	Dateityp: DALOG CATLOG PBI CATREC DDLTA
REQUESTS	VARCHAR (3)	YES Datenträgeranforderung an Konsole möglich NO keine Datenträgeranforderung an Konsole
PRIMARY_ALLOC	INTEGER	Primärzuweisung
SECONDARY_ALLOC	INTEGER	Sekundärzuweisung
SHARABLE	VARCHAR (3)	Datei mehrfachbenutzbar: YES NO

Tabelle 77: View MEDIA_DESCRIPTIONS des INFORMATION_SCHEMA

10.1.16 MEDIA_RECORDS

Information über Datenträgertypen für datenbankspezifische Dateien. Der aktuelle Berechtigungsschlüssel muss das UTILITY-Privileg für die Datenbank besitzen.

Spaltenname	Datentyp	Inhalt
MEDIA_CATALOG	CHAR (18)	Datenbankname
FILE_TYPE	CHAR (6)	Dateityp: DALOG CATLOG PBI CATREC DDLTA
DEVICE_DESCRIPTOR	CHAR (18)	Gerätetyp oder Name der Storage Group für die Datei
MEDIUM	CHAR (4)	DISC
ORDINAL_POSITION	SMALLINT	laufende Nummer des Eintrags in der Medientabelle

Tabelle 78: View MEDIA_RECORDS des INFORMATION_SCHEMA

10.1.17 PARAMETERS

Information über Parameter von Routinen (Prozeduren und UDFs), für die der aktuelle Berechtigungsschlüssel Privilegien besitzt.

Spaltenname	Datentyp	Inhalt
SPECIFIC_CATALOG	CHAR(18)	Datenbankname
SPECIFIC_SCHEMA	CHAR(31)	Name des Schemas, zu dem die Routine gehört
SPECIFIC_NAME	CHAR(31)	Spezifischer Name der Routine
ORDINAL_POSITION	SMALLINT	Laufende Nummer des Parameters in der Routine
PARAMETER_MODE	VARCHAR(5)	IN Eingabeparameter OUT Ausgabeparameter INOUT Ein- und Ausgabeparameter
IS_RESULT	VARCHAR(3)	NO für SESAM/SQL irrelevant
AS_LOCATOR	VARCHAR(3)	NO für SESAM/SQL irrelevant
PARAMETER_NAME	CHAR(31)	Name des Parameters
DATA_TYPE	VARCHAR(24)	Datentyp des Parameters: CHARACTER CHARACTER VARYING NATIONAL CHAR NATIONAL CHAR VARYING REAL DOUBLE PRECISION FLOAT INTEGER SMALLINT NUMERIC DECIMAL DATE TIME TIMESTAMP
CHARACTER_ MAXIMUM_LENGTH	SMALLINT	max. Länge des Parameters in Code Units, falls Datentyp CHARACTER, CHARACTER VARYING, NATIONAL CHAR oder NATIONAL CHAR VARYING NULL-Wert sonst

CHARACTER_OCTET_ LENGTH	SMALLINT	max. Länge des Parameters in Bytes, falls Datentyp CHARACTER, CHARACTER VARYING, NATIONAL CHAR oder NATIONAL CHAR VARYING NULL-Wert sonst
CHARACTER_SET_ CATALOG	CHAR(18)	Datenbankname, falls Datentyp CHARACTER, CHARACTER VARYING, NATIONAL CHAR oder NATIONAL CHAR VARYING NULL-Wert sonst
CHARACTER_SET_ SCHEMA	CHAR(31)	INFORMATION_SCHEMA, falls Datentyp CHARACTER, CHARACTER VARYING, NATIONAL CHAR oder NATIONAL CHAR VARYING NULL-Wert sonst
CHARACTER_SET_ NAME	CHAR(18)	EBCDIC, falls Datentyp CHARACTER oder CHARACTER VARYING UTF16, falls Datentyp NATIONAL CHAR oder NATIONAL CHAR VARYING NULL-Wert sonst
COLLATION_CATALOG	CHAR(18)	Datenbankname, falls Datentyp CHARACTER, CHARACTER VARYING, NATIONAL CHAR oder NATIONAL CHAR VARYING NULL-Wert sonst
COLLATION_SCHEMA	CHAR(31)	INFORMATION_SCHEMA, falls Datentyp CHARACTER, CHARACTER VARYING, NATIONAL CHAR oder NATIONAL CHAR VARYING NULL-Wert sonst
COLLATION_NAME	CHAR(18)	EBCDIC_BINARY,

		falls Datentyp CHARACTER oder CHARACTER VARYING UTF16_BINARY, falls Datentyp NATIONAL CHAR oder NATIONAL CHAR VARYING NULL-Wert sonst
NUMERIC_PRECISION	SMALLINT	Gesamtstellenzahl, falls numerischer Datentyp NULL-Wert sonst
NUMERIC_PRECISI-ON_RADIX	SMALLINT	Basis der Stellenzahl, falls numerischer Datentyp NULL-Wert sonst
NUMERIC_SCALE	SMALLINT	Nachkommastellenzahl, falls exakter numerischer Datentyp NULL-Wert sonst
DATETIME_PRECISION	SMALLINT	Nachkommastellenzahl, falls Datentyp TIME oder TIMESTAMP NULL-Wert sonst

Tabelle 79: View PARAMETERS des INFORMATION_SCHEMA

10.1.18 PARTITIONS

Information über Partitionen von Basistabellen. Der aktuelle Berechtigungsschlüssel muss das UTILITY-Privileg für die Datenbank besitzen oder Eigentümer der Tabelle sein.

Spaltenname	Datentyp	Inhalt
PARTITION_CATALOG	CHAR (18)	Datenbankname
TABLE_SCHEMA	CHAR (31)	Name des Schemas, zu dem die partitionierte Tabelle gehört
TABLE_NAME	CHAR (31)	Name der partitionierten Tabelle
SERIAL_NUMBER	SMALLINT	Laufende Nummer der Partition
MAX_KEY_VALUE	VARCHAR (32000)	Vergleich für die obere Partitionsgrenze, wie in der VALUE-Klausel angegeben (externe Darstellung)
SPACE_NAME	CHAR (18)	Name des Space, in dem die Partition gespeichert ist

Tabelle 80: View PARTITIONS des INFORMATION_SCHEMA

10.1.19 RECOVERY_UNITS

Information über Recovery-Einheiten für Spaces. Der aktuelle Berechtigungsschlüssel muss das UTILITY-Privileg für die Datenbank besitzen oder Eigentümer des Space sein.

Spaltenname	Datentyp	Inhalt
SPACE_CATALOG	CHAR (18)	Datenbankname
SPACE_NAME	CHAR (18)	Name des Space
RECOVERY_TIMESTAMP	TIMESTAMP (3)	Zeitpunkt der Recovery-Operation
VERSION	INTEGER	interne Nummer, falls RECOVERY_TYPE ist COPY NULL-Wert sonst
VALIDITY	VARCHAR (3)	YES Recovery-Einheit gültig für Reparatur zur nächsten Recovery-Einheit NO ungültig (kann sich durch eine RECOVER-Anweisung aber eventuell zu YES ändern) NOT ungültig (kann sich nicht mehr ändern)
RECOVERY_UNIT_NAME	VARCHAR (54)	Dateiname der Kopie, falls RECOVERY_TYPE ist COPY interne Nummer, falls RECOVERY_TYPE ist RESTART oder REST_TO NULL-Wert sonst
SPACE_OWNER	CHAR (18)	Berechtigungsschlüssel, der Eigentümer des Space ist
MEDIUM	CHAR (4)	DISC SESAM-Sicherung auf Platte TAPE SESAM-Sicherung mit ARCHIVE HSMW SESAM-Sicherung mit HSMS (Arbeitsdatei) HSMB SESAM-Sicherung mit HSMS (Additional-Mirror-Unit) SRDF SESAM-Sicherung mit HSMS (SRDF-Target) falls RECOVERY_TYPE ist COPY NULL-Wert sonst
RECOVERY_TYPE	VARCHAR (7)	

		Werte, die von der Utility Recovery ausgewertet werden: COPY CREATE RESTART REST_TO (RESTART TO) MARK	
COPY_TYPE	VARCHAR (7)	ONLINE oder OFFLINE, falls RECOVERY_TYPE ist COPY NULL-Wert sonst	
DALOG_VERSION	INTEGER	Versionsnummer der DA-LOG-Datei	DA-LOG-Pegel vor dem Eintrag der Recovery-Einheit
DALOG_SUBNUMBER	INTEGER	laufende Nummer der DA-LOG-Datei innerhalb der Version	DA-LOG-Pegel vor dem Eintrag der Recovery-Einheit
NEXT_DALOG_VERSION	INTEGER	Versionsnummer der DA-LOG-Datei	DA-LOG-Pegel nach dem Eintrag der Recovery-Einheit
NEXT_DALOG_SUBNUMBER	INTEGER	laufende Nummer der DA-LOG-Datei innerhalb der Version	DA-LOG-Pegel nach dem Eintrag der Recovery-Einheit
ARCHIVE_DIRECTORY_NAME	VARCHAR (54)	Name des ARCHIVE-Directory, falls MEDIUM = 'TAPE' Name des HSMS-Archivs, falls MEDIUM = 'HSMS', 'HSMW', 'HSMB' oder 'SRDF' NULL-Wert sonst	
PBI_TIMESTAMP	TIMESTAMP (3)	Erzeugungszeitpunkt der PBI-Datei	
PBI_COUNTER	INTEGER	undefiniert	

Tabelle 81: View RECOVERY_UNITS des INFORMATION_SCHEMA

10.1.20 REFERENTIAL_CONSTRAINTS

Information über Referenzbedingungen, die dem aktuellen Berechtigungsschlüssel gehören, sowie der Name der referenzierten Eindeutigkeits- bzw. Primärschlüsselbedingungen.

Spaltenname	Datentyp	Inhalt
CONSTRAINT_CATALOG	CHAR (18)	Datenbankname
CONSTRAINT_SCHEMA	CHAR (31)	Name des Schemas, zu dem die Tabelle mit der Referenzbedingung gehört
CONSTRAINT_NAME	CHAR (31)	Name der Referenzbedingung
UNIQUE_CONSTRAINT_CATALOG	CHAR (18)	Datenbankname
UNIQUE_CONSTRAINT_SCHEMA	CHAR (31)	Name des Schemas der referenzierten Tabelle
UNIQUE_CONSTRAINT_NAME	CHAR (31)	Name der Eindeutigkeits- bzw. Primärschlüsselbedingung der referenzierten Tabelle
MATCH_OPTION	CHAR (7)	NONE
UPDATE_RULE	CHAR (11)	NO ACTION
DELETE_RULE	CHAR (11)	NO ACTION

Tabelle 82: View REFERENTIAL_CONSTRAINTS des INFORMATION_SCHEMA

10.1.21 ROUTINES

Information über Routinen (Prozeduren und UDFs), für die der aktuelle Berechtigungsschlüssel Privilegien besitzt.

Spaltenname	Datentyp	Inhalt
SPECIFIC_CATALOG	CHAR(18)	Datenbankname
SPECIFIC_SCHEMA	CHAR(31)	Name des Schemas, zu dem die Routine gehört
SPECIFIC_NAME	CHAR(31)	spezifischer Name der Routine
ROUTINE_CATALOG	CHAR(18)	Datenbankname
ROUTINE_SCHEMA	CHAR(31)	Name des Schemas, zu dem die Routine gehört
ROUTINE_NAME	CHAR(31)	Name der Routine
ROUTINE_TYPE	VARCHAR(28)	PROCEDURE, falls Prozedur FUNCTION, falls UDF
DATA_TYPE	VARCHAR(24)	Datentyp des Rückgabewerts einer UDF CHARACTER CHARACTER VARYING NATIONAL CHAR NATIONAL CHAR VARYING REAL DOUBLE PRECISION FLOAT INTEGER SMALLINT NUMERIC DECIMAL DATE TIME TIMESTAMP NULL-Wert, falls Prozedur
CHARACTER_MAXIMUM_LENGTH	SMALLINT	max. Länge des Rückgabewerts in Code Units, falls Datentyp CHARACTER, CHARACTER VARYING, NATIONAL CHAR oder NATIONAL CHAR VARYING NULL-Wert sonst

CHARACTER_OCTET_LENGTH	SMALLINT	max. Länge des Rückgabewerts in Bytes, falls Datentyp CHARACTER, CHARACTER VARYING, NATIONAL CHAR oder NATIONAL CHAR VARYING NULL-Wert sonst
CHARACTER_SET_CATALOG	CHAR(18)	Datenbankname, falls Datentyp CHARACTER, CHARACTER VARYING, NATIONAL CHAR oder NATIONAL CHAR VARYING NULL-Wert sonst
CHARACTER_SET_SCHEMA	CHAR(31)	INFORMATION_SCHEMA, falls Datentyp CHARACTER, CHARACTER VARYING, NATIONAL CHAR oder NATIONAL CHAR VARYING NULL-Wert sonst
CHARACTER_SET_NAME	CHAR(18)	EBCDIC, falls Datentyp CHARACTER oder CHARACTER VARYING UTF16, falls Datentyp NATIONAL CHAR oder NATIONAL CHAR VARYING NULL-Wert sonst
COLLATION_CATALOG	CHAR(18)	Datenbankname, falls Datentyp CHARACTER, CHARACTER VARYING, NATIONAL CHAR oder NATIONAL CHAR VARYING NULL-Wert sonst
COLLATION_SCHEMA	CHAR(31)	INFORMATION_SCHEMA, falls Datentyp CHARACTER, CHARACTER VARYING, NATIONAL CHAR oder NATIONAL CHAR VARYING NULL-Wert sonst

COLLATION_NAME	CHAR(18)	EBCDIC_BINARY, falls Datentyp CHARACTER oder CHARACTER VARYING UTF16_BINARY, falls Datentyp NATIONAL CHAR oder NATIONAL CHAR VARYING NULL-Wert sonst
NUMERIC_PRECISION	SMALLINT	Gesamtstellenzahl, falls numerischer Datentyp NULL-Wert sonst
NUMERIC_PRECISION_RADIX	SMALLINT	Basis der Stellenzahl, falls numerischer Datentyp NULL-Wert sonst
NUMERIC_SCALE	SMALLINT	Nachkommastellenzahl, falls exakter numerischer Datentyp NULL-Wert sonst
DATETIME_PRECISION	SMALLINT	Nachkommastellenzahl, falls Datentyp TIME oder TIMESTAMP NULL-Wert sonst
ROUTINE_BODY	VARCHAR(8)	SQL Programmiersprache, in der die Routine geschrieben ist
ROUTINE_DEFINITION	VARCHAR(32000)	Text der Routine, falls der aktuelle Berechtigungsschlüssel Schema-Eigentümer ist NULL-Wert sonst
EXTERNAL_NAME	CHAR(31)	NULL-Wert, irrelevant für SESAM/SQL
EXTERNAL_LANGUAGE	VARCHAR(7)	NULL-Wert, irrelevant für SESAM/SQL
PARAMETER_STYLE	VARCHAR(7)	NULL-Wert, irrelevant für SESAM/SQL
IS_DETERMINISTIC	VARCHAR(3)	NO irrelevant für SESAM/SQL
SQL_DATA_ACCESS	VARCHAR(17)	CONTAINS SQL, falls bei der Definition der Routine CONTAINS SQL angegeben wurde

		READS SQL DATA falls bei der Definition der Routine READS SQL DATA angegeben wurde MODIFIES SQL DATA falls bei der Definition der Routine MODIFIES SQL DATA angegeben wurde
IS_NULL_CALL	VARCHAR(3)	NO, falls UDF NULL-Wert sonst
SQL_PATH	VARCHAR(256)	SQL-Pfad Entspricht in SESAM/SQL dem Namen des Schemas, in dem die Routine definiert ist
SCHEMA_LEVEL_ROUTINE	VARCHAR(3)	YES Ist Bestandteil eines Schemas
MAX_DYNAMIC_RESULT_SETS	SMALLINT	0 irrelevant für SESAM/SQL
IS_USER_DEFINED_CAST	VARCHAR(3)	NO, falls UDF NULL-Wert sonst
IS_IMPLICITLY_INVOCABLE	VARCHAR(3)	NULL-Wert, irrelevant für SESAM/SQL
SECURITY_TYPE	VARCHAR(22)	NULL-Wert, irrelevant für SESAM/SQL
AS_LOCATOR	VARCHAR(3)	NO, falls UDF NULL-Wert sonst
NEW_SAVEPOINT_LEVEL	VARCHAR(3)	NULL-Wert, irrelevant für SESAM/SQL
IS_UDT_DEPENDENT	VARCHAR(3)	NO irrelevant für SESAM/SQL

Tabelle 83: View ROUTINES des INFORMATION_SCHEMA

10.1.22 ROUTINE_COLUMN_USAGE

Information über die Routinen (Prozeduren und UDFs), die Spalten ansprechen, die dem aktuellen Berechtigungsschlüssel gehören, und die Namen der Spalten.

