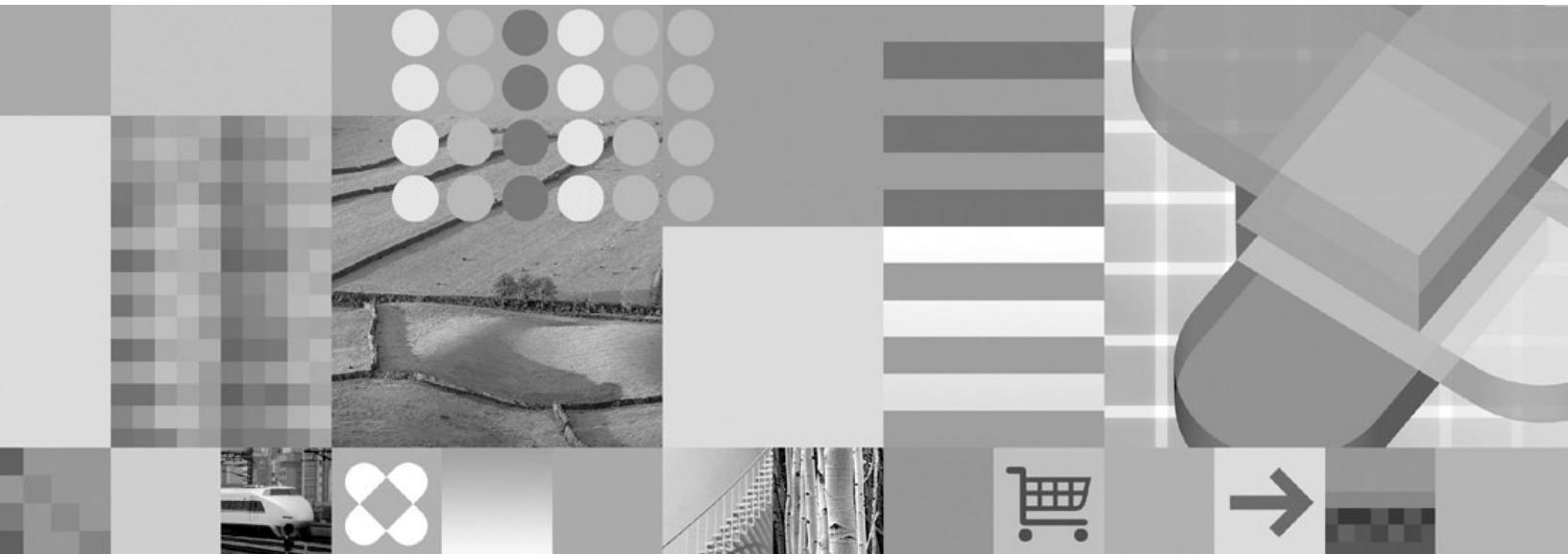


Version 7



SQL Reference

Version 7



SQL Reference

Note

Before using this information and the product it supports, be sure to read the general information under Appendix H, "Notices," on page 1173.

Sixth Edition, Softcopy Only (August 2005)

This edition applies to Version 7 of IBM DATABASE 2 Universal Database Server for OS/390 and z/OS (DB2 for OS/390 and z/OS), 5675-DB2, and to any subsequent releases until otherwise indicated in new editions. Make sure you are using the correct edition for the level of the product.

This softcopy version is based on the printed edition of the book and includes the changes indicated in the printed version by vertical bars. Additional changes made to this softcopy version of the book since the hardcopy book was published are indicated by the hash (#) symbol in the left-hand margin. Editorial changes that have no technical significance are not noted.

This and other books in the DB2 for OS/390 and z/OS library are periodically updated with technical changes. These updates are made available to licensees of the product on CD-ROM and on the Web (currently at www.ibm.com/software/data/db2/os390/library.html). Check these resources to ensure that you are using the most current information.

© Copyright International Business Machines Corporation 1982, 2005. All rights reserved.

US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

About this book	xv
Who should read this book	xv
Conventions and terminology used in this book	xv
Product terminology and citations	xv
Conventions for describing mixed data values	xvi
SQL standards	xvi
How to read the syntax diagrams	xvii
How to send your comments	xviii
Summary of changes to this book	xix
Chapter 1. DB2 concepts	1
Structured query language	2
Static SQL	2
Dynamic SQL	2
Deferred embedded SQL	2
Interactive SQL	3
DB2 Open Database Connectivity (ODBC)	3
DB2 access for Java (JDBC and SQLJ)	3
Schemas	3
Tables	4
Indexes	5
Keys	5
Unique keys	5
Primary keys	5
Parent keys	6
Foreign keys	6
Constraints	6
Unique constraints	6
Referential constraints	7
Check constraints	9
Triggers	9
Storage structures	10
Storage groups	10
Databases	10
Catalog	10
Views	10
Application processes, concurrency, and recovery	11
Locking, commit, and rollback	11
Unit of work	12
Unit of recovery	13
Rolling back work	13
Packages and application plans	14
Distributed data	14
DRDA access	15
DB2 private protocol access	16
Connection management for DRDA access and DB2 private protocol	17
Character conversion	20
Character sets and code pages	22
System CCSIDs	23
Expanding conversions	24
Contracting conversions	25
Other considerations for using UTF-8 and UTF-16	25

Chapter 2. Language elements	27
Characters	31
Tokens	31
Identifiers	32
SQL identifiers	32
Location identifiers	34
Host identifiers	34
Naming conventions	34
Qualification of unqualified object names	39
Schemas and the SQL path	40
Aliases and synonyms	41
Authorization IDs and authorization-names	42
Authorization IDs and schema names	43
Authorization IDs and statement preparation	43
Authorization IDs and dynamic SQL	43
Authorization IDs and remote execution	46
Data types	48
Character strings	49
Graphic strings	52
Binary strings	53
Large objects (LOBs)	53
Restrictions using long strings	54
Numbers	55
Datetime values	56
Row ID values	60
Distinct types	60
Promotion of data types	61
Casting between data types	62
Assignment and comparison	64
Numeric assignments	66
String assignments	68
Datetime assignments	70
Row ID assignments	71
Distinct type assignments	71
Numeric comparisons	72
String comparisons	73
Datetime comparisons	75
Row ID comparisons	75
Distinct type comparisons	75
Rules for result data types	77
String operands	77
Binary string operands	78
Numeric operands	78
Datetime operands	79
Row ID operands	79
Distinct type operands	79
Nullable attribute of a result	79
Constants	79
Integer constants	79
Floating-point constants	80
Decimal constants	80
Character string constants	80
Datetime constants	81
Graphic string constants	81
Special registers	82
General rules for special registers	83

CURRENT APPLICATION ENCODING SCHEME	85
CURRENT DATE	86
CURRENT DEGREE	86
CURRENT LOCALE LC_CTYPE	87
CURRENT MEMBER	87
CURRENT OPTIMIZATION HINT	87
CURRENT PACKAGESET	88
CURRENT PATH	88
CURRENT PRECISION	89
CURRENT RULES	90
CURRENT SERVER	91
CURRENT SQLID	91
CURRENT TIME	91
CURRENT TIMESTAMP	92
CURRENT TIMEZONE	92
USER	92
Inheriting special registers in a user-defined function or a stored procedure	93
Column names	95
Qualified column names	95
Correlation names	95
Column name qualifiers to avoid ambiguity	96
Column name qualifiers in correlated references	97
Resolution of column name qualifiers and column names	98
References to variables	99
References to host variables	100
Host variables in dynamic SQL	101
References to LOB host variables	101
References to LOB locator variables	102
References to stored procedure result sets	102
References to result set locator variables	102
Host structures in PL/I, C, and COBOL	103
Functions	104
Types of functions	104
Function resolution	107
Function invocation	110
Expressions	111
Without operators	111
With the concatenation operator	112
With arithmetic operators	114
Arithmetic with two integer operands	114
Arithmetic with an integer and a decimal operand	114
Arithmetic with two decimal operands	114
Arithmetic with floating-point operands	117
Datetime operands and durations	117
Datetime arithmetic in SQL	118
Precedence of operations	123
CASE expressions	124
CAST specification	126
Predicates	130
Basic predicate	130
Quantified predicate	132
BETWEEN predicate	134
EXISTS predicate	134
IN predicate	136
LIKE predicate	137
NULL predicate	144

Search conditions	145
Options affecting SQL	146
Precompiler options for dynamic statements.	148
Decimal point representation	148
Apostrophes and quotation marks in string delimiters	149
Katakana characters for EBCDIC.	150
Mixed data in character strings	150
Formatting of datetime strings	151
SQL standard language	151
Positioned updates of columns	153
Chapter 3. Functions	155
Column functions	161
AVG	162
COUNT	163
COUNT_BIG	164
MAX	166
MIN	167
STDDEV or STDDEV_POP	168
STDDEV_SAMP	169
SUM	170
VARIANCE, VAR, or VAR_POP	171
VARIANCE_SAMP or VAR_SAMP	172
Scalar functions	173
ABS or ABSVAL	174
ACOS.	175
ADD_MONTHS	176
ASIN	178
ATAN	179
ATANH	180
ATAN2	181
BLOB	182
CCSID_ENCODING	183
CEIL or CEILING	184
CHAR.	185
CLOB	191
COALESCE	192
CONCAT	194
COS	195
COSH.	196
DATE	197
DAY	198
DAYOFMONTH	199
DAYOFWEEK	200
DAYOFWEEK_ISO	201
DAYOFYEAR	202
DAYS	203
DBCLOB	204
DECIMAL or DEC	205
DEGREES	207
DIGITS	208
DOUBLE or DOUBLE_PRECISION	209
EXP	210
FLOAT	211
FLOOR	212
GENERATE_UNIQUE	213

#

GRAPHIC	215
HEX	218
HOUR	219
IDENTITY_VAL_LOCAL	220
IFNULL	224
INSERT	225
INTEGER or INT	228
JULIAN_DAY	229
LAST_DAY	230
LCASE or LOWER	231
LEFT	232
LENGTH	234
LN	235
LOCATE	236
LOG10	238
LTRIM	239
MAX	240
MICROSECOND	241
MIDNIGHT_SECONDS	242
MIN	243
MINUTE	244
MOD	245
MONTH	247
# MQPUBLISH	248
# MQPUBLISHXML	250
# MQREAD	252
# MQREADCLOB	254
# MQREADXML	256
# MQRECEIVE	257
# MQRECEIVECLOB	259
# MQRECEIVEXML	261
# MQSEND	263
# MQSENDXML	265
# MQSENDXMLFILE	267
# MQSENDXMLFILECLOB	269
# MQSUBSCRIBE	271
# MQUNSUBSCRIBE	273
MULTIPLY_ALT	275
NEXT_DAY	276
NULLIF	277
POSSTR	278
POWER	280
QUARTER	281
RADIANS	282
RAISE_ERROR	283
RAND	284
REAL	285
REPEAT	286
REPLACE	288
RIGHT	290
ROUND	292
ROUND_TIMESTAMP	294
ROWID	297
RTRIM	298
SECOND	299
SIGN	300

SIN.	301
SINH	302
SMALLINT	303
# SOAPHTTPC and SOAPHTTPV	304
SPACE	305
SQRT.	306
STRIP	307
SUBSTR.	309
TAN	312
TANH	313
TIME	314
TIMESTAMP	315
TIMESTAMP_FORMAT	316
TRANSLATE	317
TRUNCATE or TRUNC	320
TRUNC_TIMESTAMP	321
UCASE or UPPER	322
VARCHAR	323
VARCHAR_FORMAT	327
VARGRAPHIC	328
WEEK	331
WEEK_ISO	332
YEAR	333
# Table functions	334
# MQREADALL	335
# MQREADALLCLOB	337
# MQREADALLXML	339
# MQRECEIVEALL	341
# MQRECEIVEALLCLOB	343
# MQRECEIVEALLXML	345
Chapter 4. Queries	347
Authorization	348
subselect	349
select-clause	349
from-clause	352
where-clause	358
group-by-clause	359
having-clause	359
Examples of subselects	360
fullselect	365
Character conversion in unions and concatenations	366
Selecting the result CCSID	366
Examples of fullselects	367
select-statement	369
order-by-clause	370
read-only-clause	371
update-clause	372
optimize-for-clause	372
with-clause	373
queryno-clause	374
fetch-first-clause	374
Examples of select statements	375
Chapter 5. Statements	377
How SQL statements are invoked	380

Embedding a statement in an application program	381
Dynamic preparation and execution	382
Static invocation of a SELECT statement	383
Dynamic invocation of a SELECT statement	383
Interactive invocation	384
ALLOCATE CURSOR	386
ALTER DATABASE	388
ALTER FUNCTION (external)	391
ALTER FUNCTION (SQL scalar)	408
ALTER INDEX	414
ALTER PROCEDURE (external)	427
ALTER PROCEDURE (SQL)	438
ALTER STOGROUP	444
ALTER TABLE	447
ALTER TABLESPACE	467
ASSOCIATE LOCATORS	479
BEGIN DECLARE SECTION	482
CALL	483
CLOSE	491
COMMENT	493
COMMIT	499
CONNECT	501
CONNECT (Type 1)	504
CONNECT (Type 2)	510
CREATE ALIAS	514
CREATE AUXILIARY TABLE	516
CREATE DATABASE	519
CREATE DISTINCT TYPE	522
CREATE FUNCTION	529
CREATE FUNCTION (external scalar)	530
CREATE FUNCTION (external table)	553
CREATE FUNCTION (sourced)	571
CREATE FUNCTION (SQL scalar)	585
CREATE GLOBAL TEMPORARY TABLE	595
CREATE INDEX	601
CREATE PROCEDURE (external)	617
CREATE PROCEDURE (SQL)	636
CREATE STOGROUP	648
CREATE SYNONYM	651
CREATE TABLE	653
CREATE TABLESPACE	681
CREATE TRIGGER	699
CREATE VIEW	711
DECLARE CURSOR	718
DECLARE GLOBAL TEMPORARY TABLE	725
DECLARE STATEMENT	735
DECLARE TABLE	736
DECLARE VARIABLE	739
DELETE	742
DESCRIBE (prepared statement or table)	749
DESCRIBE CURSOR	756
DESCRIBE INPUT	758
DESCRIBE PROCEDURE	760
DROP	763
END DECLARE SECTION	775
EXECUTE	776

EXECUTE IMMEDIATE	779
EXPLAIN	781
FETCH	793
FREE LOCATOR	802
GRANT	803
GRANT (collection privileges)	806
GRANT (database privileges)	807
GRANT (distinct type or JAR privileges)	809
GRANT (function or procedure privileges)	811
GRANT (package privileges)	816
GRANT (plan privileges)	818
GRANT (schema privileges)	819
GRANT (system privileges)	821
GRANT (table or view privileges)	824
GRANT (use privileges)	827
HOLD LOCATOR	829
INCLUDE	830
INSERT	832
LABEL ON	838
LOCK TABLE	840
OPEN.	842
PREPARE	846
RELEASE (connection)	859
RELEASE SAVEPOINT	862
RENAME	863
REVOKE	865
REVOKE (collection privileges)	870
REVOKE (database privileges)	871
REVOKE (distinct type or JAR privileges)	874
REVOKE (function or procedure privileges)	876
REVOKE (package privileges)	881
REVOKE (plan privileges)	883
REVOKE (schema privileges)	884
REVOKE (system privileges)	886
REVOKE (table or view privileges)	889
REVOKE (use privileges)	892
ROLLBACK	894
SAVEPOINT	897
SELECT	899
SELECT INTO	900
SET CONNECTION	904
SET CURRENT APPLICATION ENCODING SCHEME	906
SET CURRENT DEGREE	907
SET CURRENT LOCALE LC_CTYPE	909
SET CURRENT OPTIMIZATION HINT	911
SET CURRENT PACKAGESET	912
SET CURRENT PRECISION	914
SET CURRENT RULES	915
SET CURRENT SQLID	916
SET host-variable assignment	918
SET PATH	921
SET transition-variable assignment	924
SIGNAL SQLSTATE	927
UPDATE.	928
VALUES	938
VALUES INTO	939

WHENEVER	941
Chapter 6. SQL procedure statements	943
SQL-procedure-statement	944
assignment-statement	946
CALL statement	948
CASE statement	950
compound-statement	952
GET DIAGNOSTICS statement	957
GOTO statement.	958
IF statement	960
LEAVE statement	961
LOOP statement.	962
REPEAT statement.	963
WHILE statement	964
Appendix A. Limits in DB2 for OS/390 and z/OS	965
Appendix B. Characteristics of SQL statements in DB2 for OS/390 and z/OS	969
Actions allowed on SQL statements	969
SQL statements allowed in external functions and stored procedures	972
SQL statements allowed in SQL procedures.	974
Appendix C. SQLCA and SQLDA	979
SQL communication area (SQLCA)	979
Description of fields.	979
The included SQLCA	982
The REXX SQLCA	985
SQL descriptor area (SQLDA)	986
Field descriptions	988
Unrecognized and unsupported SQLTYPES.	997
The included SQLDA	998
Identifying an SQLDA in C or C++	1003
The REXX SQLDA	1003
Appendix D. DB2 catalog tables	1005
Table spaces and indexes	1006
SQL statements allowed on the catalog	1011
Reorganizing the catalog	1013
New and changed catalog tables	1014
SYSIBM.IP NAMES table	1016
SYSIBM.LOCATIONS table	1017
SYSIBM.LULIST table	1018
SYSIBM.LUMODES table	1019
SYSIBM.LUNAMES table	1020
SYSIBM.MODESELECT table	1022
SYSIBM.SYSAUXRELS table	1023
SYSIBM.SYSCHECKDEP table	1024
SYSIBM.SYSCHECKS table	1025
SYSIBM.SYSCHECKS2 table	1026
SYSIBM.SYSCOLAUTH table	1027
SYSIBM.SYSCOLDIST table	1028
SYSIBM.SYSCOLDIST_HIST table	1029
SYSIBM.SYSCOLDISTSTATS table	1030
SYSIBM.SYSCOLSTATS table	1031

SYSIBM.SYSCOLUMNS table	1032
SYSIBM.SYSCOLUMNS_HIST table	1037
SYSIBM.SYSCONSTDEP table	1039
SYSIBM.SYSCOPY table	1040
SYSIBM.SYSDATABASE table	1044
SYSIBM.SYSDATATYPES table.	1046
SYSIBM.SYSDBAUTH table	1047
SYSIBM.SYSDBRM table	1049
SYSIBM.SYSDUMMY1 table	1051
SYSIBM.SYSFIELDS table	1052
SYSIBM.SYSFOREIGNKEYS table	1053
SYSIBM.SYSINDEXES table	1054
SYSIBM.SYSINDEXES_HIST table	1057
SYSIBM.SYSINDEXPART table	1058
SYSIBM.SYSINDEXPART_HIST table	1061
SYSIBM.SYSINDEXSTATS table	1062
SYSIBM.SYSINDEXSTATS_HIST table	1063
SYSIBM.SYSJARCLASS_SOURCE table	1064
SYSIBM.SYSJARCONTENTS table	1065
SYSIBM.SYSJARDATA table	1066
SYSIBM.SYSJAROBJECTS table	1067
SYSIBM.SYSJAVAOPTS table	1068
SYSIBM.SYSKEYCOLUSE table	1069
SYSIBM.SYSKEYS table	1070
SYSIBM.SYSLOBSTATS table	1071
SYSIBM.SYSLOBSTATS_HIST table	1072
SYSIBM.SYSPACKAGE table	1073
SYSIBM.SYSPACKAUTH table	1078
SYSIBM.SYSPACKDEP table	1079
SYSIBM.SYSPACKLIST table	1080
SYSIBM.SYSPACKSTMT table	1081
SYSIBM.SYSPARMS table	1084
SYSIBM.SYSPKSYSTEM table	1086
SYSIBM.SYSPLAN table	1087
SYSIBM.SYSPLANAUTH table	1091
SYSIBM.SYSPLANDEP table	1092
SYSIBM.SYSPLSYSTEM table	1093
SYSIBM.SYSPROCEDURES table	1094
SYSIBM.SYSRELS table	1097
SYSIBM.SYSRESAUTH table	1098
SYSIBM.SYSROUTINEAUTH table	1099
SYSIBM.SYSROUTINES table	1100
SYSIBM.SYSROUTINES_OPTS table	1106
SYSIBM.SYSROUTINES_SRC table	1107
SYSIBM.SYSSCHEMAAUTH table.	1108
SYSIBM.SYSSEQUENCES table	1109
SYSIBM.SYSSEQUENCESDEP table.	1110
SYSIBM.SYSSTMT table	1111
SYSIBM.SYSSTOGROUP table	1114
SYSIBM.SYSSTRINGS table	1115
SYSIBM.SYSSYNONYMS table	1117
SYSIBM.SYSTABAUTH table	1118
SYSIBM.SYSTABCONST table	1120
SYSIBM.SYSTABLEPART table	1121
SYSIBM.SYSTABLEPART_HIST table	1124
SYSIBM.SYSTABLES table	1126

SYSIBM.SYSTABLES_HIST table	1130
SYSIBM.SYSTABLESPACE table	1131
SYSIBM.SYSTABSTATS table	1134
SYSIBM.SYSTABSTATS_HIST table	1135
SYSIBM.SYSTRIGGERS table	1136
SYSIBM.SYSUSERAUTH table	1137
SYSIBM.SYSVIEWDEP table.	1140
SYSIBM.SYSVIEWS table	1141
SYSIBM.SYSVOLUMES table	1142
SYSIBM.USERNAMES table	1143
Appendix E. Using the catalog in database design.	1145
Retrieving catalog information about DB2 storage groups	1145
Retrieving catalog information about a table	1145
Retrieving catalog information about aliases	1145
Retrieving catalog information about columns	1146
Retrieving catalog information about indexes	1147
Retrieving catalog information about views	1147
Retrieving catalog information about authorizations.	1147
Retrieving catalog information about parent keys	1148
Retrieving catalog information about foreign keys	1148
Retrieving catalog information about check pending	1149
Retrieving catalog information about table check constraints	1149
Retrieving catalog information about LOBs	1149
Retrieving catalog information about user-defined functions and stored procedures.	1150
Retrieving catalog information about triggers	1150
Retrieving catalog information about distinct types	1150
Adding and retrieving comments	1151
Verifying the accuracy of the database definition.	1151
Appendix F. SQL reserved words	1153
Appendix G. Sample user-defined functions	1155
ALTDATE	1156
ALTTIME	1159
CURRENCY	1161
DAYNAME	1163
MONTHNAME	1164
TABLE_LOCATION	1165
TABLE_NAME	1167
TABLE_SCHEMA	1169
WEATHER	1171
Appendix H. Notices	1173
Programming interface information	1174
Trademarks	1175
Glossary	1177
Bibliography	1197
Index	X-1

About this book

This book is a reference for Structured Query Language (SQL) for DB2 Universal Database™ Server for OS/390® (DB2® for OS/390) and z/OS.