Spaltenname	Datentyp	Inhalt
SPECIFIC_CATALOG	CHAR(18)	Datenbankname
SPECIFIC_SCHEMA	CHAR(31)	Name des Schemas, zu dem die Routine gehört
SPECIFIC_NAME	CHAR(31)	spezifischer Name der Routine
ROUTINE_CATALOG	CHAR(18)	Datenbankname
ROUTINE_SCHEMA	CHAR(31)	Name des Schemas, zu dem die Routine gehört
ROUTINE_NAME	CHAR(31)	Name der Routine
TABLE_CATALOG	CHAR(18)	Datenbankname
TABLE_SCHEMA	CHAR(31)	Name des Schemas, zu dem die Tabelle gehört, die in der Routine angesprochen wird
TABLE_NAME	CHAR(31)	Name der Tabelle, die in der Routine verwendet wird
COLUMN_NAME	CHAR(31)	Spaltenname

Tabelle 84: View ROUTINE_COLUMN_USAGE des INFORMATION_SCHEMA

10.1.23 ROUTINE_PRIVILEGES

Information über die Privilegien für Routinen (Prozeduren und UDFs), die der aktuelle Berechtigungsschlüssel besitzt oder die von ihm vergeben wurden.

Spaltenname	Datentyp	Inhalt
GRANTOR	CHAR(18)	Berechtigungsschlüssel, der das Privileg vergab, oder _SYSTEM
GRANTEE	CHAR(18)	Berechtigungsschlüssel, dem das Privileg verliehen wurde, oder PUBLIC
SPECIFIC_CATALOG	CHAR(18)	Datenbankname
SPECIFIC_SCHEMA	CHAR(31)	Name des Schemas, zu dem die Routine gehört
SPECIFIC_NAME	CHAR(31)	spezifischer Name der Routine
ROUTINE_CATALOG	CHAR(18)	Datenbankname
ROUTINE_SCHEMA	CHAR(31)	Name des Schemas, zu dem die Routine gehört
ROUTINE_NAME	CHAR(31)	Name der Routine
PRIVILEGE_TYPE	CHAR(18)	EXECUTE
IS_GRANTABLE	VARCHAR(3)	YES der Berechtigungsschlüssel besitzt GRANT-Berechtigung für das Privileg NO keine GRANT-Berechtigung

Tabelle 85: View ROUTINE_PRIVILEGES des INFORMATION_SCHEMA

10.1.24 ROUTINE_ROUTINE_USAGE

Information über die Routinen (Prozeduren und UDFs), die dem aktuellen Berechtigungsschlüssel gehören und die in anderen Routinen aufgerufen werden.

Spaltenname	Datentyp	Inhalt
SPECIFIC_CATALOG	CHAR(18)	Datenbankname
SPECIFIC_SCHEMA	CHAR(31)	Name des Schemas, zu dem die aufrufende Routine gehört
SPECIFIC_NAME	CHAR(31)	spezifischer Name der aufrufenden Routine
ROUTINE_CATALOG	CHAR(18)	Datenbankname
ROUTINE_SCHEMA	CHAR(31)	Name des Schemas, zu dem die aufgerufene Routine gehört
ROUTINE_NAME	CHAR(31)	spezifischer Name der aufgerufenen Routine

Tabelle 86: View ROUTINE_ROUTINE_USAGE des INFORMATION_SCHEMA

10.1.25 ROUTINE_TABLE_USAGE

Information über die Tabellen, die dem aktuellen Berechtigungsschlüssel gehören und die in Routinen (Prozeduren und UDFs) angesprochen werden.

Spaltenname	Datentyp	Inhalt
SPECIFIC_CATALOG	CHAR(18)	Datenbankname
SPECIFIC_SCHEMA	CHAR(31)	Name des Schemas, zu dem die Routine gehört
SPECIFIC_NAME	CHAR(31)	spezifischer Name der Routine
ROUTINE_CATALOG	CHAR(18)	Datenbankname
ROUTINE_SCHEMA	CHAR(31)	Name des Schemas, zu dem die Routine gehört
ROUTINE_NAME	CHAR(31)	Name der Routine
TABLE_CATALOG	CHAR(18)	Datenbankname
TABLE_SCHEMA	CHAR(31)	Name des Schemas, zu dem die Tabelle gehört, die in der Routine angesprochen wird
TABLE_NAME	CHAR(31)	Name der Tabelle, die in der Routine verwendet wird

Tabelle 87: View ROUTINE_TABLE_USAGE des INFORMATION_SCHEMA

10.1.26 SCHEMATA

Information über alle Schemata, die dem aktuellen Berechtigungsschlüssel gehören.

Spaltenname	Datentyp	Inhalt
CATALOG_NAME	CHAR (18)	Datenbankname
SCHEMA_NAME	CHAR (31)	Name des Schemas
SCHEMA_OWNER	CHAR (18)	Berechtigungsschlüssel des Eigentümers
DEFAULT_CHARACTER_SET_CATALOG	CHAR (18)	Datenbankname
DEFAULT_CHARACTER_SET_SCHEMA	CHAR (31)	INFORMATION_SCHEMA
DEFAULT_CHARACTER_SET_NAME	CHAR (18)	EBCDIC

Tabelle 88: View SCHEMATA des INFORMATION_SCHEMA

10.1.27 SPACES

Information über die Spaces, die dem aktuellen Berechtigungsschlüssel gehören oder auf die er mit Utilities zugreifen darf.

Spaltenname	Datentyp	Inhalt
SPACE_CATALOG	CHAR (18)	Datenbankname
SPACE_NAME	CHAR (18)	Name des Space
SPACE_OWNER	CHAR (18)	Berechtigungsschlüssel, der Eigentümer des Space ist
STOGROUP_NAME	CHAR (18)	Name der Storage Group für den Space
PCT_FREE	SMALLINT	Freiplatzreservierung in Prozent
LOGGING	VARCHAR (3)	YES Logging eingeschaltet NO Logging ausgeschaltet
Die Daten von STOGROUP_NAME und PCT_FREE werden durch ALTER SPACE geändert; diese Daten werden erst bei RECOVER und REORG tatsächlich berücksichtigt.		

Tabelle 89: View SPACES des INFORMATION_SCHEMA

10.1.28 SQL_FEATURES

Information über die Features der SQL-Norm und ihrer Subfeatures im implementierten Sprachumfang. Die Bezeichnungen sind durch die SQL-Norm festgelegt.

Spaltenname	Datentyp	Inhalt
FEATURE_ID	VARCHAR (256)	Featurekennung (z.B. F111)
FEATURE_NAME	VARCHAR (256)	Featurename (z.B. “Isolation levels other than SERIALIZABLE”)
SUB_FEATURE_ID	VARCHAR (256)	Subfeaturekennung (z.B. 02)
SUB_FEATURE_NAME	VARCHAR (256)	Subfeaturename (z.B. “READ COMITTED isolation level”)
IS_SUPPORTED	VARCHAR (3)	YES wird voll unterstützt NO wird nicht oder nur teilweise unterstützt
IS_VERIFIED_BY	VARCHAR (256)	NULL-Wert
COMMENTS	VARCHAR (256)	Bemerkungen

Tabelle 90: View SQL_FEATURES des INFORMATION_SCHEMA

10.1.29 SQL_IMPL_INFO

Information über Eigenschaften der Implementation. Die Bezeichnungen sind durch die SQL-Norm festgelegt.

Spaltenname	Datentyp	Inhalt
IMPL_INFO_ID	VARCHAR (256)	Kennung für eine Implementationseigenschaft
IMPL_INFO_NAME	VARCHAR (256)	Name einer Implementationseigenschaft
INTEGER_VALUE[1]	INTEGER	Numerischer Wert für eine Implementationseigenschaft
CHARACTER_- VALUE[1]	VARCHAR (256)	Alphanumerischer Wert für eine Implementationseigenschaft
COMMENTS	VARCHAR (256)	Bemerkungen

Tabelle 91: View SQL_IMPL_INFO des INFORMATION_SCHEMA

[1]Genau eine der beiden Spalten INTEGER_VALUE oder CHARACTER_VALUE hat den NULL-Wert, je nachdem, ob für die Implementationseigenschaft ein numerischer oder alphanumerischer Wert vorliegt.

10.1.30 SQL_LANGUAGES_S

Information über die implementierten Hostsprachen und Einbettungen. Die Bezeichnungen sind durch die SQL-Norm festgelegt.

Spaltenname	Datentyp	Inhalt
SOURCE	VARCHAR (256)	ISO-Standard: 'ISO 9075'
SQL_LANGUAGE_YEAR	VARCHAR (256)	Erscheinungsjahr des Standards: '1989' '1992' '1999' '2008'
CONFORMANCE	VARCHAR (256)	Sprachumfang: '2' 'ENTRY' 'CORE'
INTEGRITY	VARCHAR (256)	'YES' “integrity enhancement” von SQL89 ist implementiert. NULL-Wert falls SQL_LANGUAGE_YEAR '1989'
IMPLEMENTATION	VARCHAR (256)	Angabe eines implementationsdefinierten Standards, falls SOURCE 'ISO 9075'
BINDING_STYLE	VARCHAR (256)	Art der Einbettung: 'EMBEDDED'
PROGRAMMING_ LANGUAGE	VARCHAR (256)	unterstützte Programmiersprache: 'COBOL'

Tabelle 92: View SQL_LANGUAGES_S des INFORMATION_SCHEMA

10.1.31 SQL_SIZING

Information über Maximalwerte der Implementation. Die Bezeichnungen sind durch die SQL-Norm festgelegt.

Spaltenname	Datentyp	Inhalt
SIZING_ID	INTEGER	Kennung des Maximalwerts
SIZING_NAME	VARCHAR (256)	Name des Maximalwerts
SUPPORTED_VALUE	INTEGER	Maximalgröße des Werts: Maximalwert 0 kein Maximalwert oder Maximalwert nicht bekannt oder variabel NULL-Wert Maximalwert spielt für SESAM/SQL keine Rolle
COMMENTS	VARCHAR (256)	Bemerkungen

Tabelle 93: View SQL_SIZING des INFORMATION_SCHEMA

10.1.32 STOGROUPS

Information über die Storage Groups, auf die der aktuelle Berechtigungsschlüssel zugreifen darf.

Spaltenname	Datentyp	Inhalt
STOGROUP_CATALOG	CHAR (18)	Datenbankname
STOGROUP_NAME	CHAR (18)	Name der Storage Group
STOGROUP_OWNER	CHAR (18)	Berechtigungsschlüssel der Eigentümer der Storage Group
CAT_ID	VARCHAR (4)	BS2000-Katalogkennung

Tabelle 94: View STOGROUPS des INFORMATION_SCHEMA

10.1.33 STOGROUP_VOLUME_USAGE

Information über die Datenträger von Storage Groups, die dem aktuellen Berechtigungsschlüssel gehören.

Spaltenname	Datentyp	Inhalt
STOGROUP_CATALOG	CHAR (18)	Datenbankname
STOGROUP_NAME	CHAR (18)	Name der Storage Group
VOLUME_NAME	CHAR (6)	Datenträgerkennzeichen von Privatplatten oder PUBLIC
DEVICE_TYPE	VARCHAR (8)	Gerätetyp der Privatplatten NULL-Wert für PUBLIC
ORDINAL_POSITION	SMALLINT	laufenden Nummer der Privatplatte in der Storage Group (bei PUBLIC 1)

Tabelle 95: View STOGROUP_VOLUME_USAGE des INFORMATION_SCHEMA

10.1.34 SYSTEM_ENTRIES

Information über Systemzugänge.

- Aktueller Berechtigungsschlüssel ohne CREATE USER-Privileg:
  Alle Systemzugänge zum eigenen Berechtigungsschlüssel, in denen der Berechtigungsschlüssel explizit eingetragen ist.
- Aktueller Berechtigungsschlüssel mit CREATE USER-Privileg, aber ohne GRANT-Berechtigung:
  Alle Systemzugänge zu Berechtigungsschlüsseln, die das CREATE USER-Privileg nicht besitzen oder ohne GRANT-Berechtigung besitzen.
- Aktueller Berechtigungsschlüssel mit CREATE USER-Privileg und mit GRANT-Berechtigung::
  Alle Systemzugänge

Spaltenname	Datentyp	Inhalt
USER_CATALOG	CHAR (18)	Datenbankname
HOST_NAME	CHAR (8)	Rechnername oder *
APPLICATION_NAME	CHAR (8)	Anwendungsname oder * bei UTM-Systemeintrag Leerzeichen bei BS2000-Systemeintrag
SYSTEM_USER_NAME	CHAR (8)	BS2000- oder UTM-Benutzerkennung
USER_NAME	CHAR (18)	Berechtigungsschlüssel

Tabelle 96: View SYSTEM_ENTRIES des INFORMATION_SCHEMA

10.1.35 TABLES

Information über alle Basistabellen und Views, für die der aktuelle Berechtigungsschlüssel Privilegien besitzt.

Spaltenname	Datentyp	Inhalt
TABLE_CATALOG	CHAR (18)	Datenbankname
TABLE_SCHEMA	CHAR (31)	Name des Schemas, zu dem die Tabelle gehört
TABLE_NAME	CHAR (31)	Name der Basistabelle oder des View
TABLE_TYPE	VARCHAR (18)	BASE TABLE oder VIEW

Tabelle 97: View TABLES des INFORMATION_SCHEMA

10.1.36 TABLE_CONSTRAINTS

Information über Integritätsbedingungen der Schemata der Datenbank, die dem aktuellen Berechtigungsschlüssel gehören.

Spaltenname	Datentyp	Inhalt
CONSTRAINT_CATALOG	CHAR (18)	Datenbankname
CONSTRAINT_SCHEMA	CHAR (31)	Name des Schemas, zu dem die Tabelle mit der Integritätsbedingung gehört
CONSTRAINT_NAME	CHAR (31)	Name der Integritätsbedingung
TABLE_CATALOG	CHAR (18)	Datenbankname
TABLE_SCHEMA	CHAR (31)	Name des Schemas, zu dem die Tabelle gehört, auf die sich die Integritätsbedingung bezieht
TABLE_NAME	CHAR (31)	Name der Tabelle, auf die sich die Integritätsbedingung bezieht
CONSTRAINT_TYPE	VARCHAR (11)	Typ der Integritätsbedingung: FOREIGN KEY UNIQUE PRIMARY KEY CHECK
IS_DEFERRABLE	CHAR (3)	NO
INITIALLY_DEFERRED	CHAR (3)	NO

Tabelle 98: View TABLE_CONSTRAINTS des INFORMATION_SCHEMA

10.1.37 TABLE_PRIVILEGES

Information über alle Tabellen-Privileigen, die der aktuelle Berechtigungsschlüssel besitzt, oder die von ihm vergeben wurden.

Spaltenname	Datentyp	Inhalt
GRANTOR	CHAR (18)	Berechtigungsschlüssel, der das Privileg vergab, bzw. _SYSTEM
GRANTEE	CHAR (18)	Berechtigungsschlüssel, dem das Privileg verliehen wurde, oder PUBLIC
TABLE_CATALOG	CHAR (18)	Datenbankname
TABLE_SCHEMA	CHAR (31)	Name des Schemas, zu dem die Tabelle gehört, für die das Privileg gilt
TABLE_NAME	CHAR (31)	Name der Tabelle, für die das Privileg gilt
PRIVILEGE_TYPE	CHAR (18)	Privileg-Typ: SELECT INSERT DELETE UPDATE REFERENCES
IS_GRANTABLE	VARCHAR (3)	YES der Berechtigungsschlüssel besitzt GRANT-Berechtigung für das Privileg NO keine GRANT-Berechtigung

Tabelle 99: View TABLE_PRIVILEGES des INFORMATION_SCHEMA

10.1.38 TRANSLATIONS

Information über Transliterationen, die in der aktuellen DBH-Session durchführbar sind.

Spaltenname	Datentyp	Inhalt
TRANSLATION_CATALOG	CHAR (18)	Datenbankname
TRANSLATION_SCHEMA	CHAR (31)	INFORMATION_SCHEMA
TRANSLATION_NAME	CHAR (31)	CCS-Name
SOURCE_CHARACTER_SET_CATALOG	CHAR (18)	Datenbankname
SOURCE_CHARACTER_SET_SCHEMA	CHAR (31)	INFORMATION_SCHEMA
SOURCE_CHARACTER_SET_NAME	CHAR (8)	EBCDIC UTF16
TARGET_CHARACTER_SET_CATALOG	CHAR (18)	Datenbankname
TARGET_CHARACTER_SET_SCHEMA	CHAR (31)	INFORMATION_SCHEMA
TARGET_CHARACTER_SET_NAME	CHAR (8)	EBCDIC UTF16

Tabelle 100: View TRANSLATIONS des INFORMATION_SCHEMA

10.1.39 USAGE_PRIVILEGES

Information über alle USAGE-Privileigen, die der aktuelle Berechtigungsschlüssel besitzt oder die von ihm vergeben wurden.

Spaltenname	Datentyp	Inhalt
GRANTOR	CHAR (18)	Berechtigungsschlüssel, der das Privileg vergab, bzw. _SYSTEM
GRANTEE	CHAR (18)	Berechtigungsschlüssel, dem das Privileg verliehen wurde, oder PUBLIC
OBJECT_CATALOG	CHAR (18)	Datenbankname
OBJECT_SCHEMA	CHAR (31)	Name des Schemas, zu dem die Sortierreihenfolge bzw. der Zeichensatz gehört, für die das Privileg gilt Leerzeichen für Storage Group
OBJECT_NAME	CHAR (18)	Name der Storage Group, der Sortierreihenfolge bzw. des Zeichensatzes, für die das Privileg gilt
OBJECT_TYPE	CHAR (18)	Objekt, für das das Privileg gilt: STOGROUP CHARACTER SET COLLATION
PRIVILEGE_TYPE	CHAR (18)	USAGE
IS_GRANTABLE	VARCHAR (3)	YES der Berechtigungsschlüssel besitzt GRANT-Berechtigung für das Privileg NO keine GRANT-Berechtigung

Tabelle 101: View USAGE_PRIVILEGES des INFORMATION_SCHEMA

10.1.40 USERS

Information über Berechtigungsschlüssel.

- Aktueller Berechtigungsschlüssel ohne CREATE USER-Privileg:
  Der eigene Berechtigungsschlüssel
- Aktueller Berechtigungsschlüssel mit CREATE USER-Privileg, aber ohne GRANT-Berechtigung:
  Alle Berechtigungsschlüssel, die das CREATE USER-Privileg nicht besitzen oder ohne GRANT-Berechtigung besitzen.
- Aktueller Berechtigungsschlüssel mit CREATE USER-Privileg und mit GRANT-Berechtigung::
  Alle Berechtigungsschlüssel

Spaltenname	Datentyp	Inhalt
USER_CATALOG	CHAR (18)	Datenbankname
USER_NAME	CHAR (18)	Berechtigungsschlüssel

Tabelle 102: View USERS des INFORMATION_SCHEMA

10.1.41 VIEWS

Information über alle Views, für die der aktuelle Berechtigungsschlüssel Privilegien besitzt.

Spaltenname	Datentyp	Inhalt
TABLE_CATALOG	CHAR (18)	Datenbankname
TABLE_SCHEMA	CHAR (31)	Name des Schemas, zu dem der View gehört
TABLE_NAME	CHAR (31)	Name des View
VIEW_DEFINITION	VARCHAR (32000)	Abfrage-Ausdruck, der den View definiert, falls der aktuelle Berechtigungsschlüssel Schema Eigentümer ist NULL-Wert sonst
CHECK_OPTION	VARCHAR (8)	NONE keine Check-Option gesetzt CASCADED Check-Option gesetzt
IS_UPDATABLE	VARCHAR (3)	YES View ist änderbar NO View ist nicht änderbar

Tabelle 103: View VIEWS des INFORMATION_SCHEMA

10.1.42 VIEW_COLUMN_USAGE

Information über Views, die Spalten ansprechen, die dem aktuellen Berechtigungsschlüssel gehören, sowie die Namen der entsprechenden Spalten.

Spaltenname	Datentyp	Inhalt
VIEW_CATALOG	CHAR (18)	Datenbankname
VIEW_SCHEMA	CHAR (31)	Name des Schemas, zu dem der View gehört
VIEW_NAME	CHAR (31)	Name des View
TABLE_CATALOG	CHAR (18)	Datenbankname
TABLE_SCHEMA	CHAR (31)	Name des Schemas, zu dem die Tabelle gehört, auf die sich der View bezieht
TABLE_NAME	CHAR (31)	Name der Tabelle, auf die sich der View bezieht
COLUMN_NAME	CHAR (31)	Spaltenname

Tabelle 104: View VIEW_COLUMN_USAGE des INFORMATION_SCHEMA

10.1.43 VIEW_ROUTINE_USAGE

Information über die User Defined Functions (UDFs), die dem aktuellen Berechtigungsschlüssel gehören und in Views benutzt werden.

Spaltenname	Datentyp	Inhalt
TABLE_CATALOG	CHAR(18)	Datenbankname
TABLE_SCHEMA	CHAR(31)	Name des Schemas, zu dem der View gehört
TABLE_NAME	CHAR(31)	Name des Views
SPECIFIC_CATALOG	CHAR(18)	Datenbankname
SPECIFIC_SCHEMA	CHAR(31)	Name des Schemas, zu dem die Routine gehört
SPECIFIC_NAME	CHAR(31)	spezifischer Name der Routine

Tabelle 105: View VIEW_ROUTINE_USAGE des INFORMATION_SCHEMA

10.1.44 VIEW_TABLE_USAGE

Information über die Tabellen, die dem aktuellen Berechtigungsschlüssel gehören, und auf denen Views basieren.

Spaltenname	Datentyp	Inhalt
VIEW_CATALOG	CHAR (18)	Datenbankname
VIEW_SCHEMA	CHAR (31)	Name des Schemas, zu dem der View gehört
VIEW_NAME	CHAR (31)	Name des View
TABLE_CATALOG	CHAR (18)	Datenbankname
TABLE_SCHEMA	CHAR (31)	Name des Schemas, zu dem die Tabelle gehört, auf die sich der View bezieht
TABLE_NAME	CHAR (31)	Name der Tabelle, auf die sich der View bezieht

Tabelle 106: View VIEW_TABLE_USAGE des INFORMATION_SCHEMA

10.2 Views des SYS_INFO_SCHEMA

Das SYS_INFO_SCHEMA enthält systemspezifische Daten. Es informiert vollständig über alle Objekte von SESAM /SQL. Das SYS_INFO_SCHEMA kann in zukünftigen SESAM-Versionen inkompatibel verändert werden.

Nur der universelle Benutzer hat Zugriff auf die Views des SYS_INFO_SCHEMA. Der universelle Benutzer kann das SELECT-Privileg weitergeben.
Die folgende Tabelle zeigt, in welchen Views des SYS_INFO_SCHEMA Informationen über welche Datenbankobjekte enthalten sind.

Objekt	**View-Name**	**Information über**
Datenbank	SYS_CATALOGS SYS_DBC_ENTRIES	Datenbank alle bekannten Datenbanken
Schema	SYS_SCHEMATA	Schemata der Datenbank
Tabelle	SYS_TABLES SYS_PARTITIONS SYS_VIEW_USAGE SYS_CHECK_USAGE	Tabellen der Datenbank Partitionen der Datenbank Tabellen, auf denen Views basieren Tabellen, auf denen Check Bedingungen definiert sind
Spalte	SYS_COLUMNS SYS_VIEW_USAGE SYS_CHECK_USAGE	Spalten der Datenbank Spalten, auf denen Views basieren Spalten, auf denen Check Bedingungen definiert sind
Privileg	SYS_PRIVILEGES SYS_SPECIAL_PRIVILEGES SYS_USAGE_PRIVILEGES SYS_ROUTINE_PRIVILEGES	Tabellen-Privilegien Sonder-Privilegien USAGE-Privilegien Privilegien für Routinen
Index	SYS_INDEXES	Indizes der Datenbank
Integritätsbedingung	SYS_TABLE_CONSTRAINTS SYS_REFERENTIAL_CONSTRAINTS SYS_CHECK_CONSTRAINTS SYS_UNIQUE_CONSTRAINTS	Integritätsbedingungen Referenz-Bedingungen Check-Bedingungen Eindeutigkeitsbedingungen
Storage Group	SYS_STOGROUPS	Storage Groups der Datenbank
Space	SYS_SPACES SYS_SPACE_PROPERTIES	Spaces Space-Eigenschaften
Routinen	SYS_PARAMETERS SYS_ROUTINES SYS_ROUTINE_ROUTINE_USAGE SYS_ROUTINE_USAGE SYS_ROUTINE_ERRORS SYS_VIEW_ROUTINE_USAGE	Parameter von Routinen Routinen Routinen in anderen Routinen Tabellen und Spalten in Routinen Fehler-Ereignisse in Routinen Routinen in Views
SQL-Anweisungen	SYS_DML_RESOURCES	„teure“ DML-Anweisungen

Benutzer	SYS_USERS SYS_SYSTEM_ENTRIES	Berechtigungsschlüssel Systemeinträge
DA-LOG-Datei	SYS_DA_LOGS	DA-LOG-Dateien
Medientabelle	SYS_MEDIA_DESCRIPTIONS	Mediensätze der datenbankspezifischen Dateien
Recovery-Einheit	SYS_RECOVERY_UNITS	Recovery-Einheiten für Spaces
Sperren	SYS_LOCK_CONFLICTS	die zuletzt aufgetretenen Sperrkonflikte
Systemumgebung	SYS_ENVIRONMENT	die Betriebssystem-Umgebung von SESAM/SQL

Tabelle 107: Views des SYS_INFO_SCHEMA

Die Views des SYS_INFO_SCHEMA sind im Folgenden alphabetisch beschrieben.

10.2.1 SYS_CATALOGS

Information über die Datenbank.

Spaltenname	Datentyp	Inhalt
CHAR_FORM_OF_USE	CHAR (18)	Name des codierten Zeichensatzes (auch: Code-Tabelle) _NONE_, wenn kein codierter Zeichensatz verwendet wird.
UNIVERSAL_USER	CHAR (18)	Berechtigungsschlüssel des universellen Benutzers
LOGGING	VARCHAR (3)	Voreinstellung für den Parameter LOG: YES NO

Tabelle 108: View SYS_CATALOGS des SYS_INFO_SCHEMA

10.2.2 SYS_CHECK_CONSTRAINTS

Information über Check-Bedingungen.

Spaltenname	Datentyp	Inhalt
CONSTRAINT_SCHEMA	CHAR (31)	Name des Schemas, zu dem die Tabelle mit der Check-Bedingung gehört
CONSTRAINT_NAME	CHAR (31)	Name der Check-Bedingung
CHECK_CLAUSE	VARCHAR (32000)	Suchbedingung
CHECK_TYPE_IND	CHAR (1)	Y Check-Bedingung ist genau Nicht-NULL-Bedingung N sonst

Tabelle 109: View SYS_CHECK_CONSTRAINTS des SYS_INFO_SCHEMA

10.2.3 SYS_CHECK_USAGE

Information über Tabellen und Spalten der Check-Bedingung.

Spaltenname	Datentyp	Inhalt
CONSTRAINT_SCHEMA	CHAR (31)	Name des Schemas, zu dem die Check Bedingung gehört
CONSTRAINT_NAME	CHAR (31)	Name der Check-Bedingung
TABLE_SCHEMA	CHAR (31)	Name des Schemas, zu dem die Tabelle bzw. die Spalte gehört, die von der Check-Bedingung verwendet wird
TABLE_NAME	CHAR (31)	Name der Tabelle, die in der Check-Bedingung verwendet wird bzw. zu der die Spalte gehört
COLUMN_NAME	CHAR (31)	Name der Tabellenspalte, die in der Check-Bedingung verwendet wird Leerzeichen, falls Information über Tabelle
OBJECT_INDICATOR	CHAR (1)	T Satz enthält Information über Tabelle C Satz enthält Information über Spalte
NOT_NULL_COLUMN	CHAR (1)	Y Check-Bedingung erzwingt Nicht-NULL-Bedingung für die Spalte N sonst

Tabelle 110: View SYS_CHECK_USAGE des SYS_INFO_SCHEMA

10.2.4 SYS_COLUMNS

Information über Spalten von Basistabellen und Views der Datenbank.