Unless otherwise stated, references to SQL in this book imply SQL for DB2 for OS/390 and z/OS, and all objects described in this book are objects of DB2 for OS/390 and z/OS. The syntax and semantics of most SQL statements are essentially the same in all IBM® relational database products, and the language elements common to the products provide a base for the definition of IBM SQL. Consult *IBM SQL Reference* if you intend to develop applications that adhere to IBM SQL.

Important

In this version of DB2® for OS/390® and z/OS™, some utility functions are available as optional products. You must separately order and purchase a license to such utilities, and discussion of those utility functions in this publication is not intended to otherwise imply that you have a license to them.

Who should read this book

This book is intended for end users, application programmers, system and database administrators, and for persons involved in error detection and diagnosis.

This book is a reference rather than a tutorial. It assumes that you are already familiar with Structured Query Language (SQL) programming concepts.

When you first use this book, consider reading Chapters 1 and 2 sequentially. These chapters describe the basic concepts of relational databases and SQL, the basic syntax of SQL, and the language elements that are common to many SQL statements. The rest of the chapters and appendixes are designed for the quick location of answers to specific SQL questions. They provide you with query forms, SQL statements, SQL procedure statements, DB2 limits, SQLCA, SQLDA, catalog tables, and SQL reserved words.

Conventions and terminology used in this book

This section explains conventions and terminology used in this book.

Product terminology and citations

In this book, DB2 Universal Database™ Server for OS/390 and z/OS is referred to as "DB2 for OS/390 and z/OS." In cases where the context makes the meaning clear, DB2 for OS/390 and z/OS is referred to as "DB2." When this book refers to other books in this library, a short title is used. (For example, "See *DB2 SQL Reference*" is a citation to *IBM® DATABASE 2™ Universal Database Server for OS/390 and z/OS SQL Reference*.)

When referring to a DB2 product other than DB2 for OS/390 and z/OS, this book uses the product's full name to avoid ambiguity.

The following terms are used as indicated:

DB2	Represents either the DB2 licensed program or a particular DB2 subsystem.
C, C++, and the C language	Represent the C or C++ programming language.
CICS®	Represents CICS/ESA® and CICS Transaction Server for OS/390.
IMS™	Represents IMS or IMS/ESA®.
MVS	Represents the MVS element of OS/390.
OS/390	Represents the OS/390 or z/OS operating system.
RACF®	Represents the functions that are provided by the RACF component of the SecureWay® Security Server for OS/390 or by the RACF component of the OS/390 Security Server.

Conventions for describing mixed data values

At sites using a double-byte character set (DBCS), character strings can include a mixture of single-byte and double-byte characters. When mixed data values are shown in the examples, the following conventions apply:

Convention	Representation
\textcircled{S}_0	"shift-out" control character (X'0E'), used only for EBCDIC data
\textcircled{S}_1	"shift-in" control character (X'0F'), used only for EBCDIC data
sbcsv-string	SBCS string of zero or more single-byte characters
dbcsv-string	DBCS string of zero or more double-byte characters
'	DBCS apostrophe
G	DBCS uppercase G

SQL standards

The SQL99 ANSI/ISO standard is a replacement for SQL92. SQL99 includes many new enhancements, such as object-relational capabilities, triggers, and many built-in functions to aid in data analysis. DB2 family functionality that appears in the SQL99 standard includes:

- SQL Extenders, comprised of text, image, and spatial extenders
- Extensions to SQL for online analytical processing and business intelligence
- Inclusion of federated database support in DB2 Universal Database for UNIX®, Windows®, OS/2®
- Object-relational extensions (user-defined types, user-defined functions, method, and so on)

DB2 for OS/390 and z/OS conforms to the following standards for SQL:

- FIPS (Federal Information Processing Standards) publication 127-2, Database Language SQL

- ANSI (American National Standards Institute) X3.135-1992, Database Language SQL, Entry Level
- ISO (International Standards Organization) 9075-1992, Database Language SQL, Entry Level
- IBM SQL Standards, Version 2

The ANSI and ISO documents are collectively referred to as SQL92. DB2 also supports a subset of the SQL99 standard, which includes a large portion of the core standard items.

How to read the syntax diagrams

The following rules apply to the syntax diagrams used in this book:

- Read the syntax diagrams from left to right, from top to bottom, following the path of the line.

The ►— symbol indicates the beginning of a statement.

The —→ symbol indicates that the statement syntax is continued on the next line.

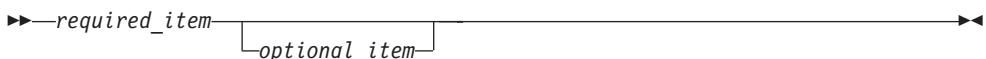
The —→ symbol indicates that a statement is continued from the previous line.

The —→► symbol indicates the end of a statement.

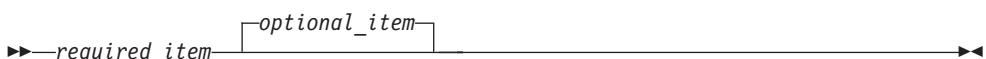
- Required items appear on the horizontal line (the main path).



- Optional items appear below the main path.

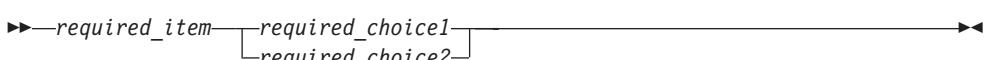


If an optional item appears above the main path, that item has no effect on the execution of the statement and is used only for readability.

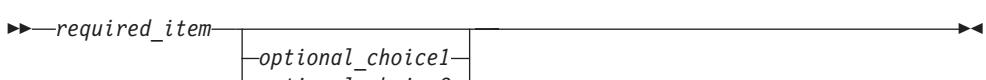


- If you can choose from two or more items, they appear vertically, in a stack.

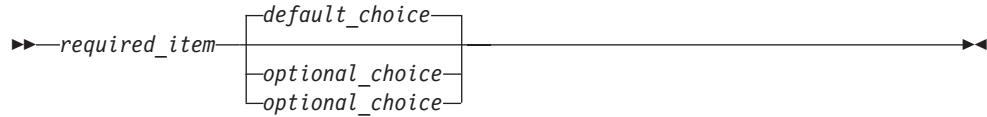
If you *must* choose one of the items, one item of the stack appears on the main path.



If choosing one of the items is optional, the entire stack appears below the main path.



If one of the items is the default, it appears above the main path and the remaining choices are shown below.



- An arrow returning to the left, above the main line, indicates an item that can be repeated.



If the repeat arrow contains a comma, you must separate repeated items with a comma.



A repeat arrow above a stack indicates that you can repeat the items in the stack.

- Keywords appear in uppercase (for example, FROM). They must be spelled exactly as shown. Variables appear in all lowercase letters (for example, *column-name*). They represent user-supplied names or values.
- If punctuation marks, parentheses, arithmetic operators, or other such symbols are shown, you must enter them as part of the syntax.

How to send your comments

Your feedback helps IBM to provide quality information. Please send any comments that you have about this book or other DB2 for OS/390 and z/OS documentation. You can use the following methods to provide comments:

- Send your comments by e-mail to db2pubs@vnet.ibm.com and include the name of the product, the version number of the product, and the number of the book. If you are commenting on specific text, please list the location of the text (for example, a chapter and section title, page number, or a help topic title).
- You can also send comments from the Web. Visit the library Web site at:

www.ibm.com/software/db2zos/library.html

This Web site has a feedback page that you can use to send comments.

- Print and fill out the reader comment form located at the back of this book. You can give the completed form to your local IBM branch office or IBM representative, or you can send it to the address printed on the reader comment form.

Summary of changes to this book

The major changes to this book are:

Chapter 1, “DB2 concepts” includes new descriptions of constraints and character conversion, including Unicode information.

Chapter 2, “Language elements” contains numerous changes to descriptions of data types, special registers, functions, expressions, and predicates. The chapter also contains the descriptions of a new special register (CURRENT APPLICATION ENCODING SCHEME).

Chapter 3, “Functions” includes descriptions of column, scalar, and table functions, including 20 new MQSeries functions. (See Table 28 on page 155 for a list and brief description of all the functions.)

Chapter 4, “Queries” contains changes for fullselect and the select statement.

Chapter 5, “Statements” includes many new statements, as well as changed statements. The new statements are:

- “ALTER FUNCTION (SQL scalar)” on page 408
- “CREATE FUNCTION (SQL scalar)” on page 585
- “DECLARE VARIABLE” on page 739
- “SET CURRENT APPLICATION ENCODING SCHEME” on page 906
- “SET host-variable assignment” on page 918
- “SET transition-variable assignment” on page 924

Statements with new clauses, new values for existing clauses, or other changes include:

- # “ALTER FUNCTION (external)” on page 391
- # “ALTER PROCEDURE (external)” on page 427
- # “ALTER PROCEDURE (SQL)” on page 438
- # “ALTER STOGROUP” on page 444
- # “ALTER TABLE” on page 447
- # “ALTER TABLESPACE” on page 467
- # “CALL” on page 483
- # “COMMENT” on page 493
- # “CONNECT (Type 1)” on page 504
- # “CONNECT (Type 2)” on page 510
- # “CREATE DISTINCT TYPE” on page 522
- # “CREATE FUNCTION (external scalar)” on page 530
- # “CREATE FUNCTION (external table)” on page 553
- # “CREATE GLOBAL TEMPORARY TABLE” on page 595
- # “CREATE INDEX” on page 601
- # “CREATE PROCEDURE (external)” on page 617
- # “CREATE PROCEDURE (SQL)” on page 636
- # “CREATE STOGROUP” on page 648
- # “CREATE TABLE” on page 653
- # “CREATE TABLESPACE” on page 681
- # “CREATE VIEW” on page 711
- # “DECLARE CURSOR” on page 718
- # “DECLARE GLOBAL TEMPORARY TABLE” on page 725
- # “DELETE” on page 742
- # “DROP” on page 763
- # “EXECUTE IMMEDIATE” on page 779

“FETCH” on page 793
“GRANT (distinct type or JAR privileges)” on page 809
“GRANT (package privileges)” on page 816
“GRANT (schema privileges)” on page 819
“INSERT” on page 832
“OPEN” on page 842
“PREPARE” on page 846
“REVOKE” on page 865
“REVOKE (distinct type or JAR privileges)” on page 874
“REVOKE (function or procedure privileges)” on page 876
“SELECT INTO” on page 900
“SET PATH” on page 921
“UPDATE” on page 928

Chapter 6, “SQL procedure statements” includes changes to statements that can be used in SQL procedures.

Appendix C, “SQLCA and SQLDA” describes changes to the SQLDA to support LOBs and distinct types.

Appendix D, “DB2 catalog tables” includes descriptions of several new catalog tables. (See “New and changed catalog tables” on page 1014 for a summary of all catalog table changes.)

Chapter 1. DB2 concepts

Structured query language	2
Static SQL	2
Dynamic SQL	2
Deferred embedded SQL	2
Interactive SQL	3
DB2 Open Database Connectivity (ODBC)	3
DB2 access for Java (JDBC and SQLJ)	3
Schemas	3
Tables	4
Indexes	5
Keys	5
Unique keys	5
Primary keys	5
Parent keys	6
Foreign keys	6
Constraints	6
Unique constraints	6
Referential constraints	7
Check constraints	9
Triggers	9
Storage structures	10
Storage groups	10
Databases	10
Catalog	10
Views	10
Application processes, concurrency, and recovery	11
Locking, commit, and rollback	11
Unit of work	12
Unit of recovery	13
Rolling back work	13
Rolling back all changes	13
Rolling back selected changes using savepoints	14
Packages and application plans	14
Distributed data	14
DRDA access	15
DB2 private protocol access	16
Connection management for DRDA access and DB2 private protocol	17
SQL connection states	18
Application process connection states	19
DB2 private connections	20
When a connection is ended	20
Character conversion	20
Character sets and code pages	22
System CCSIDs	23
Expanding conversions	24
Contracting conversions	25
Other considerations for using UTF-8 and UTF-16	25

Structured query language

Structured query language (SQL) is a standardized language for defining and manipulating data in a relational database. In accordance with the relational model of data, the database is perceived as a set of tables, relationships are represented by values in tables, and data is retrieved by specifying a result table that can be derived from one or more tables. DB2 for OS/390 and z/OS transforms the specification of a result table into a sequence of internal operations that optimize data retrieval. This transformation occurs when the SQL statement is *prepared*. This transformation is also known as *binding*.

All executable SQL statements must be prepared before they can be executed. The result of preparation is the executable or *operational form* of the statement. The method of preparing an SQL statement and the persistence of its operational form distinguish *static SQL* from *dynamic SQL*.

Static SQL

The source form of a *static SQL* statement is embedded within an application program written in a host language such as COBOL. The statement is prepared before the program is executed and the operational form of the statement persists beyond the execution of the program.

Static SQL statements in a source program must be processed before the program is compiled. This processing can be accomplished through the DB2 precompiler or the SQL statement coprocessor. The DB2 precompiler or the coprocessor checks the syntax of the SQL statements, turns them into host language comments, and generates host language statements to invoke DB2.

The preparation of an SQL application program includes precompilation, the preparation of its static SQL statements, and compilation of the modified source program, as described in Part 5 of *DB2 Application Programming and SQL Guide*.

Dynamic SQL

Programs that contain embedded *dynamic SQL* statements must be precompiled like those that contain static SQL, but unlike static SQL, the dynamic statements are constructed and prepared at run time. The source form of a dynamic statement is a character string that is passed to DB2 by the program using the static SQL statement PREPARE or EXECUTE IMMEDIATE. Whether the operational form of the statement is persistent depends on whether dynamic statement caching is enabled. For details on dynamic statement caching, see Part 6 of *DB2 Application Programming and SQL Guide*.

You can execute some SQL statements dynamically with the EXEC SQL utility control statement. See Part 2 of *DB2 Utility Guide and Reference*.

Deferred embedded SQL

A *deferred embedded SQL* statement is neither fully static nor fully dynamic. Like a static statement, it is embedded within an application, but like a dynamic statement, it is prepared during the execution of the application. Although prepared at run time, a deferred embedded SQL statement is processed with bind-time rules such that the authorization ID and qualifier determined at bind time for the plan or package owner are used. Deferred embedded SQL statements are used for DB2 private protocol access to remote data.

Interactive SQL

In this book, *interactive SQL* refers to SQL statements submitted to SPUFI (SQL processor using file input). SPUFI prepares and executes these statements dynamically. For more details about using SPUFI, see Part 1 of *DB2 Application Programming and SQL Guide*.

DB2 Open Database Connectivity (ODBC)

DB2 Open Database Connectivity (DB2 ODBC) is an alternative to using embedded static or dynamic SQL. DB2 ODBC is an application programming interface in which functions are provided to application programs to process SQL statements. The function calls are available only for C and C++ application programs. Through the interface, the application invokes a C function at execution time to connect to the data source, to issue SQL statements, and to get returned data and status information. Unlike using embedded SQL, no precompilation is required. Applications developed using this interface might be executed on a variety of data sources without being compiled against each of the databases. Note that only C and C++ applications can use this interface.

DB2 ODBC provides a consistent interface to query and retrieve system catalog information across the DB2 family of database management systems. This reduces the need to write catalog queries that are specific to each database server. DB2 ODBC can return result sets to those programs.

The *DB2 ODBC Guide and Reference* describes the APIs supported with this interface.

DB2 access for Java (JDBC and SQLJ)

JavaSoft™ JDBC and SQLJ are two methods for accessing DB2 data from the Java® programming language. In general, Java applications use JDBC for dynamic SQL and SQLJ for static SQL.

JDBC is an application programming interface (API) that Java applications can use to access any relational database. JDBC is similar to ODBC and is based on the X/Open SQL Call Level Interface specification.

SQLJ is an API that provides support for embedded static SQL in Java applications. Because DB2 for OS/390 SQLJ support includes JDBC, SQLJ applications can also execute dynamic SQL statements through JDBC.

The *DB2 Application Programming Guide and Reference for Java* describes the APIs supported with these interfaces.

Schemas

A *schema* is a collection of named objects. The objects that a schema can contain include distinct types, functions, stored procedures, and triggers. An object is assigned to a schema when it is created.

The *schema name* of the object determines the schema to which the object belongs. When a distinct type, function, or trigger is created, it is given a qualified, two-part name. The first part is the schema name (or the qualifier), which is either implicitly or explicitly specified. The second part is the name of the object. When a stored procedure is created, it is given a three-part name. The first part is a location

name, which is implicitly or explicitly specified, the second part is the schema name, which is implicitly or explicitly specified, and the third part is the name of the object.

Schemas extend the concept of qualifiers for tables, views, indexes, and aliases to enable the qualifiers for distinct types, functions, stored procedures, and triggers to be called schema names.

Tables

Tables are logical structures maintained by DB2. Tables are made up of *columns* and *rows*. There is no inherent order of the rows within a table. At the intersection of every column and row is a specific data item called a *value*. A *column* is a set of values of the same type. A *row* is a sequence of values such that the *n*th value is a value of the *n*th column of the table. Every table must have one or more columns, but the number of rows can be zero.