Spaltenname	Datentyp	Inhalt
TABLE_SCHEMA	CHAR (31)	Name des Schemas, zu dem die Tabelle gehört
TABLE_NAME	CHAR (31)	Name der Basistabelle oder des View
COLUMN_NAME	CHAR (31)	Spaltenname
ORDINAL_POSITION	SMALLINT	laufende Nummer der Spalte in der Tabelle
COLUMN_DEFAULT	VARCHAR (256)	nur bei Basistabellen: voreingestellter Wert, wie er bei der Spaltendefinition angegeben wurde (z.B. CHAR-Literal in Hochkommas), falls der aktuelle Berechtigungsschlüssel Schema-Eigentümer ist TRUNCATED, wenn die Darstellung des voreingestellten Werts mehr als 256 Zeichen umfasst und der aktuelle Berechtigungsschlüssel Schema-Eigentümer ist. Der voreingestellte Wert kann nicht angezeigt werden. NULL-Wert sonst
IS_NULLABLE	VARCHAR (3)	NO Basistabellen-Spalte darf mit Sicherheit keinen NULL-Wert annehmen YES sonst
DATA_TYPE	VARCHAR (24)	Datentyp der Spalte: CHARACTER CHARACTER VARYING NATIONAL CHAR NATIONAL CHAR VARYING REAL DOUBLE PRECISION FLOAT INTEGER SMALLINT NUMERIC DECIMAL

		DATE TIME TIMESTAMP OLDEST
CHARACTER _MAXIMUM_LENGTH	SMALLINT	max. Länge der Spalte in Code Units, falls Datentyp CHARACTER, CHARACTER VARYING, NATIONAL CHAR, NATIONAL CHAR VARYING oder OLDEST NULL-Wert sonst
NUMERIC_PRECISION	SMALLINT	Gesamtstellenzahl, falls numerischer Datentyp NULL-Wert sonst
NUMERIC_PRECISION _RADIX	SMALLINT	Basis der Stellenzahl, falls numerischer Datentyp NULL-Wert sonst
NUMERIC_SCALE	SMALLINT	Nachkommastellenzahl, falls exakter numerischer Datentyp NULL-Wert sonst
DATETIME_PRECISION	SMALLINT	Nachkommastellenzahl, falls Datentyp TIME oder TIMESTAMP NULL-Wert sonst
Die Spalten OLDEST_DESCRIPTOR* sind belegt, falls DATA_TYPE ist OLDEST:		
OLDEST_DESCRIPTOR1	CHAR (1)	Y linksbündig N nicht linksbündig NULL-Wert, wenn DATATYPE nicht OL DEST
OLDEST_DESCRIPTOR2	CHAR (1)	Y Füllzeichen N keine Füllzeichen NULL-Wert, wenn DATATYPE nicht OL DEST
OLDEST_DESCRIPTOR3	CHAR (1)	Y Null (0) als Wert erlaubt N Null (0) nicht erlaubt NULL-Wert, wenn DATATYPE nicht OLDEST
OLDEST_DESCRIPTOR4	CHAR (1)	Y Wert hat arithmetisches Ergebnis

		N Wert hat kein arithmetisches Ergebnis NULL-Wert, wenn DATATYPE nicht OLDEST
COLUMN_DESCRIPTOR1	CHAR (1)	Y Basistabellen-Spalte besitzt genau einen einfachen Index und kommt nicht in zusammengesetztem Index vor N sonst
COLUMN_DESCRIPTOR2	CHAR (1)	Y Basistabellen-Spalte besitzt genau einen zusammengesetzten Index und keinen einfachen Index N sonst
COLUMN_DESCRIPTOR3	CHAR (1)	Y Basistabellen-Spalte besitzt mehr als einen Index N sonst
COLUMN_DESCRIPTOR4	CHAR (1)	Y Basistabellen-Spalte besitzt eine CALL-DML-Voreinstellung N sonst
COLUMN_DESCRIPTOR5	CHAR (1)	Y Basistabellen-Spalte ist eine multiple Spalte N sonst
PK_DISTANCE	SMALLINT	Abstand der Spalte zum Anfang des Primärschlüssels NULL-Wert, falls Spalte nicht im Primärschlüssel oder keine Basistabellen-Spalte
SESAM_SAN	CHAR (3)	Symbolischer Attributname der Spalte NULL-Wert, falls Spalte nicht in Basistabelle definiert oder in SQL-Tabelle
SESAM_BAN	CHAR (2)	Binärer Attributname der Spalte NULL-Wert, falls Spalte nicht in Basistabelle definiert
SESAM_DEFAULT	CHAR (2)	CALL-DML-Voreinstellung (mit Vorzeichen, falls numerischer Datentyp) NULL-Wert, falls Spalte nicht in Basistabelle definiert oder in SQL-Tabelle
FIRST_OCCURRENCE	SMALLINT	erste mögliche Ausprägung einer multiplen Spalte (bei Basistabelle = 1) NULL-Wert, falls Spalte nicht multipel
LAST_OCCURRENCE	SMALLINT	letzte mögliche Ausprägung einer multiplen Spalte NULL-Wert, falls Spalte nicht multipel

Tabelle 111: View SYS_COLUMNS des SYS_INFO_SCHEMA

10.2.5 SYS_DA_LOGS

Information über DA-LOG-Dateien bzw. DA-LOG-Einheiten einer Datenbank.

Spaltenname	Datentyp	Inhalt
DALOG_VERSION	INTEGER	Versionsnummer der DA-LOG-Datei
DALOG_SUBNUMBER	INTEGER	laufende Nummer der DA-LOG-Datei innerhalb der Version
DALOG_BLOCKNUMBER	INTEGER	erster Block in der DA-LOG-Datei für diese DA-LOG-Einheit
DALOG_INIT	TIMESTAMP (3)	Erstellungszeitpunkt
BLOCK_COUNTER	INTEGER	letzter benutzter Block in der DA-LOG-Datei
MAX_USER	INTEGER	max. Anzahl paralleler Benutzer in der korrespondierenden DBH-Session
SYSTEM_DATA_BUFFER	INTEGER	Originalgröße des System Data Buffers beim Schreiben der DA-LOG-Datei
USER_DATA_BUFFER	INTEGER	Originalgröße des User Data Buffers beim Schreiben der DA-LOG-Datei

Tabelle 112: View SYS_DA_LOGS des SYS_INFO_SCHEMA

10.2.6 SYS_DBC_ENTRIES

Information über alle Datenbanken, die dem DBH bekannt sind.

Spaltenname	Datentyp	Inhalt
DBC_NUMBER	SMALLINT	DBC-Identifikationsnummer
CATALOG_NAME	CHAR (18)	Logischer Datenbankname
PHYSICAL_NAME	CHAR (18)	Physikalischer Datenbankname
USER_ID	CHAR (8)	DB-Kennung der Datenbank
COPY_NUMBER	CHAR (6)	Versionsnummer des SESAM-Sicherungsbestandes des Catalog-Space, falls es sich bei der Datenbank um einen SESAM-Sicherungsbestand handelt.
ACCESS_MODE	VARCHAR (5)	Aktueller Zugriffsmodus: READ Lesender Zugriff auf Anwenderdaten und Metadaten ist erlaubt. WRITE Lesender und ändernder Zugriff auf Anwenderdaten ist erlaubt. Metadaten dürfen nicht geändert werden. ADMIN Lesender und ändernder Zugriff auf Anwender- und Metadaten ist erlaubt. REPL Es handelt sich um ein Replikat. Auf dieses Replikat kann lesend zugegriffen werden. COPY Lesender Zugriff auf Anwenderdaten und Metadaten ist erlaubt. Die Utility-Anweisung COPY ist erlaubt.
STATUS	VARCHAR (7)	Status der Datenbank: ACTIVE Die Datenbank wurde in der laufenden DBH-Session geöffnet. CLOSED Die Datenbank ist geschlossen. FREE Die Datenbank ist physikalisch geschlossen und freigegeben. LOCKED

		Die Datenbank ist auf Grund eines SQLSTATE in der laufenden DBH-Session nicht verfügbar. RECOVER Die Datenbank befindet sich im Zustand RECOVER. REORG Die Datenbank wird reorganisiert. REFRESH Die Datenbank befindet sich im Zustand REFRESH.
STATUS_INFO	VARCHAR (21)	Informationen darüber, warum die Datenbank nicht verfügbar ist (nur bei STATUS = LOCKED).
STATUS_TIME	TIMESTAMP (3)	Zeitpunkt der Feststellung des aktuellen Status

Tabelle 113: View SYS_DBC_ENTRIES des INFORMATION_SCHEMA

10.2.7 SYS_DML_RESOURCES

Information über „teure“ DML-Anweisungen (in SQL). Eine DML-Anweisung gilt als teuer, wenn die Anzahl der von ihr ausgelösten logischen IOs und/oder ihre Aktivitätszeit im DBH sehr hoch ist im Vergleich zu anderen DML-Anweisungen.

Insbesondere die Spalten NUMBER_OF_LOGICAL_IO und ACTIVE_TIME beinhalten relevante Informationen zu den Kosten einer Anweisung.

Spaltenname	Datentyp	Inhalt
CATALOG_NAME	CHAR (18)	Datenbankname
START_TIME	TIMESTAMP (3)	Startzeitpunkt der DML-Anweisung
END_TIME	TIMESTAMP (3)	Endzeitpunkt der DML-Anweisung
HOST_NAME	CHAR (8)	Rechnername aus der Identifikation des Auftraggebers
APPLICATION_NAME	CHAR (8)	Anwendungsname aus der Identifikation des Auftraggebers
CUSTOMER_NAME	CHAR (8)	Name des Auftraggebers aus der Identifikation des Auftraggebers
CONVERSATION_ID	CHAR (8)	Identifikation des Auftraggebers bezüglich UTM und SESAM-DBAccess
TAC_NAME	CHAR (8)	Jobname der Benutzerkennung oder Name des Teilprogramms, das die DML-Anweisung ausgeführt hat
MODULE_NAME	CHAR (8)	Name der Übersetzungseinheit, in der die DML-Anweisung ausgeführt wurde
STATEMENT_NAME	VARCHAR (18)	Interner Name der DML-Anweisung
STATEMENT_TYPE	VARCHAR (31)	<Typ der Anweisung> (z.B. INSERT)
NUMBER_OF_LOGICAL_IO	INTEGER	Anzahl logischer Lese- und Schreibzugriffe
NUMBER_OF_PHYSICAL_IO	INTEGER	Anzahl physikalischer Lese- und Schreibzugriffe
ELAPSED_TIME	INTEGER	real abgelaufene Zeit (Millisekunden)
ACTIVE_TIME	INTEGER	Aktivitätszeit im DBH (Millisekunden)
ACTIVE_TIME_DBH	INTEGER	Aktivitätszeit in DBH-Tasks (Millisekunden)
ACTIVE_TIME_SVT	INTEGER	Aktivitätszeit in Service-Tasks (Millisekunden)
MEASURE_OF_COSTS	INTEGER	Interne Maßzahl für die Kosten der Anweisung

Tabelle 114: View SYS_DML_RESOURCES des SYS_INFO_SCHEMA

10.2.8 SYS_ENVIRONMENT

Informationen über die Betriebssystem-Umgebung von SESAM/SQL. Wird zu Wartungszwecken erstellt, speziell nach einer Live Migration.

Spaltenname	Datentyp	Inhalt
INFO_TIMESTAMP	TIMESTAMP (3)	Zeitpunkt der Information (nach einer Live Migration ist dies der Zeitpunkt der Live Migration, sonst ein Zeitpunkt in der Initialisierungsphase des DBH)
HW_TYPE	CHAR (8)	Hardware-Typ des aktuellen Systems
OS_VERSION	CHAR (12)	Name und Version des BS2000-Betriebssystems
MAIN_MEMORY	INTEGER	Größe des BS2000-Hauptspeichers in MByte
NUMBER_OF_CPU_MAX	INTEGER	Maximalzahl der BS2000-CPUs
NUMBER_OF_CPU_ACTIVE	INTEGER	Anzahl der aktiven BS2000-CPUs
HOST_NAME	CHAR (8)	Rechnername

Tabelle 115: View SYS_ENVIRONMENT des SYS_INFO_SCHEMA

10.2.9 SYS_INDEXES

Informationen über Indizes der Datenbank, die mit CREATE INDEX oder implizit durch eine Eindeutigkeitsbedingung (UNIQUE) erzeugt wurden.

Spaltenname	Datentyp	Inhalt
INDEX_SCHEMA	CHAR (31)	Name des Schemas, zu dem der Index gehört
TABLE_NAME	CHAR (31)	Name der Basistabelle, zu der der Index gehört
COLUMN_NAME	CHAR (31)	Name der Spalte des Index
INDEX_NAME	CHAR (18)	Name des Index
INDEX_ID	SMALLINT	Identifikationsnummer des Index
SPACE_NAME	CHAR (18)	Name des Space, in dem der Index gespeichert ist
SPACE_ID	SMALLINT	Identifikationsnummer des Space, in dem der Index gespeichert ist
ORDINAL_POSITION	SMALLINT	Position der Spalte im Index
LENGTH_I	SMALLINT	Gesamtlänge des Index (in Bytes)
LENGTH_C	SMALLINT	gibt an, bis zu welcher Länge (in Bytes) die Spalte in den Index einbezogen ist
INDEX_DISTANCE	SMALLINT	Abstand der Spalte zum Indexanfang
DATA_TYPE_C	VARCHAR (24)	Datentyp der Spalte: CHARACTER CHARACTER VARYING NATIONAL CHAR NATIONAL CHAR VARYING REAL DOUBLE PRECISION FLOAT INTEGER SMALLINT NUMERIC DECIMAL DATE TIME TIMESTAMP OLDEST
CONSTRAINT_NAME	CHAR (31)	Name der Eindeutigkeitsbedingung, wenn der Index von einer Eindeutigkeitsbedingung verwendet wird NULL-Wert sonst
STATE	VARCHAR (9)	Status: GENERATED DEFECT

GENERATE_TYPE	VARCHAR (8)	wie erzeugt: EXPLICIT IMPLICIT
STATISTICS_INFO	VARCHAR (3)	YES Statistik-Information vorhanden NO nicht vorhanden
INDEX_TYPE	VARCHAR (8)	Index-Typ: SINGLE COMPOUND
INDEX_DATE	TIMESTAMP (3)	Erzeugungszeitpunkt
INDEX_PRIMARY_KEY	CHAR (1)	Y Index wird für Compound Key von CALL-DML-Tabellen verwendet N sonst
TABLE_ID	SMALLINT	Identifikationsnummer der Basistabelle. Bei TABLE_ID >= 30720 liegt eine partitionierte Tabelle vor.

Tabelle 116: View SYS_INDEXES des SYS_INFO_SCHEMA

10.2.10 SYS_LOCK_CONFLICTS

Information über die zuletzt aufgetretenen Sperrkonflikte (in zeitlicher Ordnung).

Spaltenname	Datentyp	Inhalt
TIME_OF_CONFLICT	TIMESTAMP (3)	Zeitpunkt, zu dem der Konflikt aufgetreten ist
OBJECT_TYPE	VARCHAR (6)	Typ des zu sperrenden Objekts: DBC Datenbank-Katalog SPACE Space TABLE Basistabelle INDEX Index ROW Satz einer Basistabelle SI-VAL Wert eines Sekundärindex PLAN SQL-Plan META Metadaten-Bereich
DBC_NUMBER	SMALLINT	Identifikationsnummer der Datenbank des zu sperrenden Objekts (für OBJECT_TYPE ungleich PLAN) NULL-Wert sonst
SPACE_ID	SMALLINT	Identifikationsnummer des Space des zu sperrenden Objekts (für OBJECT_TYPE = SPACE / TABLE / INDEX / ROW / SI-VAL) NULL-Wert sonst
TABLE_ID	SMALLINT	Identifikationsnummer der Basistabelle des zu sperrenden Objekts (für OBJECT_TYPE = TABLE / ROW) NULL-Wert sonst
INDEX_ID	SMALLINT	Identifikationsnummer des Index des zu sperrenden Objekts (für OBJECT_TYPE = INDEX / SI-VAL) NULL-Wert sonst
ROW_ID	CHAR (8)	Interne Nummer des zu sperrenden Satzes (für OBJECT_TYPE = ROW) NULL-Wert sonst
SI_VALUE	CHAR (8)	Interne Darstellung des zu sperrenden Schlüsselwertes (für OBJECT_TYPE = SI-VAL) NULL-Wert sonst
PLAN_ID	INTEGER	

		Interne Nummer des zu sperrenden SQL-Plans (für OBJECT_TYPE = PLAN) NULL-Wert sonst
META_SCHEMA	CHAR (8)	Interne Nummer des zu sperrenden Schemas im Metadatenbereich (für OBJECT_TYPE = META) NULL-Wert sonst
META_TABLE	CHAR (8)	Interne Nummer der zu sperrenden Basistabelle im Metadatenbereich (für OBJECT_TYPE = META) NULL-Wert sonst
HOST_NAME	CHAR (8)	Rechnername aus der Identifikation des wartenden Auftraggebers
APPLICATION_NAME	CHAR (8)	Anwendungsname aus der Identifikation des wartenden Auftraggebers
CUSTOMER_NAME	CHAR (8)	Name des Auftraggebers aus der Identifikation des wartenden Auftraggebers
CONVERSATION_ID	CHAR (8)	Identifikation des wartenden Auftraggebers bezüglich UTM und SESAM-DBAccess
TAC_NAME	CHAR (8)	Jobname der Benutzerkennung oder Name des Teilprogramms, das die Sperre angefordert hat
MODULE_NAME	CHAR (8)	Name der Übersetzungseinheit (nur SQL), in der die wartende SQL-Anweisung ausgeführt wurde NULL-Wert sonst
STATEMENT_NAME	VARCHAR (18)	Interner Name des SQL-Statements, das auf die Sperre wartet NULL-Wert sonst
STATEMENT_TYPE	VARCHAR (31)	Typ der Anweisung (z.B. INSERT) bei SQL-Anweisungen CALL DML: <Operationscode> bei CALL-DML-Anweisungen SYSTEM bei Systemaufträgen (z.B. Administrationskommandos über SEND-MSG)
LOCK_MODE	VARCHAR (31)	Ebene der Sperranforderung: Für OBJECT_TYPE = SPACE: NO-UPDATE/SHARED-READ, SHARED-UPDATE/SHARED-READ,