Some types of tables include:

base table

A table created with the SQL statement CREATE TABLE and used to hold persistent user data.

auxiliary table

A table created with the SQL statement CREATE AUXILIARY TABLE and used to hold the data for a column that is defined in a base table.

temporary table

A table defined by either the SQL statement CREATE GLOBAL TEMPORARY TABLE (a created temporary table) or DECLARE GLOBAL TEMPORARY TABLE (a declared temporary table) and used to hold data temporarily, such as the intermediate results of SQL transactions. Both created temporary tables and declared temporary tables persist only as long as the application process. The description of a created temporary table is stored in the DB2 catalog and the description is shareable across application processes while the description of a declared temporary table is neither stored nor shareable. Thus, each application process might refer to the same declared temporary table but have its own unique description of it. For a complete comparison of the two types of temporary tables, including how they differ from base tables, see Part 2 (Volume 1) of *DB2 Administration Guide*.

result table

A set of rows that DB2 selects or generates from one or more base tables.

empty table

A table with zero rows.

sample table

One of several tables sent with the DB2 licensed program that contains sample data. Many examples in this book are based on sample tables. See Appendix A of *DB2 Application Programming and SQL Guide* for a description of the sample tables.

Indexes

An *index* is an ordered set of pointers to rows of a base table or an auxiliary table. Each index is based on the values of data in one or more columns. An index is an object that is separate from the data in the table. When you define an index using the CREATE INDEX statement, DB2 builds this structure and maintains it automatically.

Indexes can be used by DB2 to improve performance and ensure uniqueness. In most cases, access to data is faster with an index. A table with a unique index cannot have rows with identical keys. For more details on designing indexes and on their uses, see *An Introduction to DB2 for OS/390*.

Keys

A *key* is one or more columns that are identified as such in the description of a table, an index, or a referential constraint. Referential constraints are described in . The same column can be part of more than one key. A key composed of more than one column is called a composite key.

A *composite key* is an ordered set of columns of the same table. The ordering of the columns is not constrained by their ordering within the table. The term *value*, when used with respect to a composite key, denotes a *composite value*. Thus, a rule, such as “the value of the foreign key must be equal to the value of the parent key”, means that each component of the value of the foreign key must be equal to the corresponding component of the value of the parent key.

Unique keys

A *unique key* is a key that is constrained so that no two of its values are equal. DB2 enforces the constraint during the execution of the LOAD utility and the SQL INSERT and UPDATE statements. The mechanism used to enforce the constraint is a *unique index*. Thus, every unique key is a key of a unique index. Such an index is also said to have the UNIQUE attribute.

The columns of a unique key cannot contain null values.

A unique key can be defined using the UNIQUE clause of the CREATE TABLE or ALTER TABLE statement. When a unique key is defined in a CREATE TABLE statement, the table is marked unavailable until the unique index is created by the user. However, if the CREATE TABLE statement is processed by the schema processor, DB2 automatically creates the unique index. When a unique key is defined in an ALTER TABLE statement, a unique index must already exist on the columns of that unique key.

Primary keys

A *primary key* is a unique key that is a part of the definition of a table. A table can have only one primary key, and the columns of a primary key cannot contain null values. Primary keys are optional and can be defined in CREATE TABLE or ALTER TABLE statements.

The unique index on a primary key is called a *primary index*. When a primary key is defined in a CREATE TABLE statement, the table is marked unavailable until the primary index is created by the user unless the CREATE TABLE statement is processed by the schema processor. In that case, DB2 automatically creates the primary index.

DB2 concepts

When a primary key is defined in an ALTER TABLE statement, a unique index must already exist on the columns of that primary key. This unique index is designated as the primary index.

Parent keys

A *parent key* is either a primary key or a unique key in the parent table of a referential constraint. The values of a parent key determine the valid values of the foreign key in the constraint.

Foreign keys

A *foreign key* is a key that is specified in the definition of a referential constraint using the CREATE or ALTER statement. A foreign key refers to or is related to a specific parent key. A table can have zero or more foreign keys. The value of a composite foreign key is null if any component of the value is null.

Constraints

Constraints are rules that control values in columns to prevent duplicate values or set restrictions on data added to a table.

Constraints fall into the following three types:

- A *unique constraint* is a rule that prevents duplicate values in one or more columns in a table. Unique constraints are unique and primary keys. For example, a unique constraint could be defined on a supplier identifier in a supplier table to ensure that the same supplier identifier applies to one single supplier.
- A *referential constraint* is a rule about values in one or more columns in one or more tables. For example, for a set of tables sharing information about a corporation's suppliers, a supplier's ID may change. A referential constraint could be defined stating that the ID of the supplier in the table must match a supplier ID in the supplier information. If the new ID for the supplier is in the supplier information, this constraint prevents inserts, updates, or deletes caused by missing supplier information.
- A *check constraint* sets restrictions on data added to a specific table. For example, the constraint could be added to define a salary level for an employee to never be less than a stated amount when salary data is added or updated in a table for personnel information.

Unique constraints

A *unique constraint* is a rule that the values of a key are valid only if they are unique in a table. Unique constraints are optional and can be defined in the CREATE TABLE or ALTER TABLE statements with the PRIMARY KEY clause or the UNIQUE clause. The columns specified in a unique constraint must be defined as NOT NULL. A unique index enforces the uniqueness of the key during changes to the columns of the unique constraint.

A table can have an arbitrary number of unique constraints, with at most one unique constraint defined as a primary key. A table cannot have more than one unique constraint on the same set of columns.

A unique constraint that is referenced by the foreign key of a referential constraint is called the parent key.

Referential constraints

Referential integrity is the state in which all values of all foreign keys at a given DB2 are valid. A *referential constraint* is the rule that the non-null values of a foreign key are valid only if they also appear as values of a parent key. The table that contains the parent key is called the *parent table* of the referential constraint, and the table that contains the foreign key is a *dependent* of that table.

Referential constraints are optional and can be defined using CREATE TABLE and ALTER TABLE statements. Refer to Part 2 (Volume 1) of *DB2 Administration Guide* for examples.

DB2 enforces referential constraints when:

- An INSERT statement is applied to a dependent table.
- An UPDATE statement is applied to a foreign key of a dependent table.
- An UPDATE statement is applied to the parent key of a parent table.
- A DELETE statement is applied to a parent table. All affected referential constraints and all delete rules of all affected relationships must be satisfied in order for the delete operation to succeed.
- The LOAD utility with the ENFORCE CONSTRAINTS option is run on a dependent table.

The order in which referential constraints are enforced is undefined. To ensure that the order does not affect the result of the operation, there are restrictions on the definition of delete rules and on the use of certain statements. The restrictions are specified in the descriptions of the SQL statements CREATE TABLE, ALTER TABLE, INSERT, UPDATE, and DELETE.

The rules of referential integrity involve the following concepts and terminology:

parent key

A primary key or a unique key of a referential constraint.

parent table

A table that is a parent in at least one referential constraint. A table can be defined as a parent in an arbitrary number of referential constraints.

dependent table

A table that is a dependent in at least one referential constraint. A table can be defined as a dependent in an arbitrary number of referential constraints. A dependent table can also be a parent table.

descendent table

A table that is a dependent of another table or a table that is a dependent of a descendent table.

referential cycle

A set of referential constraints in which each associated table is a descendent of itself.

parent row

A row that has at least one dependent row.

dependent row

A row that has at least one parent row.

descendent row

A row that is dependent on another row or a row that is a dependent of a descendent row.

self-referencing row

A row that is a parent of itself.

self-referencing table

A table that is both parent and dependent in the same referential constraint.

The constraint is called a *self-referencing constraint*.

The following rules provide referential integrity:

insert rule

A non-null insert value of the foreign key must match some value of the parent key of the parent table.

update rule

A non-null update value of the foreign key must match some value of the parent key of the parent table.

delete rule

The choices when the referential constraint is defined are RESTRICT, NO ACTION, CASCADE, or SET NULL. SET NULL can be specified only if some column of the foreign key allows null values.

The delete rule of a referential constraint applies when a row of the parent table is deleted. More precisely, the rule applies when a row of the parent table is the object of a delete or propagated delete operation and that row has dependents in the dependent table of the referential constraint. Let P denote the parent table, let D denote the dependent table, and let p denote a parent row that is the object of a delete or propagated delete operation. If the delete rule is:

- RESTRICT or NO ACTION, an error occurs and no rows are deleted.
- CASCADE, the delete operation is propagated to the dependent rows of p in D.
- SET NULL, each nullable column of the foreign key of each dependent row of p in D is set to null.

Each referential constraint in which a table is a parent has its own delete rule, and all applicable delete rules are used to determine the result of a delete operation. Thus, a row cannot be deleted if it has dependents in a referential constraint with a delete rule of RESTRICT or NO ACTION or the deletion cascades to any of its descendants that are dependents in a referential constraint with the delete rule of RESTRICT or NO ACTION.

The deletion of a row from parent table P involves other tables and can affect rows of these tables:

- If D is a dependent of P and the delete rule is RESTRICT or NO ACTION, D is involved in the operation but is not affected by the operation.
- If D is a dependent of P and the delete rule is SET NULL, D is involved in the operation and rows of D might be updated during the operation.
- If D is a dependent of P and the delete rule is CASCADE, D is involved in the operation and rows of D might be deleted during the operation. If rows of D are deleted, the delete operation on P is said to be propagated to D. If D is also a parent table, the actions described in this list apply, in turn, to the dependents of D.

Any table that can be involved in a delete operation on P is said to be *delete-connected* to P. Thus, a table is delete-connected to table P if it is a dependent of P or a dependent of a table to which delete operations from P cascade.

Check constraints

A *check constraint* is a rule that specifies the values allowed in one or more columns of every row of a table. Check constraints are optional and can be defined using the SQL statements CREATE TABLE and ALTER TABLE. The definition of a check constraint is a restricted form of a search condition. One of the restrictions is that a column name in a check constraint on table T must identify a column of T. See Part 2 of *DB2 Application Programming and SQL Guide* for examples.

A table can have an arbitrary number of check constraints. DB2 enforces the constraints when:

- A row is inserted into the table.
- A row of the table is updated.
- The LOAD utility with the ENFORCE CONSTRAINTS option is used to populate the table.

A check constraint is enforced by applying its search condition to each row that is inserted, updated, or loaded. An error occurs if the result of the search condition is **false** for any row.

Triggers

A *trigger* defines a set of actions that are executed when a delete, insert, or update operation occurs on a specified table. When such an SQL operation is executed, the trigger is said to be *activated*.

Triggers can be used along with referential constraints and check constraints to enforce data integrity rules. Triggers are more powerful than constraints because they can also be used to cause updates to other tables, automatically generate or transform values for inserted or updated rows, or invoke functions that perform operations both inside and outside of DB2. For example, instead of preventing an update to a column if the new value exceeds a certain amount, a trigger can substitute a valid value and send a notice to an administrator about the invalid update.

Triggers are useful for defining and enforcing business rules that involve different states of the data, for example, limiting a salary increase to 10%. Such a limit requires comparing the value of a salary before and after an increase. For rules that do not involve more than one state of the data, consider using referential and check constraints.

Triggers also move the application logic that is required to enforce business rules into the database, which can result in faster application development and easier maintenance. With the logic in the database, for example, the previously mentioned limit on increases to the salary column of a table, DB2 checks the validity of the changes that any application makes to the salary column. In addition, the application programs do not need to be changed when the logic changes.

Triggers are optional and are defined using the CREATE TRIGGER statement.

For information on using triggers, see Part 2 of *DB2 Application Programming and SQL Guide*.

Storage structures

In DB2, a *storage structure* is a set of one or more VSAM data sets that hold DB2 tables or indexes. A storage structure is also called a *page set*. A storage structure can be one of the following:

table space

A table space can hold one or more base tables, or one auxiliary table. All tables are kept in table spaces. A table space can be defined using the CREATE TABLESPACE statement.

index space

An index space contains a single index. An index space is defined when the index is defined using the CREATE INDEX statement.

Storage groups

Defining and deleting the data sets of a storage structure can be left to DB2. If it is left to DB2, the storage structure has an associated *storage group*. The storage group is a list of DASD volumes on which DB2 can allocate data sets for associated storage structures. The association between a storage structure and its storage group is explicitly or implicitly defined by the statement that created the storage structure.

Alternatively, Storage Management Subsystem (SMS) can be used to manage DB2 data sets. Refer to Part 2 (Volume 1) of *DB2 Administration Guide* for more information.

Databases

In DB2, a *database* is a set of table spaces and index spaces. These index spaces contain indexes on the tables in the table spaces of the same database. Databases are defined using the CREATE DATABASE statement and are primarily used for administration. Whenever a table space is created, it is explicitly or implicitly assigned to an existing database.

Catalog

Each DB2 maintains a set of tables that contain information about the data under its control. These tables are collectively known as the *catalog*. The *catalog tables* contain information about DB2 objects such as tables, views, and indexes. In this book, “catalog” refers to a DB2 catalog unless otherwise indicated. In contrast, the catalogs maintained by access method services are known as “integrated catalog facility catalogs”.

Tables in the catalog are like any other database tables with respect to retrieval. If you have authorization, you can use SQL statements to look at data in the catalog tables in the same way that you retrieve data from any other table in the system. Each DB2 ensures that the catalog contains accurate descriptions of the objects that the DB2 controls.

Views

A view provides an alternative way of looking at the data in one or more tables. A *view* is a named specification of a result table. The specification is an SQL SELECT statement that is effectively executed whenever the view is referenced in an SQL statement. At any time, the view consists of the rows that would result if the

fullselect were executed. Thus, a view can be thought of as having columns and rows just like a base table. However, columns added to the base tables after the view is defined do not appear in the view. For retrieval, all views can be used like base tables. Whether a view can be used in an insert, update, or delete operation depends on its definition, as described in “CREATE VIEW” on page 711.

Views can be used to control access to a table and make data easier to use. Access to a view can be granted without granting access to the table. The view can be defined to show only portions of data in the table. A view can show summary data for a given table, combine two or more tables in meaningful ways, or show only the selected rows that are pertinent to the process using the view.

Example: The following SQL statement defines a view named XYZ. The view represents a table whose columns are named EMPLOYEE and WHEN_HIRED. The data in the table comes from the columns EMPNO and HIREDATE of the sample table DSN8710.EMP. The rows from which the data is taken are for employees in departments A00 and D11.

```
CREATE VIEW XYZ (EMPLOYEE, WHEN_HIRED)
  AS SELECT EMPNO, HIREDATE
    FROM DSN8710.EMP
   WHERE WORKDEPT IN ('A00', 'D11');
```

An index cannot be created for a view. However, an index created for a table on which a view is based might improve the performance of operations on the view. The column of a view inherits its attributes (such as data type, precision, and scale) from the table or view column, constant, function, or expression from which it is derived. In addition, a view column that maps back to a base table column inherits any default values or constraints specified for that column of the base table. For example, if a view includes a foreign key of its base table, insert and update operations using that view are subject to the same referential constraint as the base table. Likewise, if the base table of a view is a parent table, delete operations using that view are subject to the same rules as delete operations on the base table. See the description of “INSERT” on page 832 and “UPDATE” on page 928 for restrictions that apply to views with derived columns. For information on referential constraints, see .

Read-only views cannot be used for insert, update, and delete operations. For a discussion of read-only views, see “CREATE VIEW” on page 711.

The definition of a view is stored in the DB2 catalog. An SQL DROP VIEW statement can drop a view, and the definition of the view is removed from the catalog. The definition of a view is also removed from the catalog when any view or base table on which the view depends is dropped.

Application processes, concurrency, and recovery

All SQL programs execute as part of an *application process*. An application process involves the execution of one or more programs, and is the unit to which DB2 allocates resources and locks. Different application processes might involve the execution of different programs, or different executions of the same program. The means of initiating and terminating an application process are dependent on the environment.

Locking, commit, and rollback

More than one application process might request access to the same data at the same time. Furthermore, under certain circumstances, an SQL statement can

DB2 concepts

execute concurrently with a utility on the same table space¹. Locking is used to maintain data integrity under such conditions, preventing, for example, two application processes from updating the same row of data simultaneously. See Part 5 (Volume 2) of *DB2 Administration Guide* for more information about DB2 locks.

DB2 implicitly acquires locks to prevent uncommitted changes made by one application process from being perceived by any other. DB2 will implicitly release all locks it has acquired on behalf of an application process when that process ends, but an application process can also explicitly request that locks be released sooner. A *commit* operation releases locks acquired by the application process and commits database changes made by the same process.

DB2 provides a way to *back out* uncommitted changes made by an application process. This might be necessary in the event of a failure on the part of an application process, or in a *deadlock* situation. An application process, however, can explicitly request that its database changes be backed out. This operation is called *rollback*.

The interface used by an SQL program to explicitly specify these commit and rollback operations depends on the environment. If the environment can include recoverable resources other than DB2 databases, the SQL COMMIT and ROLLBACK statements cannot be used. Thus, these statements cannot be used in an IMS, CICS, or WebSphere™ environment. Refer to Part 4 of *DB2 Application Programming and SQL Guide* for more details.

Unit of work

A *unit of work* is a recoverable sequence of operations within an application process. A unit of work is sometimes called a *logical unit of work*. At any time, an application process has a single unit of work, but the life of an application process can involve many units of work as a result of commit or full rollback operations.

A unit of work is initiated when an application process is initiated. A unit of work is also initiated when the previous unit of work is ended by something other than the end of the application process. A unit of work is ended by a commit operation, a full rollback operation, or the end of an application process. A commit or rollback operation affects only the database changes made within the unit of work it ends. While these changes remain uncommitted, other application processes are unable to perceive them unless they are running with an isolation level of uncommitted read. The changes can still be backed out. Once committed, these database changes are accessible by other application processes and can no longer be backed out by a rollback. Locks acquired by DB2 on behalf of an application process that protects uncommitted data are held at least until the end of a unit of work.

The initiation and termination of a unit of work define *points of consistency* within an application process. A point of consistency is a claim by the application that the data is consistent. For example, a banking transaction might involve the transfer of funds from one account to another. Such a transaction would require that these funds be subtracted from the first account, and added to the second. Following the subtraction step, the data is inconsistent. Only after the funds have been added to the second account is consistency reestablished. When both steps are complete,

1. See the description of a table space under “Storage structures” on page 10. Concurrent execution of SQL statements and utilities is discussed in Part 5 (Volume 2) of *DB2 Administration Guide*.

the commit operation can be used to end the unit of work, thereby making the changes available to other application processes.

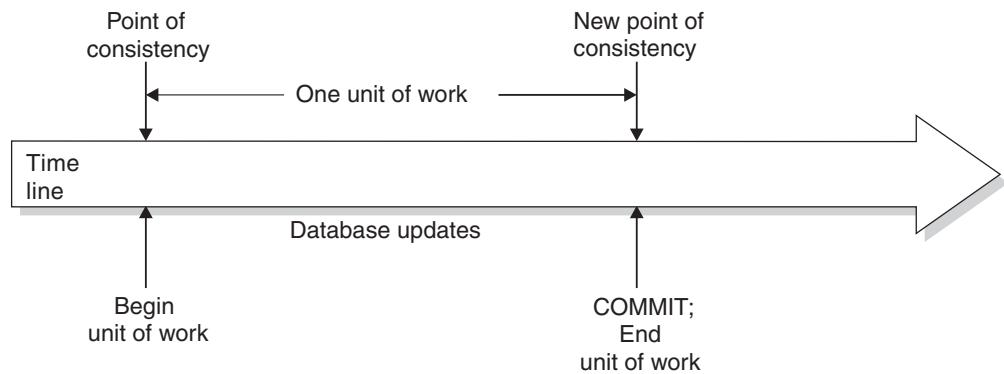


Figure 1. Unit of work with a commit operation

Unit of recovery

A DB2 unit of recovery is a recoverable sequence of operations executed by DB2 for an application process. If a unit of work involves changes to other recoverable resources, the unit of work will be supported by other units of recovery. If relational databases are the only recoverable resources used by the application process, then the scope of the unit of work and the unit of recovery are the same and either term can be used.

Rolling back work

DB2 can back out all changes made in a unit of recovery or only selected changes. Only backing out all changes results in a point of consistency.