		EXCLUSIVE-UPDATE/SHARED-READ, EXCLUSIVE-UPDATE/EXCLUSIVE-READ Sonst: SHARED, EXCLUSIVE
LOCK_TYPE	VARCHAR (8)	Wert der Sperranforderung: OBJECT für Objektsperre ADJACENT für Umgebungssperre
REQUEST_ANNOUNCED	CHAR (1)	Sperranforderung soll angemeldet werden: Y N
LOCKING_OBJECT_TYPE	VARCHAR (6)	Typ des Objekts, das die Sperre verhindert: DBC Datenbank-Katalog SPACE Space TABLE Basistabelle INDEX Index ROW Satz einer Basistabelle SI-VAL Wert eines Sekundärindex PLAN SQL-Plan META Metadaten-Bereich
LOCKING_HOST_NAME	CHAR (8)	Rechnername aus der Identifikation des sperrenden Auftraggebers
LOCKING_APPLICATION_NAME	CHAR (8)	Anwendungsname aus der Identifikation des sperrenden Auftraggebers
LOCKING_CUSTOMER_NAME	CHAR (8)	Name des Auftraggebers aus der Identifikation des sperrenden Auftraggebers
LOCKING_CONVERSATION_ID	CHAR (8)	Identifikation des sperrenden Auftraggebers bezüglich UTM und SESAM-DBAccess
LOCKING_LOCK_MODE	VARCHAR (31)	Ebene des Objekts, an dem die Sperre scheitert: NO-UPDATE/SHARED-READ, SHARED-UPDATE/SHARED-READ, EXCLUSIVE-UPDATE/SHARED-READ, EXCLUSIVE-UPDATE/EXCLUSIVE-READ für OBJECT_TYPE = SPACE SHARED, EXCLUSIVE sonst

Tabelle 117: View SYS_LOCK_CONFLICTS des SYS_INFO_SCHEMA

10.2.11 SYS_MEDIA_DESCRIPTIONS

Information über Dateieigenschaften und Datenträgertypen für datenbankspezifische Dateien.

Spaltenname	Datentyp	Inhalt
FILE_TYPE	CHAR (6)	Dateityp: DALOG CATLOG PBI CATREC DDLTA
DEVICE_DESCRIPTOR	CHAR (18)	Gerätetyp oder Name der Storage Group für die Datei
MEDIUM	CHAR (4)	DISC
ORDINAL_POSITION	SMALLINT	laufende Nummer des Eintrags in der Medientabelle
REQUESTS	VARCHAR (3)	YES Datenträgeranforderung an der Konsole möglich NO keine Datenträgeranforderung an der Konsole
PRIMARY_ALLOC	INTEGER	Primärzuweisung
SECONDARY_ALLOC	INTEGER	Sekundärzuweisung
SHARABLE	VARCHAR (3)	Datei mehrfach benutzbar: YES NO

Tabelle 118: View SYS_MEDIA_DESCRIPTIONS des SYS_INFO_SCHEMA

10.2.12 SYS_PARAMETERS

Information über Parameter von Routinen (Prozeduren und UDFs).

Spaltenname	Datentyp	Inhalt
SPECIFIC_SCHEMA	CHAR(31)	Name des Schemas, zu dem die Routine gehört
SPECIFIC_NAME	CHAR(31)	Spezifischer Name der Routine
ORDINAL_POSITION	SMALLINT	Laufende Nummer des Parameters in der Routine
PARAMETER_MODE	VARCHAR(5)	IN Eingabeparameter OUT Ausgabeparameter INOUT Ein- und Ausgabeparameter
PARAMETER_NAME	CHAR(31)	Name des Parameters
DATA_TYPE	VARCHAR(24)	Datentyp der Spalte: CHARACTER CHARACTER VARYING NATIONAL CHAR NATIONAL CHAR VARYING REAL DOUBLE PRECISION FLOAT INTEGER SMALLINT NUMERIC DECIMAL DATE TIME TIMESTAMP
CHARACTER_ MAXIMUM_LENGTH	SMALLINT	max. Länge der Spalte in Code Units, falls Datentyp CHARACTER, CHARACTER VARYING, NATIONAL CHAR oder NATIONAL CHAR VARYING NULL-Wert sonst
NUMERIC_PRECISION	SMALLINT	Gesamtstellenzahl, falls numerischer Datentyp NULL-Wert sonst
NUMERIC_PRECISI- ON_RADIX	SMALLINT	Basis der Stellenzahl, falls numerischer Datentyp NULL-Wert sonst
NUMERIC_SCALE	SMALLINT	Nachkommastellenzahl,

		falls exakter numerischer Datentyp NULL-Wert sonst
DATETIME_PRECISION	SMALLINT	Nachkommastellenzahl, falls Datentyp TIME oder TIMESTAMP NULL-Wert sonst

Tabelle 119: View SYS_PARAMETERS des SYS_INFO_SCHEMA

10.2.13 SYS_PARTITIONS

Information über Partitionen von Basistabellen.

Spaltenname	Datentyp	Inhalt
TABLE_SCHEMA	CHAR (31)	Name des Schemas, zu dem die partitionierte Tabelle gehört
TABLE_NAME	CHAR (31)	Name der partitionierten Tabelle
SERIAL_NUMBER	SMALLINT	Laufende Nummer der Partition
MAX_KEY_VALUE	VARCHAR(32000)	Vergleich für die obere Partitionsgrenze, wie in der VALUE-Klausel angegeben (externe Darstellung)
MAX_NUMBER_OF_ROWS	INTEGER	Maximal mögliche Anzahl der Sätze in der Partition
SPACE_NAME	CHAR (18)	Name des Space, in dem die Partition gespeichert ist
SPACE_ID	SMALLINT	Identifikationsnummer des Space, in dem die Partition gespeichert ist
TABLE_ID	SMALLINT	Space-bezogene Identifikationsnummer der partitionierten Tabelle
ROW_ID_PREFIX	SMALLINT	Präfix zur Bestimmung der Satznummer

Tabelle 120: View SYS_PARTITIONS des SYS_INFO_SCHEMA

10.2.14 SYS_PRIVILEGES

Information über Tabellen- und Spalten-Privilegien.

Spaltenname	Datentyp	Inhalt
GRANTEE	CHAR (18)	Berechtigungsschlüssel, dem das Privileg verliehen wurde, oder PUBLIC
TABLE_SCHEMA	CHAR (31)	Name des Schemas, zu dem die Tabelle bzw. die Spalte gehört, für die das Privileg gilt
TABLE_NAME	CHAR (31)	Name der Tabelle, für die das Privileg gilt bzw. für deren Spalte das Privileg gilt
COLUMN_NAME	CHAR (31)	Name der Spalte, auf die das Privileg eingeschränkt wurde Leerzeichen, wenn das Privileg für die gesamte Tabelle gilt
OBJECT_INDICATOR	CHAR (1)	T Satz enthält Information über Tabelle C Satz enthält Information über Spalte
PRIVILEGE_TYPE	CHAR (18)	Privileg-Typ: SELECT INSERT DELETE UPDATE REFERENCES
GRANTOR	CHAR (18)	Berechtigungsschlüssel, der das Privileg vergab, bzw. _SYSTEM
IS_GRANTABLE	VARCHAR (3)	YES der Berechtigungsschlüssel besitzt GRANT-Berechtigung für das Privileg NO keine GRANT-Berechtigung

Tabelle 121: View SYS_PRIVILEGES des SYS_INFO_SCHEMA

10.2.15 SYS_RECOVERY_UNITS

Information über Recovery-Einheiten.

Spaltenname	Datentyp	Inhalt
SPACE_NAME	CHAR (18)	Name des Space
RECOVERY_TIMESTAMP	TIMESTAMP (3)	Zeitpunkt der Erzeugung der Sicherung
VERSION	INTEGER	interne Nummer, falls RECOVERY_TYPE = 'COPY' NULL-Wert sonst
VALIDITY	VARCHAR (3)	YES Recovery-Einheit gültig für Reparatur zur nächsten Recovery Einheit NO ungültig (kann sich durch eine RECOVER-Anweisung aber eventuell zu YES ändern) NOT ungültig (kann sich nicht mehr ändern)
RECOVERY_UNIT_NAME	VARCHAR (54)	Dateiname der Kopie, falls RECOVERY_TYPE = 'COPY' interne Nummer, falls RECOVERY_TYPE = 'RESTART' oder 'REST_TO' NULL-Wert sonst
ARCHIVE_COPY_VERSION	VARCHAR (15)	Zeitpunkt der ARCHIVE-Sicherung, falls MEDIUM = 'TAPE' Zeitpunkt der HSMS-Sicherung, falls MEDIUM = 'HSMS', 'HSMW' oder 'HSMB' NULL-Wert sonst
MEDIUM	CHAR(4)	DISC SESAM-Sicherung auf Platte TAPE SESAM-Sicherung mit ARCHIVE HSMW SESAM-Sicherung mit HSMS (Arbeitsdatei) HSMB SESAM-Sicherung mit HSMS (Additional-Mirror-Unit) SRDF SESAM-Sicherung mit HSMS (SRDF-Target) falls RECOVERY_TYPE = 'COPY' NULL-Wert sonst

RECOVERY_TYPE	VARCHAR (7)	Werte, die von der Utility Recovery ausgewertet werden: COPY CREATE RESTART REST_TO (RESTART TO) MARK	
COPY_TYPE	VARCHAR (7)	ONLINE oder OFFLINE, falls RECOVERY_TYPE = 'COPY' NULL-Wert sonst	
DALOG_VERSION	INTEGER	Versionsnummer der DA-LOG-Datei	DA-LOG-Pegel vor dem Eintrag der Recovery Einheit
DALOG_SUBNUMBER	INTEGER	laufende Nummer der DA-LOG-Datei innerhalb der Version	
DALOG_BLOCKNUMBER	INTEGER	erster Block in der DA-LOG-Datei für diese DA-LOG-Einheit	
NEXT_DALOG_VERSION	INTEGER	Versionsnummer der DA-LOG-Datei	DA-LOG-Pegel nach dem Eintrag der Recovery-Einheit
NEXT_DALOG_SUBNUMBER	INTEGER	laufende Nummer der DA-LOG-Datei innerhalb der Version	
NEXT_DALOG_BLOCKNUMBER	INTEGER	erster Block in der DA-LOG-Datei für diese DA-LOG-Einheit	
LOG_COUNTER	INTEGER	zur Zeit nicht genutzt	
ARCHIVE_DIRECTORY_NAME	VARCHAR (54)	Name des ARCHIVE-Directory, falls MEDIUM = 'TAPE' Name des HSMS-Archivs, falls MEDIUM = 'HSMS', 'HSMW', 'HSMB' oder 'SRDF' NULL-Wert sonst	
ARCHIVE_PBI_VERSION	VARCHAR (15)	Zeitpunkt der ARCHIVE-Sicherung der PBI-Datei, falls MEDIUM = 'TAPE' und COPY_TYPE = 'ONLINE' NULL-Wert sonst	
PBI_TIMESTAMP	TIMESTAMP (3)	Erzeugungszeitpunkt der PBI-Datei	
PBI_COUNTER	INTEGER	undefiniert	

Tabelle 122: View SYS_RECOVERY_UNITS des SYS_INFO_SCHEMA

10.2.16 SYS_REFERENTIAL_CONSTRAINTS

Information über Referenzbedingungen. Referenzierende und referenzierte Spalte werden aufgeführt.

Spaltenname	Datentyp	Inhalt
CONSTRAINT_SCHEMA	CHAR (31)	Name des Schemas, zu dem die Tabelle mit der Referenzbedingung gehört
CONSTRAINT_NAME	CHAR (31)	Name der Referenzbedingung
TABLE_NAME	CHAR (31)	Name der Tabelle, zu der die Referenzbedingung gehört
COLUMN_NAME	CHAR (31)	Name einer referenzierenden Spalte
UNIQUE_CONSTRAINT_SCHEMA	CHAR (31)	Name des Schemas der referenzierten Tabelle
UNIQUE_CONSTRAINT_NAME	CHAR (31)	Name der Eindeutigkeits- bzw. Primärschlüsselbedingung der referenzierten Tabelle
UNIQUE_CONSTRAINT_TABLE	CHAR (31)	Name der referenzierten Tabelle
UNIQUE_CONSTRAINT_COLUMN	CHAR (31)	Name einer referenzierten Spalte
ORDINAL_POSITION	SMALLINT	Position der Spalte in der Referenzbedingung

Tabelle 123: View SYS_REFERENTIAL_CONSTRAINTS des SYS_INFO_SCHEMA

10.2.17 SYS_ROUTINES

Information über Routinen (Prozeduren und UDFs).

Spaltenname	Datentyp	Inhalt
SPECIFIC_SCHEMA	CHAR(31)	Name des Schemas, zu dem die Routine gehört
SPECIFIC_NAME	CHAR(31)	spezifischer Name der Routine
ROUTINE_TYPE	VARCHAR(28)	PROCEDURE, falls Prozedur FUNCTION, falls UDF
DATA_TYPE	VARCHAR(24)	Datentyp des Rückgabewerts einer UDF CHARACTER CHARACTER VARYING NATIONAL CHAR NATIONAL CHAR VARYING REAL DOUBLE PRECISION FLOAT INTEGER SMALLINT NUMERIC DECIMAL DATE TIME TIMESTAMP NULL-Wert, falls Prozedur
CHARACTER_MAXIMUM_LENGTH	SMALLINT	max. Länge des Rückgabewerts in Code Units, falls Datentyp CHARACTER, CHARACTER VARYING, NATIONAL CHAR oder NATIONAL CHAR VARYING NULL-Wert sonst
NUMERIC_PRECISION	SMALLINT	Gesamtstellenzahl, falls numerischer Datentyp NULL-Wert sonst
NUMERIC_PRECISION_RADIX	SMALLINT	Basis der Stellenzahl, falls numerischer Datentyp

		NULL-Wert sonst
NUMERIC_SCALE	SMALLINT	Nachkommastellenzahl, falls exakter numerischer Datentyp NULL-Wert sonst
DATETIME_PRECISION	SMALLINT	Nachkommastellenzahl, falls Datentyp TIME oder TIMESTAMP NULL-Wert sonst
ROUTINE_DEFINITION	VARCHAR(32000)	Text der Routine
SQL_DATA_ACCESS	VARCHAR(17)	CONTAINS SQL falls bei der Definition der Routine CONTAINS SQL angegeben wurde READS SQL DATA falls bei der Definition der Routine READS SQL DATA angegeben wurde MODIFIES SQL DATA falls bei der Definition der Routine MODIFIES SQL DATA angegeben wurde
IS_NULL_CALL	VARCHAR(3)	NO, falls UDF NULL-Wert sonst
IS_USER_DEFINED_CAST	VARCHAR(3)	NO, falls UDF NULL-Wert sonst
AS_LOCATOR	VARCHAR(3)	NO, falls UDF NULL-Wert sonst

Tabelle 124: View SYS_ROUTINES des SYS_INFO_SCHEMA

10.2.18 SYS_ROUTINE_ERRORS

Information über die zuletzt aufgetretenen fehlerhaften oder fehlerverdächtigen Ereignisse beim Ablauf von Routinen (Prozeduren und UDFs). Auch das Pragma DEBUG ROUTINE kann zur Protokollierung zusätzlicher Informationen führen.