Rolling back all changes

The SQL ROLLBACK statement without the TO SAVEPOINT clause specified causes a full rollback operation. If such a rollback operation is successfully executed, DB2 backs out uncommitted changes to restore the data consistency that it assumes existed when the unit of work was initiated. That is, DB2 *undoes* the work, as shown in the diagram below:

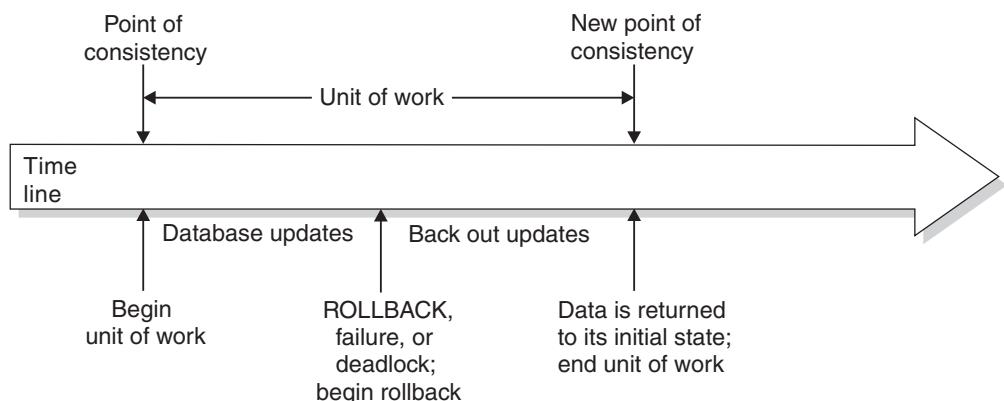


Figure 2. Rolling back all changes from a unit of work

Rolling back selected changes using savepoints

A *savepoint* represents the state of data at some particular time during a unit of work. An application process can set savepoints within a unit of work, and then as logic dictates, roll back only the changes that were made after a savepoint was set. For example, part of a reservation transaction might involve booking an airline flight and then a hotel room. If a flight gets reserved but a hotel room cannot be reserved, the application process might want to undo the flight reservation without undoing any database changes made in the transaction prior to making the flight reservation. SQL programs can use the SQL SAVEPOINT statement to set savepoints, the SQL ROLLBACK statement with the TO SAVEPOINT clause to undo changes to a specific savepoint or the last savepoint that was set, and the SQL RELEASE SAVEPOINT statement to delete a savepoint.

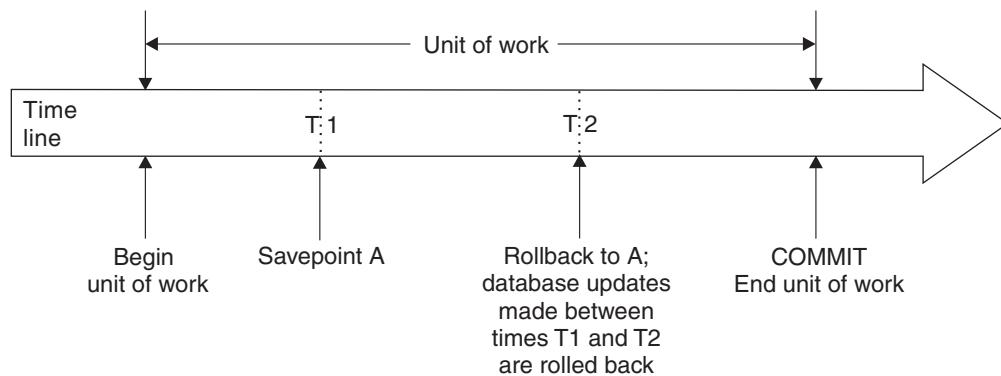


Figure 3. Rolling back changes to a savepoint within a unit of work

Packages and application plans

A *package* contains control structures used to execute SQL statements. Packages are produced during program preparation. The control structures can be thought of as the bound or operational form of SQL statements taken from a *database request module (DBRM)*. The DBRM contains SQL statements extracted from the source program during program preparation. All control structures in a package are derived from the SQL statements embedded in a single source program.

An *application plan* relates an application process to a local instance of DB2, specifies processing options, and contains one or both of the following *elements*:

- A list of package names
- The bound form of SQL statements taken from one or more DBRMs

Every DB2 application requires an application plan. Plans and packages are created using the DB2 subcommands BIND PLAN and BIND PACKAGE, respectively, as described in *DB2 Command Reference*. See Part 5 of *DB2 Application Programming and SQL Guide* for a description of program preparation and identifying packages at run time. Refer to “SET CURRENT PACKAGESET” on page 912 for rules regarding the selection of a plan element.

Distributed data

A DB2 application program can use SQL to access data at other database management systems (DBMSs) other than the DB2 at which the application's plan is bound. This DB2 is known as the *local DB2*. The local DB2 and the other DBMSs are called *database servers*. Any server other than the local DB2 is considered a *remote server*, and access to its data is a distributed operation. The recommended

method of accessing data at remote database servers is “DRDA access.” “DB2 private protocol access” on page 16 is also available but is not recommended.

For servers that support the two-phase commit process, both methods allow for updating data at several remote locations within the same unit of work. To obtain the more restrictive level of function available at DB2 Version 2 Release 3, refer to 16. Table 1 summarizes the main differences between DRDA access and DB2 private protocol access.

Table 1. Differences between DRDA access and DB2 private protocol access

Item	DRDA access	DB2 private protocol access
Program preparation	Requires a remote BIND of packages	A remote BIND is not applicable
Plan members	Can use in packages only	Can use in packages or DBRMs bound directly to the plan
Processing of embedded statements	Processed as static SQL	Processed as deferred embedded SQL. For a definition, see “Deferred embedded SQL” on page 2.
Servers	Can use any server that uses the DRDA protocols	Can use DB2 for OS/390 and z/OS servers only
SQL statements	Can use most SQL statements supported by the system that executes the statement (for details, see <i>DB2 Application Programming and SQL Guide</i>)	Limited to SQL INSERT, UPDATE, and DELETE statements, and to statements supporting SELECT
Connection management	Three-part names and aliases can be used to refer to objects at another server if the package was bound with bind option DBPROTOCOL(DRDA) implicitly or explicitly specified. Otherwise, the CONNECT statement is used to connect an application process to a server.	Three-part names and aliases are used to refer to objects at another server.

Common restrictions: IMS and CICS applications are restricted to read-only operations at a remote site if:

- Its database server does not support two-phase commit.
- It uses DB2 private protocol access to a DB2 Version 2 Release 3. (DB2 private protocol access from a DB2 Version 3 or subsequent release requester to a DB2 Version 2 Release 2 server is not supported).

See Part 4 of *DB2 Application Programming and SQL Guide* for more details about common restrictions.

DRDA access

DRDA® access supports the execution of dynamic SQL statements and SQL statements that satisfy all the following conditions:

- The static statements appear in a package bound to an accessible server.
- The statements are executed using that package.

DB2 concepts

- The objects involved in the execution of the statements are at the server where the package is bound. If the server is a DB2 subsystem, three-part names and aliases can be used to refer to another DB2 server.

DRDA access can be used in application programs by coding explicit CONNECT statements or by coding three-part names and specifying the DBPROTOCOL(DRDA) bind option.

DRDA access is based on a set of protocols known as *Distributed Relational Database Architecture* (DRDA). (These protocols are documented by the Open Group Technical Standard in *DRDA Version 2 Vol. 1: Distributed Relational Database Architecture (DRDA)*.) DRDA communication conventions are invisible to DB2 applications, and allow a DB2 to bind and rebinding packages at other servers and to execute the statements in those packages. See Part 5 of *DB2 Application Programming and SQL Guide* for the steps involved in binding packages and plans. If the server supports the two-phase commit process, use the CONNECT (Type 2) statement and other connection management statements such as RELEASE.

A system that uses DRDA can request the execution of SQL statements at any DB2. Preparing DB2 for incoming SQL requests is discussed in Part 3 of *DB2 Installation Guide*.

When preparing a program for use at a server other than DB2, observe the following rules:

- For SQL statements processed by the server, use the SQL syntax and semantic rules of that server. For other statements, use the DB2 rules. For a list of where statements are processed, see Appendix B, “Characteristics of SQL statements in DB2 for OS/390 and z/OS,” on page 969.
- Use the precompiler option SQL(ALL) when precompiling the program. Statements that violate DB2 rules are flagged, but their detection does not prevent the creation of a DBRM.

For more information, refer to the *Distributed Relational Database Library*.

Remote unit of work is a restricted level of function that is available by DRDA access when the CONNECT(1) precompiler option is specified. An application process can have only one connection at a time and cannot connect to a new server until it executes a commit or rollback operation. This restricts the situations in which the CONNECT statement can be executed. See “CONNECT” on page 501 for more information about these restrictions. For more details about CONNECT (Type 1) and a description of the connection states, refer to “CONNECT (Type 1)” on page 504.

DB2 private protocol access

DB2 private protocol access allows one DB2 to execute a range of statements at another DB2.

A statement is executed using DB2 private protocol access if it refers to objects that are not at the current server and is implicitly or explicitly bound with DBPROTOCOL(PRIVATE). The *current server* is the DBMS to which an application is actively connected. DB2 private protocol access uses *DB2 private connections*. The statements that can be executed are SQL INSERT, UPDATE, and DELETE, and SELECT statements with their associated SQL OPEN, FETCH, and CLOSE statements. “When an application process has a current server” on page 502 describes what happens when an application process has a current server.

In a program running under DB2, a *three-part name* or an *alias* can refer to a table or view at another DB2. The location name identifies the other DB2 to the DB2 server. A three-part name has the form:

location-name.aaaaaa.ssssss

where *aaaaaa.ssssss* uniquely identifies the object at the server named *location-name*. For example, the name USIBMSTODB21.DSN8710.EMP refers to a table named DSN8710.EMP at the server whose location name is USIBMSTODB21. Location naming conventions are described in “Location identifiers” on page 34. Preparing DB2 for incoming SQL requests is discussed in Part 3 of *DB2 Installation Guide*.

Alias names have the same allowable forms as table or view names. The name can refer to a table or view at the current server or to a table or view elsewhere. For more on aliases, see “Aliases and synonyms” on page 41. For more on three-part names, and on SQL naming conventions in general, see “Naming conventions” on page 34.

DRDA access has some significant advantages over DB2 private protocol:

- DRDA access uses a more compact format for sending data over the network and thus improves performance on slow network links.
- Queries sent by DB2 private protocol access are bound at the server whenever they are first executed in a unit of work. Repeated binds can reduce the performance of a query that is executed often.

A DBRM for statements executed by DRDA access is bound to a package at the server once. Those statements can include PREPARE and EXECUTE so that your application can accept dynamic statements to be executed at the server. But binding the package is an extra step in program preparation.

- You can use stored procedures with DRDA access.

While a stored procedure is running, it requires no message traffic over the network and thus reduces the biggest hindrance to high performance for distributed data.

Connection management for DRDA access and DB2 private protocol

An *SQL connection* is an association between an application process and a local or remote database server. SQL connections can be managed by the application or by using bind options. At any time:

- An application process is in the *connected* or *unconnected* state and has a set of zero or more SQL connections. Each SQL connection of an application process is uniquely identified by the name of the server of the SQL connection.
- An SQL connection is in one of the following states:
 - Current and held
 - Current and release pending
 - Dormant and held
 - Dormant and release pending

Initial state of an application process: An application process is initially in the connected state and has exactly one SQL connection. The server of that connection is the local DB2 subsystem. The initial state of an SQL connection is current and held.

The following diagram shows the state transitions:

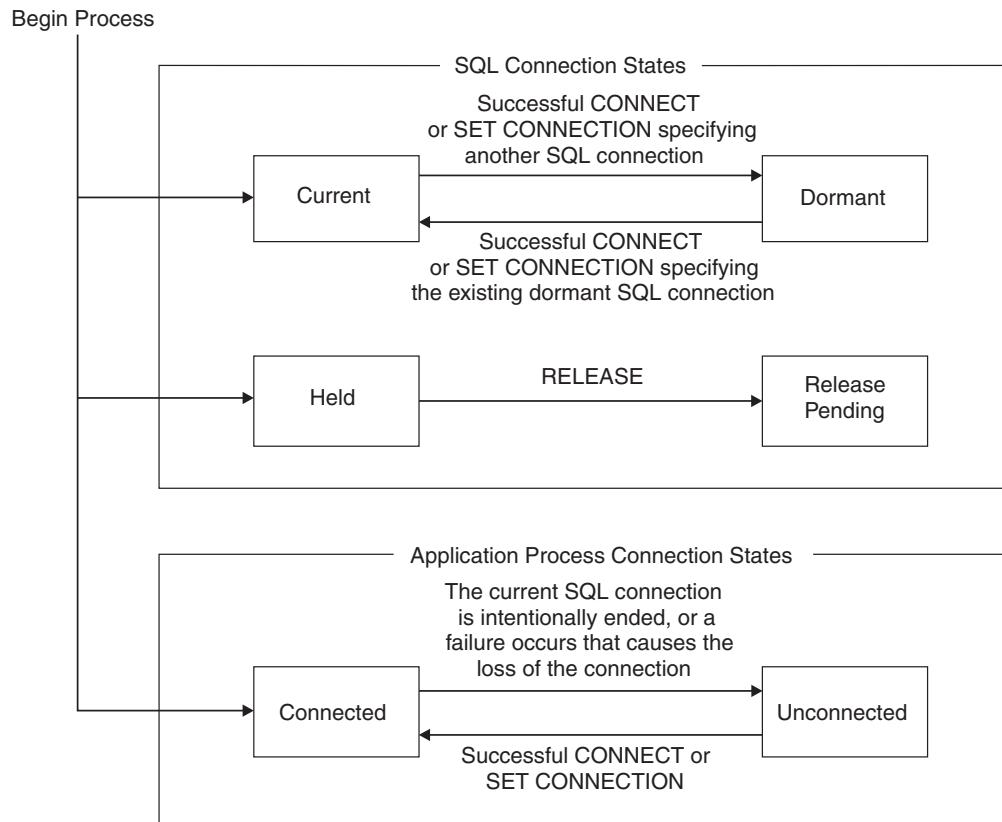


Figure 4. SQL connection and application process connection state transitions

SQL connection states

If an application process executes a CONNECT TO statement and the specified location is known to the local DB2 and is not in the set of existing connections of the application process, the location is added to the set of connections and the connection is placed in the current and held state. If the specified location is the current SQL connection of the application process, and if the SQLRULES(DB2) bind option is in effect, the states of all existing connections remain the same.

An SQL connection in the dormant state is placed in the current state using:

- The SET CONNECTION statement, or
- The CONNECT statement, if the SQLRULES(DB2) bind option is in effect.

When an SQL connection is placed in the current state, the previous current SQL connection, if any, is placed in the dormant state. No more than one SQL connection in the set of existing connections of an application process can be current at any time. Changing the state of an SQL connection from current to dormant or from dormant to current has no effect on its held or release pending status.

An SQL connection is placed in the release pending status by the RELEASE statement. When an application process executes a commit operation, every release pending connection of the process is ended. Changing the state of an SQL connection from held to release pending has no effect on its current or dormant state. Thus, an SQL connection in the release pending status can still be used until the next commit operation. Likewise, DB2 private connections in the release

pending status can be used until the next commit operation. There is no way to change the state of a connection from release pending to held.

Application process connection states

A different server can be established by the explicit or implicit execution of a CONNECT statement. The following rules apply:

- An application process cannot have more than one SQL connection to the same database server at the same time.
- When an application process executes a SET CONNECTION statement, the specified location name must be an existing SQL connection in the set of connections of the application process.
- When an application process executes a CONNECT TO statement and the SQLRULES(STD) bind option is in effect, the specified location must not be an existing SQL connection in the set of connections of the application process.

If an application process has a current SQL connection, the application process is in the *connected* state. The CURRENT SERVER special register contains the name of the server of the current SQL connection. The application process can execute SQL statements that refer to objects managed by that server. If the server is a DB2 subsystem, the application process can also execute certain SQL statements that refer to objects managed by a DB2 subsystem with which that server can establish a connection.

An application process in the unconnected state enters the connected state when it successfully executes a CONNECT or SET CONNECTION statement.

If an application process does not have a current SQL connection, the application process is in the *unconnected* state. The CURRENT SERVER special register contains blanks. The only SQL statements that can be executed successfully at the requester are CONNECT, SET CONNECTION, RELEASE, COMMIT, ROLLBACK, and local SET statements. COMMIT and ROLLBACK are also processed by an server. If the application process is in the unconnected state, the server that processes a COMMIT or ROLLBACK is the local DB2.

An application process in the connected state enters the unconnected state when its current SQL connection is intentionally ended or the execution of an SQL statement is unsuccessful because of a failure that causes a rollback operation at the server and loss of the SQL connection. SQL connections are intentionally ended when an application process successfully executes a commit operation and any of the following apply:

- The connection is in the release pending status
- The connection is not in the release pending status but it is a remote connection and:
 - The DISCONNECT(AUTOMATIC) bind option is in effect, or
 - The DISCONNECT(CONDITIONAL) bind option is in effect and an open WITH HOLD cursor is not associated with the connection.

A connect (type 1) statement is implicitly executed when an application process executes an SQL statement other than COMMIT, CONNECT TO, CONNECT RESET, SET CONNECTION, RELEASE, or ROLLBACK and if all of the following conditions apply:

- The CURRENTSERVER bind option was specified when creating the application plan of the application process and the identified server is not the local DB2.
- An explicit CONNECT statement has not already been successfully or unsuccessfully executed by the application process.

DB2 concepts

- An implicit connection has not already been successfully or unsuccessfully executed by the application process. An implicit connection occurs as the result of execution of an SQL statement that contains a three-part name in a package that is bound with the DBPROTOCOL(DRDA) option.

If the implicit CONNECT fails, the application process is in the *unconnected* state.

DB2 private connections

When the server is a DB2 subsystem, DB2 private connections are allocated as necessary to support references to objects at other DB2 subsystems. Like SQL connections, DB2 private connections are initially in the held state and can be placed in the release pending status.

An application process cannot have an explicit SQL connection and a DB2 private connection to the same DB2 subsystem at the same time. However, an implicit SQL connection and a DB2 private connection can exist concurrently. Accordingly:

- CONNECT TO *x* fails if the application process has a DB2 private connection to *x*, and
- An attempt to allocate a DB2 private connection to *x* fails if the application process has an explicit SQL connection to *x*.
- An implicit SQL connection through a three-part name is successful if the application process has a DB2 private connection to *x*, and
- An attempt to allocate a DB2 private connection to *x* is successful if the application process has an implicit SQL connection to *x*.

When a connection is ended

When a connection is ended, all resources that were acquired by the application process through the connection and all resources that were used to create and maintain the connection are deallocated. In the case of an SQL connection to a DB2 subsystem, the resources acquired can include DB2 private connections.

When the SQL connection is ended, such DB2 private connections are also ended. This is true even if the DB2 subsystem is the local DB2. For example, assume that an application process implicitly connected to the local DB2 used DB2 private protocol access to open a cursor at another DB2. If the application process executes a RELEASE CURRENT statement, that cursor is closed at the commit operation, unless the cursor has an attribute of WITH HOLD.

A connection can also be ended as a result of a communications failure in which case the application process is placed in the unconnected state. All connections of an application process are ended when the process terminates.

Character conversion

A *string* is a sequence of bytes that can represent characters. Within a string, all the characters are represented by a common encoding representation. In some cases, it might be necessary to convert these characters to a different encoding representation. The process of conversion is known as *character conversion*.

In client/server environments, character conversion can occur when an SQL statement is executed remotely. Consider, for example, these two cases:

- The values of host variables sent from the requester to the current server
- The values of result columns sent from the current server to the requester

In either case, the string could have a different representation at the sending and receiving systems. Conversion can also occur during string operations on the same system.

In a local environment, character conversion can occur when:

- An overriding CCSID is specified in the SQLDA (see “SQL descriptor area (SQLDA)” on page 986).
For languages other than REXX, the CCSID is in the SQLNAME field. For REXX, the CCSID is in the SQLCCSID field.
- A mixed character string is assigned to an SBCS column or host variable.