Spaltenname	Datentyp	Inhalt
SPECIFIC_CATALOG	CHAR(18)	Name der Datenbank, zu der die Routine gehört
SPECIFIC_SCHEMA	CHAR(31)	Name des Schemas, zu dem die Routine gehört
SPECIFIC_NAME	CHAR(31)	spezifischer Name der Routine
START_TIME	TIMESTAMP(3)	Startzeitpunkt der Routine
ERROR_TIME	TIMESTAMP(3)	Zeitpunkt des Fehler-Ereignisses
ERROR_STATE	CHAR(5)	SQLSTATE, falls eine exception condition auftrat Leerzeichen sonst
ERROR_TEXT	VARCHAR(256)	Meldungstext
LINE_NUMBER	INTEGER	Zeilennummer der fehlerhaften Anweisung im Text der Routine 0 wenn die Stelle nicht bestimmt werden konnte
COLUMN_NUMBER	INTEGER	Spaltennummer der fehlerhaften Anweisung im Text der Routine 0 wenn die Stelle nicht bestimmt werden konnte
HOST_NAME	CHAR(8)	Rechnername aus der Identifikation des Auftraggebers
APPLICATION_NAME	CHAR(8)	Anwendungsname aus der Identifikation des Auftraggebers
CUSTOMER_NAME	CHAR(8)	Name des Auftraggebers aus der Identifikation des Auftraggebers
CONVERSATION_ID	CHAR(8)	Identifikation des Auftraggebers bezüglich UTM und SESAM-DBAccess
TAC_NAME	CHAR(8)	Jobname der Benutzerkennung oder Name des Teilprogramms, das die Routine aufgerufen hat

MODULE_NAME	CHAR(8)	Name der Übersetzungseinheit, in der die Routine aufgerufen wurde
STATEMENT_NAME	VARCHAR(18)	Interner Name der SQL-Anweisung, die die Routine aufgerufen hat

Tabelle 125: View SYS_ROUTINE_ERRORS des SYS_INFO_SCHEMA

10.2.19 SYS_ROUTINE_PRIVILEGES

Information über die Privilegien für Routinen (Prozeduren und UDFs).

Spaltenname	Datentyp	Inhalt
GRANTEE	CHAR(18)	Berechtigungsschlüssel, dem das Privileg verliehen wurde, oder PUBLIC
SPECIFIC_SCHEMA	CHAR(31)	Name des Schemas, zu dem die Routine gehört
SPECIFIC_NAME	CHAR(31)	spezifischer Name der Routine
GRANTOR	CHAR(18)	Berechtigungsschlüssel, der das Privileg vergab, oder _SYSTEM
IS_GRANTABLE	VARCHAR(3)	YES der Berechtigungsschlüssel besitzt GRANT-Berechtigung für das Privileg NO keine GRANT-Berechtigung

Tabelle 126: View SYS_ROUTINE_PRIVILEGES des SYS_INFO_SCHEMA

10.2.20 SYS_ROUTINE_ROUTINE_USAGE

Information über die Routinen (Prozeduren und UDFs), die in anderen Routinen aufgerufen werden.

Spaltenname	Datentyp	Inhalt
SPECIFIC_SCHEMA	CHAR(31)	Name des Schemas, zu dem die aufrufende Routine gehört
SPECIFIC_NAME	CHAR(31)	spezifischer Name der aufrufenden Routine
ROUTINE_SCHEMA	CHAR(31)	Name des Schemas, zu dem die aufgerufene Routine gehört
ROUTINE_NAME	CHAR(31)	spezifischer Name der aufgerufenen Routine

Tabelle 127: View SYS_ROUTINE_ROUTINE_USAGE des SYS_INFO_SCHEMA

10.2.21 SYS_ROUTINE_USAGE

Information über die Tabellen und Spalten, die in Routinen (Prozeduren und UDFs) angesprochen werden.

Spaltenname	Datentyp	Inhalt
SPECIFIC_SCHEMA	CHAR(31)	Name des Schemas, zu dem die Routine gehört
SPECIFIC_NAME	CHAR(31)	spezifischer Name der Routine
TABLE_SCHEMA	CHAR(31)	Name des Schemas, zu dem die Tabelle gehört, die in der Routine angesprochen wird
TABLE_NAME	CHAR(31)	Name der Tabelle, die in der Routine verwendet wird
COLUMN_NAME	CHAR(31)	Spaltenname, in der Routine verwendet Leerzeichen, falls Informationen über Tabelle
OBJECT_INDICATOR	CHAR(1)	T Satz enthält Informationen über Tabelle C Satz enthält Informationen über Spalte

Tabelle 128: View SYS_ROUTINE_USAGE des SYS_INFO_SCHEMA

10.2.22 SYS_SCHEMATA

Information über die Schemata der Datenbank.

Spaltenname	Datentyp	Inhalt
SCHEMA_NAME	CHAR (31)	Name des Schemas
SCHEMA_OWNER	CHAR (18)	Berechtigungsschlüssel des Eigentümers

Tabelle 129: View SYS_SCHEMATA des SYS_INFO_SCHEMA

10.2.23 SYS_SPACES

Information über Spaces.

Spaltenname	Datentyp	Inhalt
SPACE_NAME	CHAR (18)	Name des Space
SPACE_NAME_SHORT	CHAR (12)	erste 12 Zeichen des Spacenamen
SPACE_ID	SMALLINT	Identifikationsnummer des Space
SPACE_OWNER	CHAR (18)	Berechtigungsschlüssel, der Eigentümer des Space ist
STOGROUP_NAME	CHAR (18)	Name der Storage Group für den Space
PCT_FREE	SMALLINT	Freiplatzreservierung in Prozent
DELTA_STOGROUP	CHAR (1)	Y Space auf Storage Group STOGROUP_NAME gespeichert N Space noch nicht auf der mit ALTER SPACE neu zugewiesenen Storage Group STOGROUP_NAME gespeichert
SPACE_DATE	TIMESTAMP (3)	Erzeugungszeitpunkt bzw. Zeitpunkt des letzten ändernden Zugriffs auf die Definition von Tabellen und Indizes des Spaces
LOGGING	VARCHAR (3)	YES Logging eingeschaltet NO Logging ausgeschaltet
Die Daten von STOGROUP_NAME und PCT_FREE werden durch ALTER SPACE geändert; diese Daten werden erst bei RECOVER und REORG tatsächlich berücksichtigt.		

Tabelle 130: View SYS_SPACES des SYS_INFO_SCHEMA

10.2.24 SYS_SPACE_PROPERTIES

Information über Spaceeigenschaften. Für nicht geöffnete Spaces wird nur ein Teil der Eigenschaften ausgegeben.

Spaltenname	Datentyp	Inhalt
SPACE_NAME	CHAR (18)	Name des Space
PROPERTY_NAME	CHAR (31)	Name der Spaceeigenschaft Folgende Eigenschaften werden ausgegeben: • SPACE_ID ist die aus dem Katalog ermittelte Nummer des Space. • SPACE_TIMESTAMP ist der Zeitstempel der letzten Änderung des Space. • CHECK_TIMESTAMP ist der Zeitstempel gegen den die Konsistenz des Space geprüft wird. • MAX_POSSIBLE_PAGE ist die größte mögliche Seitennummer des Space 1.073.741.822 (X'3FFFFFFE') für Spaces bis 4 TByte 16.777.214 (X'00FFFFFE') für Spaces bis 64 GByte • LAST_USED_PAGE ist die letzte logisch belegte 4K-Seite des Space. • SPACE_LOCK_RECOVER_PENDING gibt an, ob der Space bei derReparatur wiederhergestellt werden konnte oder nicht • SPACE_LOCK_LOAD_RUNNING gibt an, ob das Zuladen von Daten in eine Basistabelle des Space mit LOAD oder IMPORT TABLE (noch) nicht beendet wurde. • SPACE_LOCK_IS_COPY gibt an, ob der Sicherungsbestand des Space eingehängt ist und deshalb nur lesende Zugriffe erlaubt sind. • SPACE_LOCK_IS_REPLICATE gibt an, ob das Replikat des Space eingehängt ist und deshalb nur lesende Zugriffe erlaubt sind. • SPACE_LOCK_COPY_PENDING gibt an, ob der Space gegen Updates gesperrt ist, da noch ein COPY aussteht.

		• SPACE_LOCK_CHECK_PENDING gibt an, ob nach dem Laden in eine Basistabelle mit LOAD die Integritätsbedingungen noch nicht geprüft wurden. • SPACE_LOCK_REORG_PENDING gibt an, dass die maximale Space-Größe erreicht ist. Deshalb sind nur lesende Zugriffe sowie DELETE und REORG SPACE erlaubt. • SPACE_FLAG_OPENED gibt an, ob der Space eröffnet ist. • SPACE_FLAG_MODIFIED gibt an, ob der Space verändert ist. • SPACE_FLAG_DEFECT gibt an, ob der Space defekt ist. • OPEN_TIMESTAMP ist der Zeitpunkt des letzten physikalischen Öffnens des Space. Die Felder BLOCK_DENSITY_*xx* beschreiben die Anzahl der Blöcke, die seit OPEN_TIMESTAMP mit dem Blockfüllungsgrad größer als (*xx*-10)% und kleiner gleich *xx*% angefasst wurden: • BLOCK_DENSITY_10 • BLOCK_DENSITY_20 • . . . • BLOCK_DENSITY_90 • BLOCK_DENSITY_100
CHARACTER_VALUE	VARCHAR (256)	Wert der Spaceeigenschaft NULL-Wert falls INTEGER_VALUE nicht NULL
INTEGER_VALUE	INTEGER	Wert der Spaceeigenschaft NULL-Wert falls CHARACTER_VALUE nicht NULL

Tabelle 131: View SYS_SPACE_PROPERTIES des SYS_INFO_SCHEMA

10.2.25 SYS_SPECIAL_PRIVILEGES

Information über Sonder-Privilegien.

Spaltenname	Datentyp	Inhalt
GRANTEE	CHAR (18)	Berechtigungsschlüssel, dem das Privileg verliehen wurde, oder PUBLIC
PRIVILEGE_TYPE	CHAR (18)	Privileg-Typ: CREATE USER CREATE SCHEMA CREATE STOGROUP UTILITY
GRANTOR	CHAR (18)	Berechtigungsschlüssel, der das Privileg vergab, bzw. _SYSTEM
IS_GRANTABLE	VARCHAR (3)	YES der Berechtigungsschlüssel besitzt GRANT-Berechtigung für das Privileg NO keine GRANT-Berechtigung

Tabelle 132: View SYS_SPECIAL_PRIVILEGES des SYS_INFO_SCHEMA

10.2.26 SYS_STOGROUPS

Information über die Storage Groups der Datenbank.

Spaltenname	Datentyp	Inhalt
STOGROUP_NAME	CHAR (18)	Name der Storage Group
VOLUME_NAME	CHAR (6)	Datenträgerkennzeichen von Privatplatten oder PUBLIC
STOGROUP_OWNER	CHAR (18)	Berechtigungsschlüssel, der Eigentümer der Storage Group ist
CAT_ID	VARCHAR (4)	BS2000-Katalogkennung
DEVICE_TYPE	VARCHAR (8)	Gerätetyp der Privatplatten NULL-Wert für PUBLIC
ORDINAL_POSITION	SMALLINT	laufende Nummer der Privatplatte in der Storage Group 1 für PUBLIC

Tabelle 133: View SYS_STOGROUPS des SYS_INFO_SCHEMA

10.2.27 SYS_SYSTEM_ENTRIES

Information über die Systemeinträge der Datenbank.

Spaltenname	Datentyp	Inhalt
HOST_NAME	CHAR (8)	Rechnername oder *
APPLICATION_NAME	CHAR (8)	Anwendungsname oder * bei UTM-Systemeintrag Leerzeichen bei BS2000-Systemeintrag
SYSTEM_USER_NAME	CHAR (8)	BS2000- oder UTM-Benutzerkennung
USER_NAME	CHAR (18)	Berechtigungsschlüssel oder PUBLIC

Tabelle 134: View SYS_SYSTEM_ENTRIES des SYS_INFO_SCHEMA

10.2.28 SYS_TABLES

Information über Basistabellen und Views der Datenbank.

Spaltenname	Datentyp	Inhalt
TABLE_SCHEMA	CHAR (31)	Name des Schemas, zu dem die Tabelle gehört
TABLE_NAME	CHAR (31)	Name der Tabelle
TABLE_TYPE	VARCHAR (18)	BASE TABLE, VIEW oder ABSTRACT TABLE
TABLE_ID	SMALLINT	Identifikationsnummer der Basistabelle oder der abstrakten Tabelle. Bei TABLE_ID >= 30720 liegt eine partitionierte Tabelle vor. NULL-Wert für Views
SPACE_NAME	CHAR (18)	Name des Space, in dem die (nicht-partitionierte) Tabelle gespeichert ist. _PARTITIONS_ bei einer partitionierten Tabelle NULL-Wert sonst
SPACE_ID	SMALLINT	Identifikationsnummer des Space. Bei einer partitionierten Tabelle wird 32767 ausgegeben. NULL-Wert sonst
TABLE_STYLE	VARCHAR (6)	OLDEST Nur-CALL-DML-Tabelle OLD CALL-DML/SQL-Tabelle NEW SQL-Tabelle NULL-Wert sonst
TABLE_DATE	TIMESTAMP (3)	Erzeugungszeitpunkt bzw. Zeitpunkt des letzten ALTER TABLE
VIEW_DEFINITION	VARCHAR (32000)	Abfrage-Ausdruck, der View definiert, für Views NULL-Wert sonst
TABLE_PRIMARY_KEY	CHAR (1)	S einfacher Primärschlüssel

		C zusammengesetzter Primärschlüssel NULL-Wert sonst
CHECK_OPTION	VARCHAR (8)	NONE keine Check-Option gesetzt CASCADED Check-Option gesetzt NULL-Wert sonst
IS_UPDATABLE	VARCHAR (3)	YES View ist änderbar NO View ist nicht änderbar NULL-Wert sonst
TEMPORARY_VIEW	VARCHAR (3)	YES View ist temporär NO View ist permanent NULL-Wert sonst

Tabelle 135: View SYS_TABLES des SYS_INFO_SCHEMA

10.2.29 SYS_TABLE_CONSTRAINTS

Information über alle Integritätsbedingungen der Tabellen der Datenbank.

Spaltenname	Datentyp	Inhalt
CONSTRAINT_SCHEMA	CHAR (31)	Name des Schemas, zu dem die Tabelle mit der Integritätsbedingung gehört
CONSTRAINT_NAME	CHAR (31)	Name der Integritätsbedingung
CONSTRAINT_TYPE	VARCHAR (11)	Typ der Integritätsbedingung: FOREIGN KEY UNIQUE PRIMARY KEY CHECK
TABLE_NAME	CHAR (31)	Name der Tabelle, zu dem die Integritätsbedingung gehört

Tabelle 136: View SYS_TABLE_CONSTRAINTS des SYS_INFO_SCHEMA

10.2.30 SYS_UNIQUE_CONSTRAINTS

Information über Primärschlüssel- und Eindeutigkeitsbedingungen.