Most users do not need a knowledge of character conversion. When character conversion does occur, it does so automatically, and the conversion, if successful, is invisible to the application.

The following list defines some of the terms used for character conversion.

ASCII Acronym for American Standard Code for Information Interchange, an encoding scheme used to represent characters. It is limited to 256 code positions. The term ASCII is used throughout this book to refer to IBM-PC Data or ISO 8-bit data.

character set

A defined set of characters, a character being the smallest component of written language that has semantic value. For example, the following character set appears in several code pages:

- 26 nonaccented letters A through Z
- 26 nonaccented letters a through z
- digits 0 through 9
- . , : ; ? () ' " / - _ & + % * = < >

code page

A set of assignments of characters to code points. In EBCDIC, for example, "A" is assigned code point X'C1' and "B" is assigned code point X'C2'. In Unicode, "A" is assigned code point "U+0041". Within a code page, each code point has only one specific meaning.

code point

A unique bit pattern that represents a character. It is a numerical index or position in an encoding table used for encoding characters.

coded character set

A set of unambiguous rules that establishes a character set and the one-to-one relationships between the characters of the set and their coded representations. It is a character set in which each character is assigned a numeric code value.

coded character set identifier (CCSID)

A two-byte, unsigned binary integer that uniquely identifies an encoding scheme and one or more pairs of character sets and code pages.

EBCDIC

Acronym for Extended Binary-Coded Decimal Interchange Code, an encoding scheme used to represent character data, a group of coded character sets that consist of 8-bit coded characters. EBCDIC coded character sets use the first 64 code positions (X'00' to X'3F') for control codes and the range X'41' to X'FE' for graphic characters.

encoding scheme

A set of rules used to represent character data. All string data stored in a table must use the same encoding scheme and all tables within a table

DB2 concepts

space must use the same encoding scheme, except for global temporary tables, declared temporary tables, and workfile tablespaces. Encoding schemes include:

- ASCII
- EBCDIC
- Unicode

substitution byte

A unique character that is substituted during character conversion for any characters in the source encoding representation that do not have a match in the target encoding representation.

Unicode

A universal encoding scheme for written characters and text that enables the exchange of data internationally. It provides a character set standard that can be used all over the world. It uses a 16-bit encoding form that provides code points for more than 65,000 characters and an extension called UTF-16 that allows for encoding as many as a million more characters. It provides the ability to encode all characters used for the written languages of the world and treats alphabetic characters, ideographic characters, and symbols equivalently because it specifies a numeric value and a name for each of its characters. It includes punctuation marks, mathematical symbols, technical symbols, geometric shapes, and dingbats. Three encoding forms include the following:

- UTF-8: Unicode Transformation Format, a 8-bit encoding form designed for ease of use with existing ASCII-based systems. The CCSID value for data in UTF-8 format is 1208.
- UCS-2: Universal Character Set coded in 2 octets, which means that characters are represented in 16-bits per character.
- UTF-16: Unicode Transformation Format, a 16-bit encoding form designed to provide code values for over a million characters and a superset of UCS-2. The CCSID value for data in UTF-16 format is 1200.²

Character conversion can affect the results of several SQL operations. In this book, the effects are described in:

“Conversion rules for string assignment” on page 70

“Conversion rules for string comparison” on page 73

“Character conversion in unions and concatenations” on page 366

Character sets and code pages

The following example shows how a typical character set might map to different code points in two different code pages.

2. DB2 UDB for OS/390 implements UTF-8 and UTF-16. UTF-8 data is supported in mixed data fields and UTF-16 is supported in graphic data fields.

code page: pp1 (ASCII)

	0	1	2	3	4	5		E	F
0				0	@	P		Å	
1				1	A	Q		À	α
2			†	2	B	R		Å	β
3				3	C	S		Á	γ
4				4	D	T		Ã	σ
5			%	5	E	U		Ä	ε
E		.	>	N			¼	ö	
F		/	*	Ø			®		

code point: 2F

character set ss1
(in code page pp1)

code page: pp2 (EBCDIC)

	0	1		A	B	C	D	E	F
0					#				0
1					\$	A	J		1
2				s	%	B	K	S	2
3				t	—	C	L	T	3
4				u	*	D	M	U	4
5				v	(E	N	V	5
E					!	:	À		
F				À	¢	;	Á	{	

character set ss1
(in code page pp2)

Even with the same encoding scheme, different coded character sets exist, and the same code point can represent a different character in different coded character sets. Furthermore, a byte in a character string does not necessarily represent a character from a single-byte character set (SBCS). Character strings are also used for mixed data (that is a mixture of single-byte characters and multibyte characters) and for data that is not associated with any character set (called bit data). Note that this is not the case with graphic strings; every pair of bytes in every graphic string is assumed to represent a character from a double-byte character set (DBCS).

Character encoding for IBM systems is described in *Character Data Representation Architecture Reference and Registry*. For more information on Unicode, see the Unicode standards Web site at <http://www.unicode.org>.

System CCSIDs

Every string used in an SQL operation has a CCSID. The CCSID identifies the manner in which the characters in the string are encoded. Strings can be encoded in ASCII, EBCDIC, or Unicode. A string representing characters can be one of three types:

- A character string composed of SBCS (*single-byte character set*) characters. In an SBCS string, each character is represented by a single byte. SBCS is a subtype of the character data type.
- A graphic string composed of DBCS (*double-byte character set*) characters. In a graphic string, each character is represented by a pair of bytes, except in UTF-16 where surrogates take four bytes per character. For more information on surrogates, see “Other considerations for using UTF-8 and UTF-16” on page 25.

DB2 concepts

- An MBCS *multibyte character set*, in which multibyte characters, including single-byte characters, can occur. In an EBCDIC mixed string, certain *shift characters* serve as left- and right-delimiters for sequences of multibyte characters that do not include single-byte characters. MIXED is a subtype of the character data type.

Every query has some context for an encoding scheme (ASCII, EBCDIC, or Unicode) whether data is explicitly selected from the database or not. If other string data from a table or view is selected by a query, the encoding scheme of that string determines the resulting encoding scheme. The resulting CCSID for string data is the appropriate CCSID for the encoding scheme of the statement. If there is no other string data from a table or a view in the query, the default encoding scheme is used.

When DB2 is installed, the values specified in fields ASCII CODED CHAR SET, EBCDIC CODED CHAR SET, and UNICODE CCSID on installation panel DSNTIPF determine the system CCSIDs for ASCII, EBCDIC, or Unicode data. The ASCII CODED CHAR SET and EBCDIC CODED CHAR SET fields should contain valid SBCS CCSIDs if the field MIXED DATA on the same installation panel is NO, or should contain valid MIXED CCSIDs if the field MIXED DATA is YES. For example, one CCSID whose value is 37 identifies a widely used form of EBCDIC encoding. That particular CCSID could be the system CCSID for EBCDIC SBCS strings. The UNICODE CCSID field on the same installation panel should contain 1208, which is the CCSID of UTF-8 data. The field DEF ENCODING SCHEME on the installation panel DSNTIPF determines whether the default encoding scheme is ASCII, EBCDIC, or Unicode.

At a given DB2, all columns that contain SBCS strings for an encoding scheme are assumed to have a common CCSID for SBCS data. For example, when character data is fetched from a table at another DBMS, DB2 converts the data from the CCSID at the source system to the corresponding CCSID at the target system. If the character string has a subtype of BIT, its bytes do not represent characters and are not converted.

If the CCSID of an input host variable or a host variable substituted for a parameter marker is different from the CCSID determined at bind time, and if either CCSID is X'FFFF' (BIT data), an error occurs. Otherwise, the host variable is converted to the coded character set determined by the CCSID at bind time.

For more information about character string subtypes and SBCS and DBCS DB2 sites, see “Data types” on page 48. For information on the subsystem parameters that determine the default encoding scheme and the system CCSIDs, see *DB2 Installation Guide*.

Expanding conversions

An *expanding conversion* occurs when the length of the converted string is greater than that of the source string. For example, an expanding conversion occurs when an ASCII mixed data string that contains DBCS characters is converted to EBCDIC mixed data. Because of the addition of shift codes, an error occurs when an expanding conversion is performed on a fixed-length input host variable that requires conversion from ASCII mixed to EBCDIC mixed. The remedy is to use a varying-length string variable with a maximum length that is sufficient to contain the expansion.

Expanding conversions also can occur when string data is converted to or from Unicode. It can also occur between UTF-8 and UTF-16, depending on the data being converted. UTF-8 uses one, two, three, or four bytes per character. UTF-16 uses two or four bytes per character. If UTF-8 were being converted to UTF-16, a one byte character would be expanded to two bytes.

Contracting conversions

A *contracting conversion* occurs when the length of the converted string is smaller than that of the source string. For example, a contracting conversion occurs when an EBCDIC mixed data string that contains DBCS characters is converted to ASCII mixed data due to the removal of shift codes.

Contracting conversions also can occur when string data is converted to or from Unicode data. It can also occur between UTF-8 and UTF-16, depending on the data being converted.

Other considerations for using UTF-8 and UTF-16

Storage for UTF-8 is one to four bytes per character. Storage for UTF-16 is two or four bytes per character. For UTF-16, surrogates use four bytes per character. Surrogates are designed to allow representation of characters in future extensions of the Unicode standard.

Chapter 2. Language elements

Characters	31
Tokens	31
Identifiers	32
SQL identifiers	32
Ordinary identifiers	32
Delimited identifiers	33
Short and long identifiers	33
Location identifiers	34
Host identifiers	34
Naming conventions	34
Qualification of unqualified object names	39
Unqualified alias, index, table, and view names	39
Unqualified data type, function, and procedure names	40
Schemas and the SQL path	40
Aliases and synonyms	41
Authorization IDs and authorization-names.	42
Authorization IDs and schema names	43
Authorization IDs and statement preparation	43
Authorization IDs and dynamic SQL	43
Authorization IDs and remote execution.	46
DRDA access with DB2 for OS/390 and z/OS only.	46
DRDA access with a server or requester other than DB2	47
DB2 private protocol access	47
Authorization ID translations	47
Other security measures	47
Data types	48
Character strings	49
The effect of encoding schemes on DBCS characters in mixed strings	49
Examples	50
DB2 and SBCS defaults	50
DB2 and DBCS defaults	50
Fixed-length character strings	51
Varying-length character strings.	51
Character string host variables	52
Graphic strings	52
Fixed-length graphic strings	52
Varying-length graphic strings	52
Graphic string host variables	53
Binary strings	53
Large objects (LOBs)	53
Restrictions using long strings	54
Numbers	55
Small integer (SMALLINT).	55
Large integer (INTEGER)	55
Single precision floating-point (REAL)	55
Double precision floating-point (DOUBLE or FLOAT)	55
Decimal (DECIMAL or NUMERIC)	55
String representations of numbers	56
Numeric host variables	56
Datetime values	56
Date.	56
Time.	57
Timestamp	57

Language elements

String representations of datetime values	57
Restrictions on the use of local datetime formats	59
Row ID values	60
Distinct types	60
Promotion of data types	61
Casting between data types	62
Assignment and comparison	64
Numeric assignments	66
Decimal or integer to floating-point.	66
Floating-point or decimal to integer	66
Decimal to decimal	67
Integer to decimal.	67
Floating-point to floating-point	67
Floating-point to decimal	67
To COBOL integers	68
String assignments	68
Storage assignment	69
Retrieval Assignment.	69
Assignments involving mixed data strings	69
Assignments involving C NUL-terminated strings	69
Conversion rules for string assignment	70
Datetime assignments	70
Row ID assignments.	71
Distinct type assignments	71
Numeric comparisons	72
String comparisons	73
String comparisons with field procedures	73
Conversion rules for string comparison	73
Datetime comparisons	75
Row ID comparisons.	75
Distinct type comparisons	75
Rules for result data types.	77
String operands	77
Binary string operands	78
Numeric operands.	78
Datetime operands	79
Row ID operands	79
Distinct type operands	79
Nullable attribute of a result	79
Constants	79
Integer constants	79
Floating-point constants	80
Decimal constants.	80
Character string constants.	80
Datetime constants	81
Graphic string constants	81
Special registers	82
General rules for special registers	83
CURRENT APPLICATION ENCODING SCHEME	85
CURRENT DATE	86
CURRENT DEGREE.	86
CURRENT LOCALE LC_CTYPE	87
CURRENT MEMBER	87
CURRENT OPTIMIZATION HINT	87
CURRENT PACKAGESET	88
CURRENT PATH	88

CURRENT PRECISION	89
CURRENT RULES	90
CURRENT SERVER.	91
CURRENT SQLID.	91
CURRENT TIME	91
CURRENT TIMESTAMP	92
CURRENT TIMEZONE	92
USER	92
Inheriting special registers in a user-defined function or a stored procedure	93
Column names	95
Qualified column names	95
Correlation names.	95
Column name qualifiers to avoid ambiguity	96
Column name qualifiers in correlated references	97
Resolution of column name qualifiers and column names	98
References to variables.	99
References to host variables	100
Host variables in dynamic SQL	101
References to LOB host variables	101
References to LOB locator variables	102
References to stored procedure result sets	102
References to result set locator variables.	102
Host structures in PL/I, C, and COBOL	103
Functions	104
Types of functions	104
Built-in functions	104
User-defined functions.	105
Cast functions.	105
Additional way to classify functions	106
Function resolution	107
Method of finding the best fit	108
SQL path considerations for built-in functions	110
Function invocation	110
Expressions.	111
Without operators	111
With the concatenation operator	112
With arithmetic operators.	114
Arithmetic with two integer operands	114
Arithmetic with an integer and a decimal operand.	114
Arithmetic with two decimal operands	114
Decimal addition and subtraction	115
Decimal multiplication	115
Decimal division	116
Arithmetic with floating-point operands	117
Datetime operands and durations.	117
Datetime arithmetic in SQL	118
Date arithmetic	119
Time arithmetic	121
Timestamp arithmetic	122
Precedence of operations	123
CASE expressions	124
CAST specification	126
Predicates	130
Basic predicate	130
Quantified predicate	132
BETWEEN predicate	134

Language elements

EXISTS predicate	134
IN predicate	136
LIKE predicate	137
Examples	142
NULL predicate	144
Search conditions	145
Options affecting SQL	146
Precompiler options for dynamic statements.	148
Decimal point representation	148
Apostrophes and quotation marks in string delimiters	149
Katakana characters for EBCDIC.	150
Mixed data in character strings	150
Formatting of datetime strings	151
SQL standard language	151
Positioned updates of columns	153

This chapter defines the basic syntax of SQL and language elements that are common to many SQL statements.

Characters

The basic symbols of SQL are characters from the EBCDIC syntactic character set. These *characters* are classified as letters, digits, or special characters:

- A *letter* is any one of the uppercase alphabetic characters A through Z plus the three EBCDIC code points reserved as alphabetic extenders for national languages (the code points X'5B', X'7B', and X'7C', which display as \$, #, and @ using code pages 37 and 500).
- A *digit* is any one of the characters 0 through 9.
- A *special character* is any character other than a letter or a digit.

SQL statements can also contain *double-byte character set (DBCS)* characters. Regardless of the value of the field MIXED DATA on installation panel DSNTIPF, double-byte characters can be used in SQL ordinary identifiers and graphic string constants when enclosed by the necessary shift characters. If the value of MIXED DATA is YES, double-byte characters can also be used in string constants and delimited identifiers. In SQL application programs, any use of double-byte characters must be contained within a single line. Thus, a graphic string constant cannot be continued from one line to the next and, if MIXED DATA is YES, a character string constant and delimited identifier can be continued from one line to the next only if the break occurs between single-byte characters. This restriction also applies to the use of double-byte characters within tokens of the host language.

Tokens

The basic syntactical units of the language are called *tokens*. A token consists of one or more characters of which none are blanks, control characters, or characters within a string constant or delimited identifier.

Tokens are classified as *ordinary* or *delimiter* tokens:

- An *ordinary token* is a numeric constant, an ordinary identifier, a host identifier, or a keyword.

Examples:

1	.1	+2	SELECT	E	3
---	----	----	--------	---	---

- A *delimiter token* is a string constant, a delimited identifier, an operator symbol, or any of the special characters shown in the syntax diagrams. A question mark (?) is also a delimiter token when it serves as a parameter marker, as explained in “PREPARE” on page 846.

Examples:

,	'string'	"fld1"	=	.
---	----------	--------	---	---

String constants and certain delimited identifiers are the only tokens that can include a space or control character. Any token can be followed by a space or control character. Every ordinary token must be followed by a delimiter token, a space, or a control character; if the syntax does not allow a delimiter token, a space or a control character must follow the ordinary token.

Spaces: A space is a sequence of one or more blank characters.

Tokens

Control characters: A *control character* is a special character that is used for string alignment. Treated similar to a space, a control character does not cause a particular action to occur. DB2 handles the following control characters:

Control character	EBCDIC hex value
Tab	05
Form feed	0C
Carriage return	0D
New line or next line	15
Line feed (new line)	25

Uppercase and lowercase: Any token can include lowercase letters, but a lowercase letter in an ordinary token is folded to uppercase. However, it is only folded to uppercase in a C program if the appropriate precompiler is specified. Delimiter tokens are never folded to uppercase.

Example: The statement:

```
select * from DSN8710.EMP where lastname = 'Smith';
```

is equivalent, after folding, to:

```
SELECT * FROM DSN8710.EMP WHERE LASTNAME = 'Smith';
```

Identifiers

An *identifier* is a token used to form a name. An identifier in an SQL statement is an SQL identifier, a location identifier, or a host identifier. See Appendix A, “Limits in DB2 for OS/390 and z/OS,” on page 965 for the identifier length limits that DB2 imposes.

SQL identifiers

SQL identifiers can be *ordinary identifiers* or *delimited identifiers*. They can also be *short identifiers* or *long identifiers*. Thus, an SQL identifier can be in one of four categories: short ordinary, long ordinary, short delimited, or long delimited.

Ordinary identifiers

An *ordinary identifier* is a letter followed by zero or more characters, each of which is a letter, a digit, or the underscore character. An ordinary identifier with an EBCDIC encoding scheme can include Katakana characters if the value of field EBCDIC CODED CHAR SET on installation panel DSNTIPF is set to 930 or 5026 when the statement is parsed.

DBCS characters are allowed in SQL ordinary identifiers. An SQL ordinary identifier, when used as the name of a table, column, alias, synonym, view, statement, cursor, correlation, distinct type, stored procedure, user-defined function, or trigger name can be specified using either DBCS characters or *single-byte character set* (SBCS) characters. However, an SQL ordinary identifier cannot contain a mixture of SBCS and DBCS characters.

The following list shows the rules for forming DBCS SQL ordinary identifiers. These are EBCDIC rules because DB2 processes SQL statements in EBCDIC.

- The identifier must start with a shift-out (X'0E'), end with a shift-in (X'0F'), and an odd-numbered byte between those shifts must not be a shift-out.
- The maximum length is 18 bytes including the shift-out and the shift-in. In other words, there is a maximum of 16 bytes (8 double-byte characters) between the shift-out and the shift-in.

- There must be an even number of bytes between the shift-out and the shift-in.
- DBCS blanks (X'4040') are not acceptable between the shift-out and the shift-in.
- The identifiers are not folded to uppercase or changed in any other way.
- Continuation to the next line is not allowed.

An ordinary identifier must not be identical to a keyword that is a reserved word in any context in which the identifier is used. For a list of reserved words, see Appendix F, “SQL reserved words,” on page 1153.

Example: The following example is an ordinary identifier:

```
SALARY
```

Delimited identifiers

A *delimited identifier* is a sequence of one or more characters enclosed within escape characters. The escape character is the quotation mark ("") except for:

- Dynamic SQL when the field SQL STRING DELIMITER on installation panel DSNTIPF is set to the quotation mark ("") and either of these conditions is true:
 - DYNAMICRULES run behavior applies. For a list of the DYNAMICRULES bind option values that specify run, bind, define, or invoke behavior, see Table 2 on page 44.
 - DYNAMICRULES bind, invoke, or define behavior applies and installation panel field USE FOR DYNAMIC RULES is YES.

In this case, the escape character is the apostrophe (').

However, for COBOL application programs, if DYNAMICRULES run behavior does not apply and installation panel field USE FOR DYNAMICRULES is NO, a COBOL compiler option specifies whether the escape character is the quotation mark or apostrophe.

- Static SQL in COBOL application programs. A COBOL compiler option specifies whether the escape character is the quotation mark ("") or the apostrophe (').

A delimited identifier can be used when the sequence of characters does not qualify as an ordinary identifier. Such a sequence, for example, could be an SQL reserved word, or it could begin with a digit. Two consecutive escape characters are used to represent one escape character within the delimited identifier. A delimited identifier that contains double-byte characters also must contain the necessary shift characters.

Example: If the escape character is the quotation mark, the following example is a delimited identifier:

```
"VIEW"
```

Short and long identifiers

SQL identifiers are also classified according to their maximum length. A *long identifier* has a maximum length of 18 bytes. A *short identifier* has a maximum length of 8 bytes. These limits do not include the escape characters of a delimited identifier.

Whether an identifier is long or short depends on what it represents. For example, the name of a storage group is a short identifier, whereas an unqualified table name is a long identifier. “Naming conventions” on page 34 describes what identifiers can represent and whether those representing a given type of entity are long or short.

Identifiers

Database names and table space names are examples of short identifiers that will be used as part of data set names. Such identifiers, whether ordinary or delimited, must conform to the MVS rules for forming data set names. For example, a short ordinary identifier used to name a database must not contain an underscore character.

Location identifiers

A location identifier is like an SQL identifier, except as follows:

- The maximum length is 16 bytes.
- The ordinary form must not include alphabetic extenders, lowercase letters, or Katakana characters.
- The characters allowed in the delimited form are the same as those allowed in the ordinary form.

Host identifiers

A *host identifier* is a name declared in the host program. The rules for forming a host identifier are the rules of the host language.

Naming conventions

The rules for forming a name depend on the type of the object designated by the name. The syntax diagrams use different terms for different types of names. The following list defines these terms.

alias-name

A qualified or unqualified name that designates an alias, table, or view. An alias name designates an alias when it is preceded by the keyword ALIAS, as in CREATE ALIAS, DROP ALIAS, COMMENT ON ALIAS, and LABEL ON ALIAS. In all other contexts, an alias name designates a table or view. For example, COMMENT ON ALIAS A specifies a comment about the alias A, whereas COMMENT ON TABLE A specifies a comment about the table or view designated by A.

An alias can be defined at a local server and can refer to a table or view that is at the current server or a remote server. The alias name can be used wherever the table name or view name can be used to refer to the table or view in an SQL statement. The rules for forming an alias name are the same as the rules for forming a table name or a view name, as explained below. A fully qualified alias name (a three-part name) can refer to an alias at a remote server. However, the table or view identified by the alias at the remote server must exist at the remote server.

Statements that use three-part names and refer to distributed data result in either DB2 private protocol access or DRDA access to the remote site.

DRDA access for three-part names is used when the plan or package that contains the query to distributed data is bound with bind option

DBPROTOCOL(DRDA), or the value of field DATABASE PROTOCOL on installation panel DSNTIP5 is DRDA and bind option PROTOCOL was not specified when the plan or package was bound. When an application program uses three-part name aliases for remote objects and DRDA access, the application program must be bound at each location that is specified in the three-part names. Also, each alias needs to be defined at the local site. An alias at a remote site can refer to yet another server as long as a referenced alias eventually refers to a table or view.

authorization-name

A short identifier that designates a set of privileges. It can also designate a user or group of users, but DB2 does not control this property. See “Authorization IDs and authorization-names” on page 42 for the distinction between an authorization name and an authorization ID.

aux-table-name

A qualified or unqualified name that designates an auxiliary table. The rules for the name are the same as the rules for *table-name*. See “table-name” on page 38.

bpname

A name that identifies a buffer pool. The following list shows the names of the different buffer pool sizes.

4KB	BP0, BP1, BP2, ..., BP49
8KB	BP8K0, BP8K1, BP8K2, ..., BP8K9
16KB	BP16K0, BP16K1, BP16K2, ..., BP16K9
32KB	BP32K, BP32K1, BP32K2, ..., BP32K9

built-in-data-type

A qualified or unqualified name that identifies an IBM-supplied data type. A qualified name is SYSIBM followed by a period and the name of the built-in data type. An unqualified name has an implicit qualifier, the schema name, which is determined by the rules in “Qualification of unqualified object names” on page 39.

catalog-name

A short identifier that designates an integrated catalog facility catalog. The identifier must start with a letter and must not include special characters.

collection-id

A long identifier that identifies a collection of packages; therefore, a collection ID is a qualifier for a package ID. Refer to Chapter 1 of *DB2 Command Reference* for naming conventions.

column-name

A qualified or unqualified name that designates a column of a table or view.
A qualified column name is a qualifier followed by a period and a long identifier. The qualifier is a table name, a view name, a synonym, an alias, or a correlation name.
An unqualified column name is a long identifier.

A SQL identifier of 1 to 64 bytes used in an SQL procedure body to identify a condition.

constraint-name

A short identifier that designates a referential constraint or a long identifier that designates a unique constraint or a check constraint on a table.

|

correlation-name

A long identifier that designates a table, a view, or individual rows of a table or view.

cursor-name

A long identifier that designates an SQL cursor.

database-name

A short identifier that designates a database. The identifier must start with a letter and must not include special characters.

Naming conventions

descriptor-name

A host identifier that designates an SQL descriptor area (SQLDA). See “References to host variables” on page 100 for a description of a host identifier. A descriptor name never includes an indicator variable.

distinct-type-name

A qualified or unqualified name that designates a distinct type.

A qualified distinct type name is a two-part name. The first part is a short identifier. The short identifier is the schema name of the distinct type. The second part is a long identifier. A period must separate each of the parts.

An unqualified distinct type name is a long identifier with an implicit qualifier. The implicit qualifier is the schema name, which is determined by the context in which the distinct type appears as described by the rules in “Qualification of unqualified object names” on page 39.

function-name

A qualified or unqualified name that designates a user-defined function, a cast function that was generated when a distinct type was created, or a built-in function.

A qualified function name is a two-part name. The first part is a short identifier. The short identifier is the schema name of the function. The second part is a long identifier. A period must separate each of the parts.

An unqualified function name is a long identifier with an implicit qualifier. The implicit qualifier is the schema name, which is determined by the context in which the function appears as described by the rules in “Qualification of unqualified object names” on page 39.

host-variable

A sequence of tokens that designates a host variable. A host variable includes at least one host identifier, as explained in “References to host variables” on page 100.

index-name

A qualified or unqualified name that designates an index.

A qualified index name is a short identifier followed by a period and a long identifier. The short identifier is the authorization ID that owns the index.

An unqualified index name is a long identifier with an implicit qualifier. The implicit qualifier is an authorization ID that is determined by the rules set forth in “Qualification of unqualified object names” on page 39.

For an index on a declared temporary table, the qualifier must be SESSION.

label An SQL identifier of 1 to 64 bytes used in an SQL procedure body to label a statement.

location-name

A location identifier that identifies an instance of a database management system.

package-id

A short identifier that identifies a package. For packages created using DB2, a package ID is the name of the program whose precompilation produced the package’s DBRM. Refer to Chapter 1 of *DB2 Command Reference* for naming conventions.

plan-name

A short identifier that identifies an application plan. Refer to Chapter 1 of *DB2 Command Reference* for naming conventions.

procedure-name

A qualified or unqualified name that designates a stored procedure.

A fully qualified procedure name is a three-part name. The first part is a location name that identifies the DBMS at which the procedure is stored. The second part is the schema name of the stored procedure. The third part is a long identifier. A period must separate each of the parts.

A two-part procedure name is implicitly qualified with the location name of the current server. The first part is the schema name of stored procedure. The second part is a long identifier. A period must separate the two parts.

A one-part or unqualified procedure name is a long identifier with two implicit qualifiers. The first implicit qualifier is the location name of the current server. The second implicit qualifier depends on the server. If the server is DB2 for OS/390 and z/OS, the implicit qualifier is schema name, which is determined by the context in which the unqualified name appears as described by the rules in “Qualification of unqualified object names” on page 39.

The content of an SQL procedure name (whether as a simple identifier or a delimited identifier) must contain only the uppercase alphabetic characters A through Z, the characters 0 through 9, or the underscore. The name must begin with an alphabetic character.

program-name

A short identifier that designates an exit routine.

schema-name

A short identifier that designates a schema. A *schema-name* that is used as a qualifier of the name of an object is often also an authorization ID. The objects that are qualified with a schema name are distinct types, stored procedures, triggers, and user-defined functions. Built-in data types and built-in functions are also qualified with a schema name.

specific-name

A qualified or unqualified name that designates a unique name for a user-defined function.

A qualified specific name is a two-part name. The first part is a short identifier. The short identifier is the schema name. The second part is a long identifier. A period must separate each of the parts.

An unqualified specific name is a long identifier with an implicit qualifier. The implicit qualifier is the schema name, which is determined by the context in which the unqualified name appears as described by the rules in “Qualification of unqualified object names” on page 39.

A specific name can be used to identify a function to alter, comment on, drop, grant privileges on, revoke privileges from, or be the source function for another function. A specific name cannot be used to invoke a function. In addition to being used in certain SQL statements, a specific name must be used in DB2 commands to uniquely identify a function.

SQL-variable-name

A qualified or unqualified name that designates a variable in an SQL procedure body. The unqualified form of an SQL variable name is an SQL identifier of 1 to 64 bytes. If the SQL variable is a delimited identifier, the

Naming conventions

contents of the delimited identifier must conform to the rules for ordinary identifiers. The qualified form is an SQL procedure statement label followed by a period (.) and an SQL identifier.

statement-name

A long identifier that designates a prepared SQL statement.

stogroup-name

A short identifier that designates a storage group. The identifier must start with a letter and must not include special characters.

svpt-name

A savepoint identifier that designates a savepoint. A savepoint identifier is like an SQL identifier except it has a maximum length of 128 bytes.

synonym

A long identifier that designates a synonym, a table, or a view. The table or view must exist at the current server. A synonym designates a synonym when it is preceded by the keyword SYNONYM, as in CREATE SYNONYM and DROP SYNONYM. In all other contexts, a synonym designates a local table or view and can be used wherever the name of a table or view can be used in an SQL statement. A qualified name is never interpreted as a synonym.

table-name

A qualified or unqualified name that designates a table.

A fully qualified table name is a three-part name. The first part is a location name that designates the DBMS at which the table is stored. The second part is the authorization ID that designates the owner of the table. The third part is a long identifier. A period must separate each of the parts.

A two-part table name is implicitly qualified by the location name of the current server. The first part is the authorization ID that designates the owner of the table. The second part is a long identifier. A period must separate the two parts.

A one-part or unqualified table name is a long identifier with two implicit qualifiers. The first implicit qualifier is the location name of the current server. The second is an authorization ID, which is determined by the rules set forth in “Qualification of unqualified object names” on page 39.

For a declared temporary table, the qualifier that designates the owner (the second part in a three-part name and the first part in a two-part name) must be SESSION. For complete details on specifying a name when a declared temporary table is defined and then later referring to that declared temporary table in other SQL statements, see “DECLARE GLOBAL TEMPORARY TABLE” on page 725.

table-space-name

A short identifier that designates a table space of an identified database. The identifier must start with a letter and must not include special characters. If a database is not identified, DSNDB04 is implicit.

trigger-name

A qualified or unqualified name that designates a trigger.

A qualified trigger name is a two-part name. The first part is a short identifier. The short identifier is the schema name of the trigger. The second part is also a short identifier. A period must separate each of the parts.

An unqualified trigger name is a short identifier with an implicit qualifier. The implicit qualifier is the schema name, which is determined by the rules set forth in “Qualification of unqualified object names.”

version-id

An identifier³ of 1 to 64 characters that is assigned to a package when the package is created. The version ID that is assigned is taken from the version ID associated with the program being bound. Version IDs are specified for programs as a parameter of the DB2 precompiler. Refer to Chapter 1 of *DB2 Command Reference* for naming conventions.

view-name

A qualified or unqualified name that designates a view.

A fully qualified view name is a three-part name. The first part is a location name that designates the DBMS where the view is defined. The second part is the authorization ID that designates the owner of the view. The third part is a long identifier. A period must separate each of the parts.

A two-part view name is implicitly qualified by the location name of the current server. The first part is the authorization ID that designates the owner of the view. The second part is a long identifier. A period must separate the two parts.

A one-part or unqualified view name is a long identifier with two implicit qualifiers. The first implicit qualifier is the location name of the current server. The second is an authorization ID, which is determined by the rules set forth in “Qualification of unqualified object names.”

Qualification of unqualified object names

Unqualified object names are implicitly qualified. The rules for qualifying a name differ depending on the type of object that the name identifies.

Unqualified alias, index, table, and view names

Unqualified alias, index, table, and view names are implicitly qualified as follows:

- For static SQL statements, the implicit qualifier is the identifier specified in the QUALIFIER option of the BIND subcommand used to bind the SQL statements. If this bind option was not used when the plan or package was created or last rebound, the implicit qualifier is the authorization ID of the owner of the plan or package.
- For dynamic SQL statements, the behavior as specified by the combination of bind option DYNAMICRULES and the run-time environment determines the implicit qualifier. (For a list of these behaviors and the DYNAMICRULES values that determine them, see Table 2 on page 44).
 - If *DYNAMICRULES run behavior* applies, the implicit qualifier is the SQL authorization ID in the CURRENT SQLID special register. Run behavior is the default.
 - If *bind behavior* applies, the identifier implicitly or explicitly specified in the QUALIFIER option of the BIND subcommand, as explained above for static SQL statements.
 - If *define behavior* applies, the implicit qualifier is the owner of the function or stored procedure (the owner is the definer).
 - If *invoke behavior* applies, the implicit qualifier is the authorization ID of the invoker of the function or stored procedure.

3. The *version-id* can begin with a digit, for example, when it is a timestamp.

Naming conventions

Exception: For bind, define, and invoke behavior, the implicit qualifier of PLAN_TABLE (output from the EXPLAIN statement) is always the value in special register CURRENT SQLID.

Unqualified data type, function, and procedure names

The qualification of data type, function, and stored procedure names depends on the SQL statement in which the unqualified name appears:

- If an unqualified name is the main object of an ALTER, CREATE, COMMENT ON, DROP, GRANT, or REVOKE statement, the name is implicitly qualified with a schema name as follows:
 - In a static statement, the implicit schema name is the identifier specified in the QUALIFIER option of the BIND subcommand used to bind the SQL statements. If this bind option was not used when the plan or package was created or last rebound, the implicit qualifier is the authorization ID of the owner of the plan or package.
 - In a dynamic statement, the implicit schema name is the SQL authorization ID in the CURRENT SQLID special register.
- Otherwise, the implicit schema name for the unqualified name is determined as follows:
 - For data type names, DB2 searches the SQL path and selects the first schema in the path such that the data type exists in the schema and the user has authorization to use the data type.
 - For function names, DB2 uses the SQL path in conjunction with function resolution, as described under “Function resolution” on page 107.
 - For stored procedure names⁴, DB2 searches the SQL path and selects the first schema in the path such that the schema contains a procedure with the same name and number of parameters and the user has authorization to use the procedure.

For information on the SQL path, see “Schemas and the SQL path.”

Schemas and the SQL path

The SQL path is an ordered list of schema names. DB2 uses the path to resolve the schema name for unqualified data type, function, and stored procedure names that appear in any context other than as the main object of an ALTER, CREATE, DROP, COMMENT ON, GRANT or REVOKE statement.⁵ Searching through the path from left to right, DB2 implicitly qualifies the object name with the first schema name in the path that contains the same object with the same unqualified name for which the user has appropriate authorization. For procedures, DB2 selects a matching procedure name only if the number of parameters is also the same. For functions, DB2 uses a process called function resolution in conjunction with the SQL path to determine which function to choose because several functions with the same name can reside in a schema. (For details, see “Function resolution” on page 107.)

For example, if the SQL path is SMITH, XGRAPHIC, SYSIBM and an unqualified distinct type name MYTYPE was specified, DB2 looks for MYTYPE first in schema SMITH, then XGRAPHIC, and then SYSIBM.

The PATH bind option establishes the SQL path used to resolve:

4. In CALL statements only.

5. The SQL path does not apply to unqualified procedure names in ASSOCIATE LOCATOR and DESCRIBE PROCEDURE statements. For these statements, an implicit schema name is not generated.

- Unqualified data type and function names in static SQL statements
- Unqualified procedure names in SQL CALL statements that specify the procedure name as a literal (CALL 'literal')

If the PATH bind option was not specified when the plan or package was created or last rebound, its default value is: SYSIBM, SYSFUN, SYSPROC, *plan or package qualifier*.

The CURRENT PATH special register determines the SQL path used to resolve:

- Unqualified data type and function names in dynamic SQL statements
- Unqualified procedure names in SQL CALL statements that specify the procedure name in a host variable (CALL *host-variable*)

Generally, the initial value of the CURRENT PATH special register is:

- The value of the PATH bind option, or
- SYSIBM, SYSFUN, SYSPROC, *value of CURRENT SQLID special register* if the PATH bind option was not specified.

For additional details on the initial value of CURRENT PATH special register and changing its value, see “CURRENT PATH” on page 88 and “SET PATH” on page 921.

If schema SYSIBM or SYSPROC is not explicitly specified in the SQL path, the schema is implicitly assumed at the front of the path; if both are not specified, they are assumed in the order of SYSIBM, SYSPROC. For example, assume that the SQL path is explicitly specified as SYSIBM, GEORGIA, SMITH. As an implicitly assumed schema, SYSPROC is added to the beginning of the explicit path effectively making the path:

SYSPROC, SYSIBM, GEORGIA, SMITH

Aliases and synonyms

A table or view can be referred to in an SQL statement by its name, by an alias that has been defined for its name, or by a synonym that has been defined for its name. Thus, aliases and synonyms can be thought of as alternate names for tables and views.

The option of referencing a table or view by an alias or a synonym is not explicitly shown in the syntax diagrams or mentioned in the description of SQL statements. Nevertheless, an alias or a synonym can be used wherever a table or view can be referred to in an SQL statement, with two exceptions: a local alias cannot be used in CREATE ALIAS, and a synonym cannot be used in CREATE SYNONYM. If an alias is used in CREATE SYNONYM, it must identify a table or view at the current server. The synonym is defined on the name of that table or view. If a synonym is used in CREATE ALIAS, the alias is defined on the name of the table or view identified by the synonym.

The effect of using an alias or a synonym in an SQL statement is that of text substitution. For example, if A is an alias for table Q.T, one of the steps involved in the preparation of SELECT * FROM A is the replacement of 'A' by 'Q.T'. Likewise, if S is a synonym for Q.T, one of the steps involved in the preparation of SELECT * FROM S is the replacement of 'S' by 'Q.T'.

The differences between aliases and synonyms are as follows:

- SYSADM or SYSCTRL authority or the CREATE ALIAS privilege is required to define an alias. No authorization is required to define a synonym.

Aliases and synonyms

- An alias can be defined on the name of a table or view, including tables and views that are not at the current server. A synonym can only be defined on the name of a table or view at the current server.
- An alias can be defined on an undefined name. A synonym can only be defined on the name of an existing table or view.
- Dropping a table or view has no effect on its aliases. But dropping a table or view does drop its synonyms.
- An alias is a qualified name that can be used by any authorization ID. A synonym is an unqualified name that can only be used by the authorization ID that created it.
- An alias defined at one DB2 subsystem can be used at another DB2 subsystem. A synonym can only be used at the DB2 subsystem where it is defined.
- When an alias is used, an error occurs if the name that it designates is undefined or is the name of an alias at the current server. (The alias can represent another alias at a different server, which can represent yet another alias at yet another server as long as eventually a referenced alias represents a table or view.) When a synonym is used, this error cannot occur.