Spaltenname	Datentyp	Inhalt
CONSTRAINT_SCHEMA	CHAR (31)	Name des Schemas, zu dem die Tabelle mit der Integritätsbedingung gehört
CONSTRAINT_NAME	CHAR (31)	Name der Integritätsbedingung
TABLE_NAME	CHAR (31)	Name der Tabelle, zu dem die Integritätsbedingung gehört
COLUMN_NAME	CHAR (31)	Name einer Spalte der Integritätsbedingung
ORDINAL_POSITION	SMALLINT	Position der Spalte in der Integritätsbedingung
CONST_TYPE_IND	CHAR (1)	P Primärschlüsselbedingung U Eindeutigkeitsbedingung
INDEX_NAME	CHAR (18)	Name des Index für die Eindeutigkeitsbedingung NULL-Wert für Primärschlüsselbedingung

Tabelle 137: View SYS_UNIQUE_CONSTRAINTS des SYS_INFO_SCHEMA

10.2.31 SYS_USAGE_PRIVILEGES

Information über das USAGE-Privileg.

Spaltenname	Datentyp	Inhalt
GRANTEE	CHAR (18)	Berechtigungsschlüssel, dem das Privileg verliehen wurde, oder PUBLIC
OBJECT_SCHEMA	CHAR (31)	Name des Schemas, zu dem die Sortierreihenfolge bzw. der Zeichensatz gehört, für die das Privileg gilt Leerzeichen für Storage Group
OBJECT_NAME	CHAR (18)	Name der Storage Group, der Sortierreihenfolge bzw. des Zeichensatzes, für die das Privileg gilt
OBJECT_TYPE	CHAR (18)	Objekt, für das das Privileg gilt: STOGROUP CHARACTER SET COLLATION
GRANTOR	CHAR (18)	Berechtigungsschlüssel, der das Privileg vergab, bzw. _SYSTEM
IS_GRANTABLE	VARCHAR (3)	YES der Berechtigungsschlüssel besitzt GRANT-Berechtigung für das Privileg NO keine GRANT-Berechtigung

Tabelle 138: View SYS_USAGE_PRIVILEGES des SYS_INFO_SCHEMA

10.2.32 SYS_USERS

Information über alle Berechtigungsschlüssel der Datenbank.

Spaltenname	Datentyp	Inhalt
USER_NAME	CHAR (18)	Berechtigungsschlüssel
USER_NAME_SHORT	CHAR (10)	erste 10 Zeichen des Berechtigungsschlüssels

Tabelle 139: View SYS_USERS des SYS_INFO_SCHEMA

10.2.33 SYS_VIEW_USAGE

Information über die Tabellen und Spalten, die von Views verwendet werden.

Spaltenname	Datentyp	Inhalt
VIEW_SCHEMA	CHAR (31)	Name des Schemas, zu dem der View gehört
VIEW_NAME	CHAR (31)	Name des View
TABLE_SCHEMA	CHAR (31)	Name des Schemas, zu dem die Tabelle bzw. die Spalte gehört, die vom View verwendet wird
TABLE_NAME	CHAR (31)	Name der Tabelle, die im View verwendet wird bzw. zu der die Spalte gehört
COLUMN_NAME	CHAR (31)	Name der Tabellenspalte, die im View verwendet wird Leerzeichen, falls Information über Tabelle
OBJECT_INDICATOR	CHAR (1)	T Satz enthält Information über Tabelle C Satz enthält Information über Spalte
VIEW_COLUMN	CHAR (31)	Name der Viewspalte, wenn der View änderbar ist, OBJECT_INDICATOR den Wert C hat und die Viewspalte sich von einer Tabellenspalte (in COLUMN_NAME) ableitet NULL-Wert sonst

Tabelle 140: View SYS_VIEW_USAGE des SYS_INFO_SCHEMA

10.2.34 SYS_VIEW_ROUTINE_USAGE

Information über die User Defined Functions (UDFs), die in Views benutzt werden.

Spaltenname	Datentyp	Inhalt
TABLE_SCHEMA	CHAR(31)	Name des Schemas, zu dem der View gehört
TABLE_NAME	CHAR(31)	Name des Views
SPECIFIC_SCHEMA	CHAR(31)	Name des Schemas, zu dem die Routine gehört
SPECIFIC_NAME	CHAR(31)	spezifischer Name der Routine

Tabelle 141: View SYS_VIEW_ROUTINE_USAGE des SYS_INFO_SCHEMA

11 Anhang

Dieses Kapitel enthält folgende Teile:

- Übersicht über die grundlegenden Syntaxelemente von SESAM/SQL
- Syntaxübersicht CSV-Datei
- SQL-Schlüsselwörter

11.1 Syntaxelemente von SESAM/SQL

Im Folgenden sind die im Handbuch in den Kapiteln 3 bis 6 definierten grundlegenden Syntaxelemente in alphabetischer Reihenfolge zusammengestellt.
Für diese Syntaxelemente wird in der Syntax der SQL-Anweisungen nur ihre Bezeichnung (das ist der Name links vom Definitionszeichen „::=“) angegeben.

> ¡ In der Syntax kursiv gedruckte eckige Klammern sind Sonderzeichen und müssen in der Anweisung angegeben werden.

abfrageausdruck ::=
[*abfrageausdruck* { UNION [ALL | DISTINCT] | EXCEPT [DISTINCT] }]

{ *select_ausdruck* | TABLE *tabelle* | *join_ausdruck* | (*abfrageausdruck*) }

aggregat ::= <{ *wert* | NULL },...>

alphanumerisches_literal ::=
{ '[*zeichen* ...]'[*trenner* ...'[*zeichen* ...]']... |
X'[*hex hex*]...'[*trenner* ...'[*hex hex*]...']... }

hex ::= 0|1|2|3|4|5|6|7|8|9|a|b|c|d|e|f|A|B|C|D|E|F

annotation ::= /*% *annotation_text* %*/

anweisungsbezeichner ::= *einf_name*

argumente ::= siehe *user_defined_function*

ausdruck ::=
{
  wert |
 [*tabelle* .] { *spalte* | { *spalte* (*posnr*) | *spalte[posnr]* } | { *spalte* (*min..max*) | *spalte[min..max]* }
} |
  funktion |
  unterabfrage |
  un_op ausdruck |
  ausdruck bin_op ausdruck |

case_ausdruck |
cast_ausdruck |
(*ausdruck*)

}

spalte ::= *einf_name*
posnr ::= *vorzeichenlose_ganzzahl*
min ::= *vorzeichenlose_ganzzahl*
max ::= *vorzeichenlose_ganzzahl*

un_op ::= { + | - }

bin_op ::= { * | / | + | = | || }

berechtigungsschlüssel ::= *einf_name*

buchstabe ::= siehe *einf_name*

case_ausdruck ::=

{

CASE

WHEN *suchbedingung* THEN { *ausdruck* | NULL }
...

[ELSE { *ausdruck* | NULL }]
END |

CASE *ausdruckx*
WHEN *ausdruck1* [, *ausdruck2*] ... THEN { *ausdruck* | NULL }
...

[ELSE { *ausdruck* | NULL }]

END |

NULLIF (*ausdruck1* , *ausdruck2*) |

COALESCE (*ausdruck1* , *ausdruck2,* ... *ausdruckn*) |

{ MIN | MAX }(*ausdruck1,ausdruck2,* ..., *ausdruckn*)

```
 }
```

cast_ausdruck ::= CAST ({ *ausdruck* | NULL } AS *datentyp*)

catalog ::= *einf_name*

datentyp ::=

```
{
   [{[ dimension ] | ( dimension )}] CHAR[ACTER][( länge )] |
   CHAR[ACTER] VARYING( max ) | VARCHAR( max ) |
   [{[ dimension ] | ( dimension ) }] { NATIONAL CHAR[ACTER] | NCHAR } [ ( cu_länge
[CODE_UNITS])] |
   { NATIONAL CHAR[ACTER] VARYING | NCHAR VARYING | NVARCHAR }  ( cu_max [CODE_UNITS])
|
   [{[ dimension ] | ( dimension )}]
    {
      SMALLINT |
      INT[EGER] |
      NUMERIC [( stellen [, bruchteil ])] |DEC[IMAL][( stellen [, bruchteil ])] |
      REAL |
      DOUBLE PRECISION |
      FLOAT [( stellen )] |
      DATE  |
      TIME (3)  |
      TIMESTAMP (3)
    }
}
```

einf_basistabellenname ::= *einf_name*

einf_bedingungsname ::= *einf_name*

einf_indexname ::= *einf_name*

einf_name ::= { *regulärer_name* | *spezialname* }

regulärer_name ::= *buchstabe* [{ *buchstabe* | *ziffer* | _ }] ...

spezialname ::= " *zeichen...* "

buchstabe ::= a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p|q|r|s|t|u|v|w|x|y|z|
A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|T|U|V|W|X|Y|Z

ziffer ::= 0|1|2|3|4|5|6|7|8|9

einf_schemaname ::= *einf_name*

einf_spacename ::= *einf_name*

einf_stogroupname ::= *einf_name*

einf_viewname ::= *einf_name*

festpunktzahl ::= siehe *numerisches_literal*

funktion ::= { *zeitfunktion* | *zeichenkettenfunktion* | *numerische_funktion* | *mengenfunktion* | *tabellenfunktion* | *kryptofunktion* | *user_defined_function* }

ganzzahl ::= siehe *numerisches_literal*

gleitpunktzahl ::= siehe *numerisches_literal*

hex ::= siehe *alphanumerisches_literal*

index ::= siehe *qualifizierter_name*

integritätsbedingungsname ::= siehe *qualifizierter_name*

join_ausdruck ::=
{

tabellenangabe CROSS JOIN *tabellenangabe* |
tabellenangabe [INNER | { LEFT | RIGHT | FULL } [OUTER]]
JOIN *tabellenangabe* ON *suchbedingung* |
tabellenangabe UNION JOIN *tabellenangabe* |
(*join_ausdruck*)

}

korrelationsname ::= *einf_name*

kryptofunktion ::= { ENCRYPT (*ausdruck* , *schlüssel*) | DECRYPT (*ausdruck2* , *schlüssel* , *datentyp*) }

schlüssel ::= *ausdruck*

literal ::= { *alphanumerisches_literal* | *national_literal* | *spezial_literal* | *numerisches_literal* | *zeit_literal* }

max ::= *vorzeichenlose_ganzzahl*

mengenfunktion ::= { *operator* ([ALL | DISTINCT] *ausdruck*) | COUNT(*) }

operator ::= {AVG | COUNT | MAX | MIN | SUM }

min ::= *vorzeichenlose_ganzzahl*

modifikatoren ::= siehe *praedikat*

muster ::= siehe *praedikat*

national_literal ::=

```
{  N'[ zeichen ...]'[ trenner ...'[ zeichen ...]'] ... |
   NX'[ 4hex ...]'[ trenner ...'[ 4hex ...]'] ...|
   U&'[ uc-zeichen ...]'[ trenner... '[ uc-zeichen '...] ... [UESCAPE' esc '] }
```

uc-zeichen ::= { *zeichen* | *esc 4hex* | *esc+ 6hex* | *esc esc* }

numerische_funktion ::=

```
{
   ABS ( ausdruck ) |
   CEIL[ING] ( ausdruck ) |
   FLOOR ( ausdruck ) |
   MOD ( dividend,divisor ) |
   SIGN ( ausdruck ) |
   { CHAR_LENGTH | CHARACTER_LENGTH } ( ausdruck [USING { CODE_UNITS | OCTETS }]) |
   OCTET_LENGTH ( ausdruck ) |
   POSITION ( ausdruck IN ausdruck [USING CODE_UNITS]) |
   JULIAN_DAY_OF_DATE ( ausdruck ) |
   EXTRACT ( bestandteil FROM ausdruck )
}
```

numerisches_literal ::= { *ganzzahl* | *festpunktzahl* | *gleitpunktzahl* }

ganzzahl ::= [{+|-}] *vorzeichenlose_ganzzahl* [.]

festpunktzahl ::= [{+|-}] { *vorzeichenlose_ganzzahl* [. *vorzeichenlose_ganzzahl*] |
vorzeichenlose_ganzzahl . | . *vorzeichenlose_ganzzahl* }

gleitpunktzahl ::= *festpunktzahl* E[{+|-}] *vorzeichenlose_ganzzahl*

vorzeichenlose_ganzzahl ::= *ziffer* ...

operand ::= siehe *praedikat*

praedikat ::=
{

zeile vergleichs_op zeile |
vektor_spalte vergleichs_op ausdruck |
zeile vergleichs_op { ALL | SOME | ANY } *unterabfrage* |
zeile [NOT] BETWEEN *zeile* AND *zeile* |
vektor_spalte [NOT] BETWEEN *ausdruck* AND *ausdruck* |
ausdruck IS [NOT] CASTABLE AS *datentyp* |
zeile [NOT] IN { *unterabfrage* | (*zeile* , ...) } |
vektor_spalte [NOT] IN (*ausdruck* , *ausdruck* , ...) |
operand [NOT] LIKE *muster* [ESCAPE *zeichen* ...] |
operand [NOT] LIKE_REGEX *regulärer_ausdruck* [FLAG *modifikatoren*] |
ausdruck IS [NOT] NULL |
EXISTS *unterabfrage*

}

zeile ::= { (*ausdruck* , ...) | *ausdruck* | *unterabfrage* }

vektor_spalte ::= [*tabelle* .]{ *spalte[min..max]* | *spalte* (*min..max*) }

vergleichs_op ::= { = | < | > | <= | >= | <> }

operand ::= *ausdruck*

muster ::= *ausdruck*

zeichen ::= *ausdruck*

regulärer_ausdruck ::= *ausdruck*

modifikatoren ::= *ausdruck*

pragma ::= --%PRAGMA *pragma_text* , ... *zeilenende*

qualifizierter_name ::=
{

index |
integritätsbedingungsname |
routine |
schema |
space |

stogroup |
tabelle

}

index ::= [[*catalog* .] *einf_schemaname* .] *einf_indexname*

integritätsbedingungsname ::= [[*catalog* .] *einf_schemaname* .] *einf_bedingungsname*

routine ::= [[*catalog* .] *einf_schemaname* .] *einf_routinenname*

schema ::= [*catalog* .] *einf_schemaname*

space ::= [*catalog* .] *einf_spacename*

stogroup ::= [*catalog* .] *einf_stogroupname*

tabelle ::=

{

[[*catalog* .] *einf_schemaname* .] *einf_basistabellenname* |
[[*catalog* .] *einf_schemaname* .] *einf_viewname* |
korrelationsname
}

regulaerer Ausdruck ::= siehe *praedikat*

regulärer_name ::= siehe *einf_name*

routine ::= siehe *qualifizierter_name*

routinenparameter ::= *einf_name*

schema ::= siehe *qualifizierter_name*

schlüssel ::= siehe *kryptofunktion*

select_ausdruck ::=

SELECT [ALL | DISTINCT] *select_liste*

FROM *tabellenangabe* ,...

```
[WHERE suchbedingung ]

[GROUP BY spalte ,...]

[HAVING suchbedingung ]


select_liste ::= { * | { tabelle .* | ausdruck [[AS] spalte ] } }


space ::= siehe qualifizierter_name

spalte ::= siehe ausdruck


spaltenbedingung ::=

{

NOT NULL |

UNIQUE |

PRIMARY  KEY |

REFERENCES tabelle [( spalte )] |

CHECK ( suchbedingung )

}


spaltendefinition ::=
  spalte { datentyp [ voreinstellung ] | FOR REF( tabelle ) }
  [[CONSTRAINT integritätsbedingungsname ] spaltenbedingung ] ...
  [ call_dml_klausel ]


voreinstellung ::= DEFAULT

{

  alphanumerisches_literal |

  national_literal |

  numerisches_literal |

  zeit_literal |

  CURRENT_DATE |
  CURRENT_TIME(3) |
  LOCALTIME(3) |
  CURRENT_TIMESTAMP(3) |
  LOCALTIMESTAMP(3) |
```

```
  USER |
  CURRENT_USER |
  SYSTEM_USER |
  NULL |
  REF( tabelle )
}

call_dml_klausel ::= CALL DML call_dml_voreinst [ call_dml_symb_name ]

spezial_literal ::=
{
   CURRENT_CATALOG |
   CURRENT_ISOLATION_LEVEL |
   CURRENT_REFERENCED_CATALOG |
   CURRENT_SCHEMA |
   [CURRENT_]USER |
   SYSTEM_USER
}

spezialname ::= siehe einf_name

stogroup ::= siehe qualifizierter_name

suchbedingung ::=
{ praedikat | suchbedingung { AND | OR } suchbedingung | NOT suchbedingung | ( suchbedingung ) }

tabelle ::= siehe qualifizierter_name

tabellenangabe ::=
{
  tabelle [[AS] korrelationsname [( spalte , ...)]] |
  unterabfrage [AS] korrelationsname [( spalte , ...)] |
  TABLE([ catalog .] tabellenfunktion ) [WITH ORDINALITY] [[AS] korrelationsname [( spalte , ...)]]
|
  join_ausdruck
}
```

tabellenbedingung ::=
{

UNIQUE (*spalte* ,...) |
PRIMARY KEY (*spalte* ,...) |
FOREIGN KEY (*spalte* ,...) REFERENCES *tabelle* [(*spalte* ,...)] |
CHECK (*suchbedingung*)

}

tabellenfunktion ::=
{ CSV ([FILE] *datei* DELIMITER *delimiter* [QUOTE *quote*] [ESCAPE *escape*], *datentyp* ,...) | DEE
[()] }

unterabfrage ::= (*abfrageausdruck*)

user_defined_function ::= *einf_routinenname argumente*

argumente ::= ([*ausdruck* [{, *ausdruck* }...]])

vektor_spalte ::= siehe *praedikat*

vergleichs_op ::= siehe *praedikat*

voreinstellung ::= siehe *spaltendefinition*

vorzeichenlose_ganzzahl ::= siehe *numerisches_literal*

wert ::=
{

literal |
: *benutzervariable* [[INDICATOR] : *indikatorvariable*] |
routinenparameter |
lokale_variable |

?