Authorization IDs and authorization-names

An *authorization ID* is a character string that designates a defined set of privileges. Processes can successfully execute SQL statements only if they have the authority to perform the specified functions. A process derives this authority from its authorization IDs. An authorization ID can also designate a user or a group of users, but DB2 does not control this property.

DB2 uses authorization IDs to provide:

- Authorization checking of SQL statements
- Implicit qualifiers for database objects like tables, views, aliases, and indexes

Whenever a connection is established between DB2 and a process, DB2 obtains an authorization ID and passes it to the authorization exit. The list of one or more authorization IDs returned by the exit are used as the authorization IDs of the process.

Every process has exactly one primary authorization ID. Any other authorization IDs of a process are secondary authorization IDs. As explained below, the use of these authorization IDs depends on whether the process is a bind process or an application process.

An *authorization-name* specified in an SQL statement should not be confused with an authorization ID of a process. For example, assume that SMITH is your TSO logon, DYNAMICRULES run behavior is in effect, and you execute the following statements interactively:

```
CREATE TABLE TDEPT LIKE DSN8710.DEPT;  
GRANT SELECT ON TDEPT TO KEENE;
```

Also assume that your site has not replaced the default exit routine for connection authorization and that you have not executed SET CURRENT SQLID. Thus, when the GRANT statement is prepared and executed by SPUFI, the SQL authorization ID is SMITH. KEENE is an authorization name specified in the GRANT statement.

Authorization to execute the GRANT statement is checked against SMITH, and SMITH is the implicit qualifier of TDEPT. The authorization rule is that the privilege

set designated by SMITH must include the SELECT privilege with the GRANT option on SMITH.TDEPT. There is no check involving KEENE.

If SMITH is the implicit qualifier for a statement that contains NAME1, NAME1 identifies the same object as SMITH.NAME1. If the implicit qualifier is other than SMITH, NAME1 and SMITH.NAME1 identify different objects.

Authorization IDs and schema names

An authorization ID that is the same as the name of a schema implicitly has the CREATEIN, ALTERIN, and DROPIN privileges for that schema.

Authorization IDs and statement preparation

A process that creates a plan or package is called a *bind process*. The connection with DB2 is the result of the execution of a BIND or REBIND subcommand. Both subcommands allow for the specification of the authorization ID of the owner of the plan or package. The authorization ID specified as owner must be one of the authorization IDs of the process, unless one of these has SYSADM or SYSCTRL authority. In this case, the owner can be set to any value. BINDAGENT can specify an owner other than himself (or one of his representatives), but it has to be someone that granted him BINDAGENT. The default owner for BIND is the primary authorization ID. The default owner for REBIND is the previous owner of the plan or package (ownership is unchanged if an owner is not explicitly specified). BIND and REBIND are discussed in Chapter 2 of *DB2 Command Reference*.

The authorization ID used for the authorization checking of embedded SQL statements is that of the owner of the plan or package. If an embedded SQL statement refers to tables or views at a DB2 subsystem other than the one at which the plan or package is bound, the authorization checking is deferred until run time. For more information on this, see “Authorization IDs and remote execution” on page 46.

If VALIDATE(BIND) is specified, the privileges required to use or manipulate objects at the DB2 subsystem at which the plan or package is bound must exist at bind time. If the privileges or the referenced objects do not exist and SQLERROR(NOPACKAGE) is in effect, the bind operation is unsuccessful. If SQLERROR(CONTINUE) is specified, then the bind is successful and any statements in error are flagged. If any statements in error are flagged, an error will occur when you attempt to execute them at run time.

If a plan or package is bound with VALIDATE(RUN), authorization checking is still performed at bind time, but the referenced objects and the privileges required to use these objects need not exist at this time. If any privilege required for a statement does not exist at bind time, an authorization check is performed whenever the statement is first executed within a unit of work, and all privileges required for the statement must exist at that time. If any privilege does not exist, execution of the statement is unsuccessful. When the authorization check is performed at run time, it is performed against the plan or package owner, not the SQL authorization ID. For the effect of this option on cursors, see “DECLARE CURSOR” on page 718.

Authorization IDs and dynamic SQL

This discussion applies to dynamic SQL statements that refer to objects at the current server. For those that refer to objects elsewhere, see “Authorization IDs and remote execution” on page 46.

Authorization IDs and authorization-names

Bind option DYNAMICRULES determines the authorization ID that is used for checking authorization when dynamic SQL statements are processed. In addition, the option also controls other dynamic SQL attributes such as the implicit qualifier that is used for unqualified alias, index, table, and view names; the source for application programming options; and whether certain SQL statements can be invoked dynamically.

The set of values for the authorization ID and other dynamic SQL attributes is called the dynamic SQL statement *behavior*. The four possible behaviors are run, bind, define, and invoke. As Table 2 shows, the combination of the value of the DYNAMICRULES bind option and the run-time environment determines which behavior is used. DYNAMICRULES(RUN), which implies run behavior, is the default.

Table 2. How DYNAMICRULES and the run-time environment determine dynamic SQL statement behavior

DYNAMICRULES value	Behavior of dynamic SQL statements	
	Stand-alone program environment	User-defined function or stored procedure environment
RUN	Run behavior	Run behavior
BIND	Bind behavior	Bind behavior
DEFINERUN	Run behavior	Define behavior
DEFINEBIND	Bind behavior	Define behavior
INVOKERUN	Run behavior	Invoke behavior
INVOKEBIND	Bind behavior	Invoke behavior

Note: BIND and RUN values can be specified for both packages and plans. The other values can be specified only for packages.

In the following behavior descriptions, a package that *runs under* a user-defined function or stored procedure package is a package whose associated program meets one of the following conditions:

- The program is called by a user-defined function or stored procedure.
- The program is in a series of nested calls that start with a user-defined function or stored procedure.

Run behavior

DB2 uses the authorization ID of the application process and the SQL authorization ID (the value of special register CURRENT SQLID) for authorization checking of dynamic SQL statements.

A process that uses a plan and its associated packages is called an *application process*. At any time, the SQL authorization ID is the value of CURRENT SQLID. This SQL special register can be initialized by the connection or sign-on exit routine. If the exit does not set a value, the initial value of CURRENT SQLID is the primary authorization ID of the process. You can use the SQL statement SET CURRENT SQLID to change the value of CURRENT SQLID. Unless some authorization ID of the process has SYSADM authority, the new value must be one of the authorization IDs of the process. Thus, CURRENT SQLID usually contains either the primary authorization ID of the process or one of its secondary authorization IDs.

Privilege set: If the dynamically prepared statement is other than an ALTER, CREATE, DROP, GRANT, RENAME, or REVOKE statement, each

privilege required for the statement can be a privilege designated by any authorization ID of the process. Therefore, the privilege set is the union of the set of privileges held by each authorization ID.

If the dynamic SQL statement is an ALTER, CREATE, DROP, GRANT, RENAME, or REVOKE statement, the only authorization ID that is used for authorization checking is the SQL authorization ID. Therefore, the privilege set is the privileges held by that single authorization ID of the process.

Implicit qualification: As explained under “Qualification of unqualified object names” on page 39, when an SQL statement is dynamically prepared, the SQL authorization ID is also used as the implicit qualifier for all unqualified tables, aliases, views, and indexes.

Bind behavior

The same rules that are used to determine the authorization ID for static (embedded) statements are used for dynamic statements. DB2 uses the primary authorization ID of the owner of the package or plan for authorization checking of dynamic SQL statements, as explained in detail under “Authorization IDs and statement preparation” on page 43.

Privilege set: The privilege set is the privileges that are held by the primary authorization ID of the owner of the package or plan.

Implicit qualification: The identifier specified in the QUALIFIER option of the bind command that is used to bind the SQL statements is the implicit qualifier for all unqualified tables, views, aliases, and indexes. If this bind option was not used when the plan or package was created or last rebound, the implicit qualifier is the authorization ID of the owner of the plan or package.

Define behavior

Define behavior applies only if the dynamic SQL statement is in a package that is run as a stored procedure or user-defined function (or *runs under* a stored procedure or user-defined function package), and the package was bound with DYNAMICRULES(DEFINEBIND) or DYNAMICRULES(DEFINERUN). DB2 uses the authorization ID of the stored procedure or user-defined function owner (the definer) for authorization checking of dynamic SQL statements in the application package.

Privilege set: The privilege set is the privileges that are held by the authorization ID of the stored procedure or user-defined function owner.

Implicit qualification: The authorization ID of the stored procedure or user-defined function owner is also the implicit qualifier for unqualified table, view, alias, and index names.

Invoke behavior

Invoke behavior applies only if the dynamic SQL statement is in a package that is run as a stored procedure or user-defined function (or *runs under* a stored procedure or user-defined function package), and the package was bound with DYNAMICRULES(INVOKEBIND) or DYNAMICRULES(INVOKERUN). DB2 uses the authorization ID of the stored procedure or user-defined function invoker for authorization checking of dynamic SQL statements in the application package.

Privilege set: The privilege set is the privileges that are held by the authorization ID of the stored procedure or user-defined function invoker. However, if the invoker is the primary authorization ID of the process or the CURRENT SQLID value, secondary authorization IDs are also checked if

Authorization IDs and authorization-names

they are needed for the required authorization. Therefore, in that case, the privilege set is the union of the set of privileges held by each authorization ID.

Implicit qualification: The authorization ID of the stored procedure or user-defined function invoker is also the implicit qualifier for unqualified table, view, alias, and index names.

Restricted statements when run behavior does not apply: When bind, define, or invoke behavior is in effect, you cannot use the following dynamic SQL statements: ALTER, CREATE, DROP, GRANT, RENAME, and REVOKE.

For more information on authorization and examples of determining authorization for dynamic SQL statements, see Part 3 of *DB2 Administration Guide*. For complete details about the DYNAMICRULES bind option, see *DB2 Command Reference*.

Authorization IDs and remote execution

The authorization rules for remote execution depend on whether the distributed operation is:

- DRDA access with a DB2 for OS/390 and z/OS server and requester
- DRDA access with a server and requester other than DB2
- DB2 private protocol access

DRDA access with DB2 for OS/390 and z/OS only

Any static statement executed using DRDA access is in a package bound at a server other than the local DB2. Before the package can be bound, its owner must have the BINDADD privilege and the CREATE IN privilege for the package's collection. Also required are enough privileges to execute the package's static SQL statements that refer to data on that server. All these privileges are recorded in the DB2 catalog of the server, not that of the local DB2. Such privileges must be granted by GRANT statements executed at the server. This allows the server to control the creation and use of packages that are run from other DBMSs.

A user who invokes an application that has a plan at the local DB2 must have the EXECUTE privilege on the plan recorded in the DB2 catalog of the requester. If the application uses a package bound at a server other than the local DB2 and the package is not a user-defined function, stored procedure, or trigger package, the plan owner must have the EXECUTE privilege on the package recorded in the DB2 catalog of the server. The plan needs no other privilege to execute the package. EXECUTE authority is also required to use a package that is a user-defined function, stored procedure, or trigger package; however, the plan owner is not the required holder of the privilege, as explained in Part 3 (Volume 1) of *DB2 Administration Guide*. In the case of trigger packages, the authorization ID of the SQL statement that activates the trigger must have the EXECUTE privilege on the trigger. Again, all these privileges must be recorded in the DB2 catalog of the server.

Having the appropriate privileges recorded as described above allows the execution of the static SQL statements in the package, and the execution of dynamic SQL statements if DYNAMICRULES bind, define, or invoke behavior is in effect. If DYNAMICRULES run behavior is in effect, the authorization rules for dynamic SQL statements is different. Authorization for the execution of dynamic SQL statements must come from the set of authorization IDs derived during connection processing. An application goes through connection processing when it first connects to a

server or when it reuses a CICS or IMS thread that has a different primary authorization ID. For details on connection processing, see Part 3 (Volume 1) of *DB2 Administration Guide*.

If an application uses Recoverable Resources Manager Services attachment facility (RRSAF) and has no plan, authority to execute the package is determined in the same way as when the requester is not DB2 for OS/390 and z/OS, which is described next under “DRDA access with a server or requester other than DB2.”

DRDA access with a server or requester other than DB2

DB2 for OS/390 and z/OS as the server: If the requester is not a DB2 for OS/390 and z/OS subsystem, there is no DB2 application plan involved. In this case, the privilege set of the authorization ID, which is determined by the DYNAMICRULES behavior, must have the EXECUTE privilege on the package. Dynamic SQL statements in the package are executed according to the DYNAMICRULES behavior, as described in “Authorization IDs and dynamic SQL” on page 43.

DB2 for OS/390 and z/OS as the requester: The authorization rules for remote execution are those of the server.

DB2 private protocol access

Any statement that refers to a table or view at a DB2 subsystem other than the current server and is bound with bind option DBPROTOCOL(PRIVATE) is executed using DB2 private protocol access. Such statements are processed as deferred embedded SQL statements. The additional cost of the dynamic bind occurs once for every unit of work where the statement is executed. Authorization to execute such statements is checked against the owner of a plan or package. Authorization IDs for executing dynamic statements are handled just as they are for DRDA access. In either case, the pertinent privileges must be recorded in the catalog of the DBMS that executes the statement.

Authorization ID translations

Three authorization IDs played roles in the foregoing discussion. These are the user's primary authorization ID and those for the owner of the application plan and the owner of a package. Each of these is sent to the remote DBMS. And each may undergo translations before it is used.

For example, a user known as SMITH at the local DBMS could be known, after translation, as JONES at the server. Likewise, a package owner known as GRAY could be known as WINTERS at the server. If so, JONES or WINTERS would be used, instead of SMITH or GRAY, to determine the authorization ID for dynamic SQL statements in the package. If the DYNAMICRULES run behavior applies, JONES, who is executing the dynamic statement at the server, is used. If DYNAMICRULES bind behavior applies, WINTERS, the package owner at the server, is used.

Two sets of communications database (CDB) catalog tables control the translations. One set is at the local DB2, and the other set is at the remote DB2. Translation can take place at either or both sites. For how to use and maintain these tables, see Part 3 (Volume 1) of *DB2 Administration Guide*.

Other security measures

The fact that DB2 authority requirements are satisfied does not guarantee that a user has access to a given server. Other security measures may also come into play. For example, requests to execute remote SQL statements could be denied

Authorization IDs and authorization-names

based on Resource Access Control Facility (RACF®) considerations. Developing such security measures is discussed in Part 3 (Volume 1) of *DB2 Administration Guide*.

Data types

The smallest unit of data that can be manipulated in SQL is called a *value*. How values are interpreted depends on the data type of their source. The sources of values are:

- Columns
- Constants
- Expressions
- Functions
- Host variables
- Special registers

DB2 supports both IBM-supplied data types (built-in data types) and user-defined data types (distinct types). This section describes the built-in data types. For a description of distinct types, see “Distinct types” on page 60.

Figure 5 shows the built-in data types that DB2 supports.

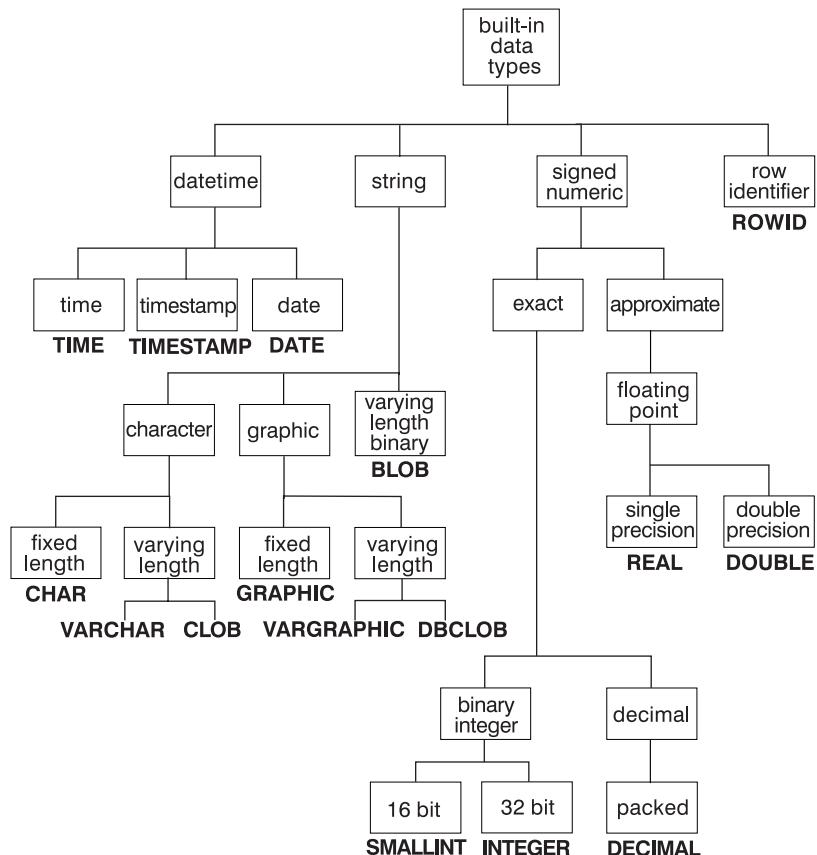


Figure 5. Built-in data types supported by DB2

Nulls: All data types include the null value. Distinct from all nonnull values, the null value is a special value that denotes the absence of a (nonnull) value. Although all data types include the null value, some sources of values cannot provide the null value. For example, constants, columns that are defined as NOT NULL, and special

registers cannot contain null values; the COUNT and COUNT_BIG functions cannot return a null value; and ROWID columns cannot store a null value although a null value can be returned for a ROWID column as the result of a query.

Character strings

A *character string* is a sequence of bytes. The length of the string is the number of bytes in the sequence. If the length is zero, the value is called the *empty string*. The empty string should not be confused with the null value.

Except for C NUL-terminated strings, the length of a varying-length string is specified by the value of its length control field. For varying-length character strings, the length control field specifies the number of bytes.

Each character string has a subtype of SBCS, MIXED, or BIT.

- The bytes of a character string with subtype SBCS represent characters from a single-byte character set (SBCS). Such strings are called SBCS data.
- The bytes of a character string with subtype MIXED can represent a mixture of characters from a multibyte character set (MBCS), which includes characters from a single-byte character set (SBCS). Strings that may contain MBCS characters are called *mixed data*. EBCDIC mixed data may contain shift characters, which are not MBCS data. Unicode data is always MIXED, regardless of the MIXED option on the DSNTIPF install option panel.
- The bytes of a character string with BIT subtype do not represent characters; therefore, character conversion never occurs for these strings. Such strings are called BIT data.

Character strings with a CLOB data type can only have an SBCS or MIXED subtype.

Character subtypes provide a simple and portable way of specifying the CCSID of a character string column. The subtype is implicitly or explicitly specified when the column is defined in a CREATE or ALTER TABLE statement. The default depends on which encoding scheme is in use. With Unicode, the default is MIXED. With ASCII and EBCDIC, the default is SBCS or MIXED depending on the value of the field MIXED DATA on installation panel DSNTIPF. The following list shows the CCSID for each subtype:

BIT	The CCSID is X'FFFF' (65535).
SBCS	The CCSID is the system CCSID for SBCS data.
MIXED	The CCSID is the system CCSID for mixed data.

The FOREIGNKEY column of the SYSCOLUMNS catalog table stores information about the subtype of a character string column. An administrator can update this column to change the subtype of existing columns. DB2 does not ensure that the bytes of a character string are consistent with its CCSID and does not use CCSIDs for any purpose other than character conversion.

The effect of encoding schemes on DBCS characters in mixed strings

The method of representing DBCS characters within a mixed string differs among the encoding schemes.

- ASCII reserves a set of code points for SBCS characters and another set as the first half of DBCS characters. Upon encountering the first half of a DBCS character, the system knows that it is to read the next byte in order to obtain the complete character.