}

zeichen ::= siehe *praedikat*

zeichenkettenfunktion ::=

```
{
   SUBSTRING ( ausdruck FROM  startposition [FOR  teilkettenlänge ][USING CODE_UNITS]) |
   TRANSLATE ( ausdruck USING [[ catalog .]INFORMATION_SCHEMA.] transname [DEFAULT zeichen
   ] [ ,länge ]) |
   TRIM ([[LEADING |TRAILING | BOTH] [ zeichen ] FROM] ausdruck ) |
   LOWER ( ausdruck ) |
   UPPER ( ausdruck ) |
   HEX_OF_VALUE ( ausdruck2 ) |
   VALUE_OF_HEX ( ausdruck3 , datentyp ) |
   REP_OF_VALUE ( ausdruck2 ) |
   VALUE_OF_REP ( ausdruck3 , datentyp ) |
   COLLATE ( ausdruck USING { DUCET_WITH_VARS | DUCET_NO_VARS } [ ,länge ]) |
   NORMALIZE ( ausdruck [,NFC | NFD [, länge ]])
}
```

zeichen ::= *ausdruck*

länge ::= *vorzeichenlose_ganzzahl*

zeitfunktion ::=

```
{
   CURRENT_DATE |
   CURRENT_TIME(3) |
   LOCALTIME(3) |
   CURRENT_TIMESTAMP(3) |
   LOCALTIMESTAMP(3) |
   DATE_OF_JULIAN_DAY ( ausdruck )
}
```

zeit_literal ::=

```
{
 DATE ' jahr-monat-tag ' |
 TIME ' stunde:minute:sekunde '
 TIMESTAMP' jahr-monat-tag stunde:minute:sekunde '
}
```

zeile ::= siehe *praedikat*

ziffer ::= siehe *einf_name*

11.2 Syntaxübersicht CSV-Datei

Kommentare in dieser Syntaxdarstellung sind in Anführungszeichen " eingeschlossen.

CSV_file_format ::=
[[*CSV_record*] *CSV_record_separator* [[*CSV_record*] *CSV_record_separator*]...] [*CSV_record*]

CSV_record ::=
{ [*CSV_field CSV_delimiter*]... *CSV_non-empty_field* | *CSV_field CSV_delimiter* [*CSV_field CSV_delimiter*]... }

CSV_record_separator ::=
{ X'04' "(EBCDIC control character NEL)" |
X'0D' "(EBCDIC control character CR)" |
X'15' "(EBCDIC control character LF)" |
X'25' "(EBCDIC control character)" |
"Satzende in einer BS2000-SAM-Datei" }

CSV_field ::= { *CSV non-empty field* | 'BLANK' "(leer)" }

CSV_non-empty_field ::= { *CSV plain field* | *CSV quoted field* }

CSV_plain_field ::= *CSV_plain_intro* [*CSV_plain_part*]...

CSV_plain_intro ::= { *CSV_escape_sequence* | "alle Zeichen mit Ausnahme von*CSV_record_separator*, *CSV_delimiter*, *CSV_escape* und *CSV_quote*" }

CSV_plain_part ::= { *CSV_escape_sequence* | "alle Zeichen mit Ausnahme von *CSV_record_separator*, *CSV_delimiter* und *CSV_escape*" }

CSV_quoted_field ::= *CSV_quote* [*CSV_quoted part*]... *CSV_quote*

CSV_quoted_part ::=
{ *CSV_quote CSV_quote* | *CSV_escape_sequence* | "Satzende in einer BS2000-SAM-Datei" | "alle Zeichen mit Ausnahme von *CSV_quote* und *CSV_escape*" }

CSV_escape_sequence ::=
{ *CSV_escape CSV_record_separator* | *CSV_escape CSV_delimiter* | *CSV_escape CSV_quote* | *CSV_escape CSV_escape* }

CSV_delimiter ::= *zeichen*

CSV_quote ::= *zeichen*

CSV_escape ::= *zeichen*

Zu *CSV_delimiter*, *CSV_quote* und *CSV_escape* siehe auch die Syntaxbeschreibung der CSV-Funktion auf "CSV() - BS2000-Datei als Tabelle lesen".

11.3 SQL-Schlüsselwörter

In SESAM/SQL gibt es Wörter, die als Schlüsselwörter für SQL- und Utility-Anweisungen reserviert sind. Die reservierten Schlüsselwörter dürfen Sie in SQL- und Utility-Anweisungen sowie beim Arbeiten mit dem Utility-Monitor nicht als Namen von Views, Tabellen, Spalten usw. angeben, es sei denn, Sie geben das betreffende Schlüsselwort in Form eines Spezialnamens an.

Die Synonymverarbeitung des ESQL-Precompilers bietet eine komfortable Möglichkeit zum Ersetzen von Schlüsselwörtern bzw. zur Redefinition von Namen.

Mit Hilfe der Precompiler-Option SOURCE-PROPERTIES kann der Anwender den Parameter ESQL-DIALECT auf ISO, OLD oder ALL-FEATURES setzen. Dadurch wird entschieden, ob der SQL-Dialekt ISO, OLD oder FULL zu benutzen ist.

Die folgende Tabelle zeigt die reservierten Schlüsselwörter und ordnet sie den SQL-Dialekten zu, in denen sie reserviert sind.

Schlüsselwort	ISO	OLD	FULL
ABS	X		X
ABSOLUTE	X		X
ACTION	X		X
ADD	X		X
ALL	X	X	X
ALLOCATE	X		X
ALTER	X		X
AND	X	X	X
ANY	X	X	X
ARE	X		X
AS	X	X	X
ASC	X	X	X
ASSERTION	X		X
AT	X		X
AUTHORIZATION	X	X	X
AVG	X	X	X
BEGIN	X	X	X
BETWEEN	X	X	X
BIT	X		X
BIT_LENGTH	X		X
BLOB	X		X

BOTH	X		X
BY	X	X	X
CALL	X		X
CASCADE	X		X
CASCADED	X		X
CASE	X		X
CAST	X		X
CATALOG	X		X
CEIL	X		X
CEILING	X		X
CHAR	X	X	X
CHARACTER	X	X	X
CHARACTER_LENGTH	X		X
CHAR_LENGTH	X		X
CHECK	X	X	X
CLOSE	X	X	X
COALESCE	X		X
COLLATE	X		X
COLLATION	X		X
COLUMN	X		X
COMMIT	X	X	X
CONNECT	X		X
CONNECTION	X		X
CONSTRAINT	X		X
CONSTRAINTS	X		X
CONTINUE	X	X	X
CONVERT	X		X
COPY			X
CORRESPONDING	X		X
COUNT	X	X	X
CREATE	X	X	X
CROSS	X		X

CURRENT	X	X	X
CURRENT_CATALOG	X		X
CURRENT_DATE	X		X
CURRENT_ISOLATION_LEVEL			X
CURRENT_REFERENCED_CATALOG			X
CURRENT_SCHEMA	X		X
CURRENT_TIME	X		X
CURRENT_TIMESTAMP	X		X
CURRENT_USER	X		X
CURSOR	X	X	X
DATA	X		X
DATE	X		X
DATE_OF_JULIAN_DAY			X
DAY	X		X
DEALLOCATE	X		X
DEC	X	X	X
DECIMAL	X	X	X
DECLARE	X	X	X
DECRYPT			X
DEFAULT	X	X	X
DEFERRABLE	X		X
DEFERRED	X		X
DELETE	X	X	X
DESC	X	X	X
DESCRIBE	X		X
DESCRIPTOR	X		X
DIAGNOSTICS	X		X
DIRECTORY			X
DISCONNECT	X		X
DISTINCT	X	X	X
DOMAIN	X		X
DOUBLE	X	X	X

DROP	X		X
ELSE	X		X
ENCRYPT			X
END	X	X	X
END-EXEC		X	
ESCAPE	X	X	X
EXCEPT	X		X
EXCEPTION	X		X
EXEC	X	X	X
EXECUTE	X	X	X
EXISTS	X	X	X
EXP	X		X
EXPORT			X
EXTERNAL	X		X
EXTRACT	X		X
FALSE	X		X
FETCH	X	X	X
FIRST	X	X	X
FLOAT	X	X	X
FLOOR	X		X
FOR	X	X	X
FORCED			X
FOREIGN	X	X	X
FOUND	X	X	X
FROM	X	X	X
FULL	X		X
GET	X		X
GLOBAL	X		X
GO	X	X	X
GOTO	X	X	X

GRANT	X	X	X
GROUP	X	X	X
HAVING	X	X	X
HEX_OF_VALUE			X
HOLD	X		X
HOUR	X		X
IDENTITY	X		X
IMMEDIATE	X	X	X
IMPORT			X
IN	X	X	X
INDICATOR	X	X	X
INITIALLY	X		X
INNER	X		X
INPUT	X		X
INSERT	X	X	X
INT	X	X	X
INTEGER	X	X	X
INTERSECT	X		X
INTERVAL	X		X
INTO	X	X	X
IS	X	X	X
ISOLATION	X		X
JOIN	X		X
JULIAN_DAY_OF_DATE			X
KEY	X	X	X
LANGUAGE	X	X	X
LAST	X	X	X
LEADING	X		X
LEFT	X		X

LEVEL	X	X	X
LIKE	X	X	X
LIKE_REGEX	X		X
LN	X		X
LOAD			X
LOCAL	X		X
LOCALTIME	X		X
LOCALTIMESTAMP	X		X
LOWER	X		X
MATCH	X		X
MATCHED	X		X
MAX	X	X	X
MERGE	X		X
MIGRATE			X
MIN	X	X	X
MINUTE	X		X
MOD	X		X
MODIFY			X
MODULE	X	X	X
MONTH	X		X
NAMES	X		X
NATIONAL	X		X
NATURAL	X		X
NCHAR	X		X
NEW	X		X
NEXT	X	X	X
NO	X		X
NORMALIZE	X		X
NOT	X	X	X
NULL	X	X	X
NULLIF	X		X

NUMERIC	X	X	X
NVARCHAR			X
OCTET_LENGTH	X		X
OF	X	X	X
OLD	X		X
ON	X	X	X
ONLY	X	X	X
OPEN	X	X	X
OPTION	X	X	X
OR	X	X	X
ORDER	X	X	X
OUTER	X		X
OUTPUT	X		X
OVERLAPS	X		X
PARTIAL	X		X
PERMIT		X	X
POSITION	X		X
POWER	X		X
PRECISION	X	X	X
PREPARE	X	X	X
PRESERVE	X		X
PRIMARY	X	X	X
PRIOR	X	X	X
PRIVILEGES	X	X	X
PROCEDURE	X	X	X
PUBLIC	X	X	X
READ	X	X	X
REAL	X	X	X
RECOVER			X
REF	X		X

REFERENCES	X	X	X
REFRESH			X
RELATIVE	X		X
REORG			X
REP_OF_VALUE			X
RESTORE		X	X
RESTRICT	X		X
RETURN	X	X	X
REVOKE	X		X
RIGHT	X		X
ROLLBACK	X	X	X
ROWS	X		X
SCHEMA	X	X	X
SCOPE	X		X
SCROLL	X	X	X
SECOND	X		X
SECTION	X	X	X
SELECT	X	X	X
SESSION	X		X
SESSION_USER	X		X
SET	X	X	X
SIGN			X
SIZE	X		X
SMALLINT	X	X	X
SOME	X	X	X
SORTED			X
SQL	X	X	X
SQLCODE		X	X
SQLERROR	X	X	X
SQLSTATE	X		X
SQRT	X		X
STORE		X	X

SUBSTRING	X		X
SUM	X	X	X
SYSTEM	X		X
SYSTEM_USER	X		X
TABLE	X	X	X
TEMPORARY	X	X	X
THEN	X		X
TIME	X		X
TIMESTAMP	X		X
TIMEZONE_HOUR	X		X
TIMEZONE_MINUTE	X		X
TO	X	X	X
TRAILING	X		X
TRANSACTION	X	X	X
TRANSLATE	X		X
TRANSLATION	X		X
TRIM	X		X
TRUE	X		X
TRUNC			X
UESCAPE	X		X
UNION	X	X	X
UNIQUE	X	X	X
UNKNOWN	X		X
UNLOAD			X
UPDATE	X	X	X
UPPER	X		X
USAGE	X		X
USER	X	X	X
USING	X	X	X
VALUE	X		X

VALUES	X	X	X
VALUE_OF_HEX			X
VALUE_OF_REP			X
VARCHAR	X		X
VARYING	X		X
VIEW	X	X	X
WHEN	X		X
WHENEVER	X	X	X
WHERE	X	X	X
WITH	X	X	X
WITHOUT	X		X
WORK	X	X	X
WRITE	X	X	X
YEAR	X		X
ZONE	X		X

Tabelle 142: SESAM/SQL-Schlüsselwörter

12 Literatur

Die Handbücher finden Sie im Internet unter http://manuals.ts.fujitsu.com. Handbücher, die mit einer Bestellnummer angezeigt werden, können Sie in auch gedruckter Form bestellen.

SESAM/SQL-Server (BS2000)
Basishandbuch
Benutzerhandbuch

SESAM/SQL-Server (BS2000)
SQL-Sprachbeschreibung Teil 2: Utilities
Benutzerhandbuch

SESAM/SQL-Server (BS2000)
CALL-DML Anwendungen
Benutzerhandbuch

SESAM/SQL-Server (BS2000)
Datenbankbetrieb
Benutzerhandbuch

SESAM/SQL-Server (BS2000)
Utility-Monitor
Benutzerhandbuch

SESAM/SQL-Server (BS2000)
Meldungen
Benutzerhandbuch

SESAM/SQL-Server (BS2000)
Performance
Benutzerhandbuch

ESQL-COBOL (BS2000)
ESQL-COBOL für SESAM/SQL-Server
Benutzerhandbuch

SESAM-DBAccess
Server-Installation, Administration (nur auf dem Handbuch-Server verfügbar)