Data types

- EBCDIC makes use of two special code points:
 - A shift-out character (X'0E') to introduce a string of DBCS characters.
 - A shift-in character (X'0F') to end a string of DBCS characters.
- DBCS sequences within mixed data strings are recognized as the string is read from left to right. At any time, the recognizer is in SBCS mode or DBCS mode. In SBCS mode, which is the initial mode, any byte other than a shift-out is interpreted as an SBCS character. When a shift-out is read, the recognizer enters DBCS mode. In DBCS mode, the next byte and every second byte after that byte is interpreted as the first byte of a DBCS character unless it is a shift character. If the byte is a shift-out, an error occurs. If the byte is a shift-in, the recognizer returns to SBCS mode. An error occurs if the recognizer is in DBCS mode after processing the last byte of the string. Because of the shift characters, EBCDIC mixed data requires more storage than ASCII mixed data.
- UTF-8 is a varying-length encoding of byte sequences with the high bits indicating the part of the sequence to which a byte belongs. The first byte indicates the number of bytes to follow in a byte sequence.
- UTF-16 uses a fixed 16-bit code assignment for each character, except for characters encoded by surrogates, which consist of a pair of 16-bit values.

Examples

元 gen 火 ki

CHAR(9) in ASCII.

\$0元 \$1 gen \$0火 \$1 ki

CHAR(13) in EBCDIC.

元 gen 火 ki

CHAR(11) in Unicode.

Because of the differences of the representation of mixed data strings in ASCII and EBCDIC, mixed data is not transparently portable. To minimize the effects of these differences, use varying-length strings in applications that require mixed data and operate on both ASCII and EBCDIC data.

DB2 and SBCS defaults

For a DB2 installation with SBCS, the default encoding scheme is ASCII or EBCDIC and the value of field MIXED DATA on installation panel DSNTIPF is NO. The subtype of character strings is SBCS with SBCS being the default. The values of fields ASCII CODED CHAR SET or EBCDIC CODED CHAR SET determine the system CCSID that identifies the SBCS coded character set used for that DB2. The default for Unicode data is mixed.

DB2 and DBCS defaults

For a DB2 installation with DBCS, the default encoding scheme is ASCII or EBCDIC and the value of field MIXED DATA on installation panel DSNTIPF is YES. For ASCII and EBCDIC, the subtype of character strings is SBCS, BIT, or MIXED, with MIXED being the default. The values of fields ASCII CODED CHAR SET or EBCDIC CODED CHAR SET determine the system CCSIDs used for SBCS data, mixed data, and graphic data. For Unicode, the MIXED DATA field is not relevant because the subtype for Unicode data is always mixed.

ASCII and EBCDIC are structured so that some data is represented in SBCS characters and some data is represented in DBCS or mixed (MBCS) characters. Mixed data allows both SBCS and DBCS characters as does UTF-8, in a sense. In UTF-8, A–Z, a–z, 0–9, and a few other characters are SBCS while everything else is MBCS. UTF-16 is different. In UTF-16, A–Z, a–z, 0–9, and most everything else, except surrogate characters, are DBCS. Surrogates consist of a pair of 16-bit values.

A mixed data string can have zero or more sequences of SBCS characters and zero or more sequences of multiple-byte characters. Each EBCDIC multiple-byte sequence must be preceded by the shift-out control character (X'0E') and followed by the shift-in control character (X'0F'). There must be an even number of bytes between the shift characters and each pair of bytes is assumed to represent a DBCS character.

DB2 recognizes DBCS sequences within mixed data strings when performing character-sensitive operations at a DB2 installation with DBCS. These operations include parsing, character conversion, and the pattern matching specified by the LIKE predicate. DB2 also recognizes DBCS sequences:

- In source language statements, static SQL statements, and deferred embedded SQL statements if the GRAPHIC precompiler option is implicitly or explicitly specified
- In dynamic SQL statements if DYNAMICRULES bind, define, or invoke behavior is in effect, the value of installation panel field USE FOR DYNAMICRULES is NO, and the GRAPHIC precompiler option is implicitly or explicitly specified

Fixed-length character strings

All the values of a column with a fixed-length character string data type have the same length, which is determined by the length attribute of the column. The length attribute must be between 1 and 255 inclusive. Every fixed-length string column is a *short string column*. A fixed-length character string column can also be called a CHAR or CHARACTER column.

Varying-length character strings

The types of varying-length character strings are:

- VARCHAR (or synonyms CHAR VARYING and CHARACTER VARYING)⁶
- CLOB (or synonyms CHAR LARGE OBJECT and CHARACTER LARGE OBJECT)

The values of a column with any one of these string types can have different lengths. The length attribute of the column determines the maximum length a value can have.

For a VARCHAR column, the length attribute must be between 1 and m inclusive, where m is determined by the maximum record size as described in “Maximum record size” on page 676 in the description of the CREATE TABLE statement. For a CLOB column, the length attribute must be between 1 and 2 147 483 647 inclusive. (2 147 483 647 is 2 gigabytes minus 1 byte.) For more information about CLOBS, see “Large objects (LOBs)” on page 53.

6. The syntax of the ALTER and CREATE TABLE statements allows a column to be defined as LONG VARCHAR as an alternative for VARCHAR(a) where a is the maximum number of characters that is associated with the column. However, after processing the CREATE or ALTER TABLE statement, DB2 considers the column to be VARCHAR(a).

Data types

A VARCHAR column with a maximum length that is greater than 255 bytes or a CLOB column of any length is a *long string column*. For the restrictions that apply to the use of long string columns, see “[Restrictions using long strings](#)” on page 54.

Character string host variables

Host variables with CHAR and CLOB string types can be defined in all host languages. (In C, CHAR string variables are limited to a length of 1.) Host variables with a VARCHAR string type can be defined in all host languages except Fortran. In Assembler, C, and COBOL, VARCHAR string variables are simulated as described in Part 2 of *DB2 Application Programming and SQL Guide*. In C, VARCHAR string variables can also be represented by NUL-terminated strings.

A VARCHAR string variable with a maximum length that is greater than 255 bytes or a CLOB string variable of any length is a *long string variable*. Long string variables are subject to the same restrictions as long string columns. For information, see “[Restrictions using long strings](#)” on page 54.

Graphic strings

A *graphic string* is a sequence of DBCS characters. The length of the string is the number of characters in the sequence. Like character strings, graphic strings can be empty. An empty string should not be confused with the null value.

Fixed-length graphic strings

All the values of a column with a fixed-length graphic string data type have the same length, which is determined by the length attribute of the column. The length attribute must be between 1 and 127 inclusive. Every fixed-length graphic string column is a short string column. A fixed-length graphic string column can also be called a GRAPHIC column.

Varying-length graphic strings

The types of varying-length graphic strings are:

- VARGRAPHIC⁷
- DBCLOB

The values of a column with any one of these string types can have different lengths. The length attribute of the column determines the maximum length a value can have. For varying-length graphic strings, the length control field specifies the number of MBCS (multi-byte) characters, with each character represented by a pair of bytes, except in UTF-16 where surrogates use four bytes for each character.

For VARGRAPHIC columns, the length attribute of the column must be between 1 and m inclusive, where m is determined by the maximum record size as described in “[Maximum record size](#)” on page 676 in the description of the CREATE TABLE statement. For DBCLOB columns, the length attribute must be between 1 and 1 073 741 823 inclusive. In all cases, the length control field of a varying-length graphic string indicates the number of characters, not bytes. For UTF-16, two DBCS characters can be used to describe a glyph, but the characters are still considered DBCS characters. For more information about DBCLOBs, see “[Large objects \(LOBs\)](#)” on page 53.

7. The syntax of the ALTER and CREATE TABLE statements allows a column to be defined as LONG VARGRAPHIC as an alternative for VARGRAPHIC(a) where a is the maximum number of characters that is associated with the column. However, after processing the CREATE or ALTER TABLE statement, DB2 considers the column to be VARGRAPHIC(a).

A VARGRAPHIC column with a maximum length that is greater than 127 characters or a DBCLOB column of any length is a *long string column*. For the restrictions that apply to the use of long string columns, see “Restrictions using long strings” on page 54.

Graphic string host variables

Host variables with a graphic string type can be defined in all host languages except Fortran.

A VARGRAPHIC string variable with a maximum length that is greater than 127 characters or a DBCLOB string variable of any length is a *long string variable*. Long string variables are subject to the same restrictions as long string columns. For information, see “Restrictions using long strings” on page 54.

Binary strings

A *binary string* is a sequence of bytes. The length of a binary string (BLOB string) is the number of bytes in the sequence. The CCSID is X'FFFF' (65535).

For a BLOB column, the length attribute must be between 1 and 2 147 483 647 inclusive. (2 147 483 647 is 2 gigabytes minus 1 byte.)

A host variable with a BLOB string type can be defined in all host languages.

For more information about BLOBs, see “Large objects (LOBs).”

Large objects (LOBs)

The term *large object (LOB)* refers to any of the following data types:

- | | |
|---------------|--|
| CLOB | A <i>character large object (CLOB)</i> is a varying-length string with a maximum length of 2 147 483 647 bytes (2 gigabytes minus 1 byte). A CLOB is designed to store large SBCS data or mixed data, such as lengthy documents. For example, you can store information such as an employee resume, the script of a play, or the text of novel in a CLOB. Alternatively, you can store such information in UTF-8 in a mixed CLOB. A CLOB is a varying-length character string. |
| DBCLOB | A <i>double-byte character large object (DBCLOB)</i> is a varying-length string with a maximum length of 1 073 741 823 double-byte characters. A DBCLOB is designed to store large DBCS data. For example, you could store the information mentioned for CLOB (an employee resume, the script for a play, or the text of a novel) in UTF-16 in a DBCLOB. A DBCLOB is a varying-length graphic string. |
| BLOB | A <i>binary large object (BLOB)</i> is a varying-length string with a maximum length of 2 147 483 647 bytes (2 gigabytes minus 1 byte). A BLOB is designed to store non-traditional data such as pictures, voice, and mixed media. BLOBs can also store structured data for use by distinct types and user-defined functions. A BLOB is considered to be a binary string. |

Although BLOB strings and FOR BIT DATA character strings might be used for similar purposes, the two data types are not compatible. The BLOB function can be used to change a FOR BIT DATA character string into a BLOB string.

Data types

Each LOB column is a long string column, and each LOB string variable is a long string variable. For information on the restrictions that apply to the use of long strings and the additional restrictions that apply only to LOBs, see “[Restrictions using long strings](#).”

When an application does not need a LOB value to be stored in application memory, the application can use a *large object locator* (LOB locator) to reference the LOB value.

A LOB locator is a host variable with a value that represents a single LOB value in the database server. A LOB locator can also represent a LOB expression, such as:

```
SUBSTR( <lob 1> CONCAT <lob 2> CONCAT <lob 3>, <start>, <length> )
```

For information on manipulating LOBs with LOB locators, see Part 3 of *DB2 Application Programming and SQL Guide*.

Restrictions using long strings

A long string has one of the following varying-length string data types:

- *For character strings.* A VARCHAR string with a maximum length that is greater than 255 bytes or any CLOB string.
- *For graphic strings.* A VARGRAPHIC string with a maximum length that is greater than 127 characters or any DBCLOB string.
- *For binary strings.* Any BLOB string.

Table 3 indicates the contexts in which long strings cannot be referenced. The restrictions differ slightly for long strings with LOB data types (CLOB, DBCLOB, and BLOB).

Table 3. Contexts in which long strings cannot be referenced

Context of usage	VARCHAR (>255) or VARGRAPHIC (>127)	LOB (CLOB, DBCLOB, or BLOB)
A GROUP BY clause	Not allowed	Not allowed
An ORDER BY clause	Not allowed	Not allowed
A CREATE INDEX statement	Not allowed	Not allowed
A SELECT DISTINCT statement	Not allowed	Not allowed
A subselect of a UNION without the ALL keyword	Not allowed	Not allowed
A <i>host variable</i> in a EXECUTE IMMEDIATE or a PREPARE statement	—	Not allowed
Predicates	Cannot be used in any predicate except EXISTS, LIKE, and NULL. This restriction includes a <i>simple-when-clause</i> in a CASE expression. <i>expression WHEN expression</i> in a <i>simple-when-clause</i> is equivalent to a predicate with <i>expression=expression</i> .	Cannot be used in any predicate except EXISTS, LIKE, and NULL. This restriction includes a <i>simple-when-clause</i> in a CASE expression. <i>expression WHEN expression</i> in a <i>simple-when-clause</i> is equivalent to a predicate with <i>expression=expression</i> .
The definition of primary, unique, and foreign keys	Not allowed	Not allowed
Check constraints	—	Cannot be specified for a LOB column
Field procedure	—	Cannot be specified for a LOB column.

Table 3. Contexts in which long strings cannot be referenced (continued)

Context of usage	VARCHAR (>255) or VARGRAPHIC (>127)	LOB (CLOB, DBCLOB, or BLOB)
Parameters of built-in functions	Some functions that allow varying-length character strings, varying-length graphic strings, or both types of strings as input arguments do not support VARCHAR or VARGRAPHIC long strings, CLOB or DBCLOB strings, or both as input. See the description of the individual functions in Chapter 3, “Functions,” on page 155 for the data types that are allowed as input to each function.	
Distributed data applications	—	LOB columns cannot be used if remote access is performed with DB2 private protocol access.

Numbers

The numeric data types are binary integer, floating-point, and decimal. Binary integer includes small integer and large integer. Floating-point includes single precision and double precision. Binary numbers are exact representations of integers, decimal numbers are exact representations of real numbers, and floating-point numbers are approximations of real numbers.

All numbers have a sign and a precision. When the value of a column or the result of an expression is a decimal or floating-point zero, its sign is positive. The precision of binary integers and decimal numbers is the total number of binary or decimal digits excluding the sign. The precision of floating-point numbers is either single or double, referring to the number of hexadecimal digits in the fraction.

Small integer (SMALLINT)

A *small integer* is a System/390 binary integer with a precision of 15 bits. The range of small integers is -32768 to +32767.

Large integer (INTEGER)

A *large integer* is a System/390 binary integer with a precision of 31 bits. The range of large integers is -2147483648 to +2147483647.

Single precision floating-point (REAL)

A *single precision floating-point* number is a System/390 short (32 bits) floating-point number. The range of single precision floating-point numbers is about -7.2E+75 to 7.2E+75. In this range, the largest negative value is about -5.4E-79, and the smallest positive value is about 5.4E-079.

Double precision floating-point (DOUBLE or FLOAT)

A *double precision floating-point* number is a System/390 long (64 bits) floating-point number. The range of double precision floating-point numbers is about -7.2E+75 to 7.2E+75. In this range, the largest negative value is about -5.4E-79, and the smallest positive value is about 5.4E-079.

Decimal (DECIMAL or NUMERIC)

A *decimal* number is a System/390 packed decimal number with an implicit decimal point. The position of the decimal point is determined by the precision and the scale of the number. The scale, which is the number of digits in the fractional part of the number, cannot be negative or greater than the precision. The maximum precision is 31 digits.

All values of a decimal column have the same precision and scale. The range of a decimal variable or the numbers in a decimal column is -n to +n, where n is the

Data types

largest positive number that can be represented with the applicable precision and scale. The maximum range is $1 - 10^{31}$ to $10^{31} - 1$.

String representations of numbers

Values whose data types are small integer, large integer, floating-point, and decimal are stored in an internal form that is transparent to the user of SQL. But string representations of numbers can be used in some contexts. A valid string representation of a number must conform to the rules for numeric constants. For more information, see “Constants” on page 79.

The encoding scheme in use determines what type of strings may be used for string representation of numbers. For ASCII and EBCDIC, a string representation of a number must be a character string. For UNICODE, a string representation of a number can be either a character string or a graphic string. Thus, the only time a graphic string can be used for a number is when the encoding scheme is UNICODE.

Numeric host variables

Binary integer variables can be defined in all host languages.

Floating-point variables can be defined in all host languages. All languages support System/390 floating-point format. Assembler, C, and C++ also support IEEE floating-point format. In assembler, C, and C++ programs, the precompiler option FLOAT tells DB2 whether floating-point variables contain data in System/390 floating-point format or IEEE floating-point format.

Decimal variables can be defined in all host languages except Fortran. In COBOL, decimal numbers can be represented in the packed decimal format used for columns or in the format denoted by DISPLAY SIGN LEADING SEPARATE.

Datetime values

The datetime data types are described in the following sections. Such values are neither strings nor numbers. Nevertheless, datetime values can be used in certain arithmetic and string operations and are compatible with certain strings. Moreover, strings can represent datetime values, as discussed in “String representations of datetime values” on page 57.

Date

A *date* is a three-part value (year, month, and day) designating a point in time using the Gregorian calendar, which is assumed to have been in effect from the year 1 A.D.⁸ The range of the year part is 0001 to 9999. The range of the month part is 1 to 12. The range of the day part is 1 to 28, 29, 30, or 31, depending on the month and year.

The internal representation of a date is a string of 4 bytes. Each byte consists of two packed decimal digits. The first 2 bytes represent the year, the third byte the month, and the last byte the day.

The length of a DATE column as described in the catalog is the internal length, which is 4 bytes. The length of a DATE column as described in the SQLDA is the external length, which is 10 bytes unless a date exit routine was specified when your DB2 subsystem was installed. (Writing a date exit routine is described in

8. Historical dates do not always follow the Gregorian calendar. Dates between 1582-10-04 and 1582-10-15 are accepted as valid dates although they never existed in the Gregorian calendar.

Appendix B (Volume 2) of *DB2 Administration Guide*.) In that case, the string format of a date can be up to 255 bytes in length. Accordingly, DCLGEN⁹ defines fixed-length string variables for DATE columns with a length equal to the value of the field LOCAL DATE LENGTH on installation panel DSNTIP4, or a length of 10 bytes if a value for the field was not specified.

Time

A *time* is a three-part value (hour, minute, and second) designating a time of day using a 24-hour clock. The range of the hour part is 0 to 24. The range of the minute and second parts is 0 to 59. If the hour is 24, the minute and second parts are both zero.

The internal representation of a time is a string of 3 bytes. Each byte consists of two packed decimal digits. The first byte represents the hour, the second byte the minute, and the last byte the second.

The length of a TIME column as described in the catalog is the internal length which is 3 bytes. The length of a TIME column as described in the SQLDA is the external length which is 8 bytes unless a time exit routine was specified when your DB2 subsystem was installed. (Writing a date exit routine is described in Appendix B (Volume 2) of *DB2 Administration Guide*.) In that case, the string format of a time can be up to 255 bytes in length. Accordingly, DCLGEN⁹ defines fixed-length string variables for TIME columns with a length equal to the value of the field LOCAL TIME LENGTH on installation panel DSNTIP4, or a length of 8 bytes if a value for the field was not specified.

Timestamp

A *timestamp* is a seven-part value (year, month, day, hour, minute, second, and microsecond) that represents a date and time as defined previously, except that the time includes a fractional specification of microseconds.

The internal representation of a timestamp is a string of 10 bytes, each of which consists of two packed decimal digits. The first 4 bytes represent the date, the next 3 bytes the time, and the last 3 bytes the microseconds.

The length of a TIMESTAMP column as described in the catalog is the internal length which is 10 bytes. The length of a TIMESTAMP column as described in the SQLDA is the external length which is 26 bytes. DCLGEN therefore defines 26-byte, fixed-length string variables for TIMESTAMP columns.

String representations of datetime values

Values whose data types are date, time, or timestamp are represented in an internal form that is transparent to the user of SQL. But dates, times, and timestamps can also be represented by strings. These representations directly concern the SQL user because there are no special SQL constants for datetime values and no host variables with a data type of date, time, or timestamp.

The encoding scheme in use determines what type of strings may be used for string representation of datetime values. For ASCII and EBCDIC, a string representation of a datetime value must be a character string. For UNICODE, a string representation of a datetime value can be either a character string or a graphic string. Thus, the only time a graphic string can be used for a datetime value is when the encoding scheme is UNICODE.

9. DCLGEN is a DB2 DSN subcommand for generating table declarations for designated tables or views. The declarations are stored in MVS data sets, for later inclusion in DB2 source programs.

Data types

For retrieval, datetime values must be assigned to string variables. When a date or time is assigned to a variable, the string format is determined by a precompiler option or subsystem parameter. When a string representation of a datetime value is used in other operations, it is converted to a datetime value. However, this can be done only if the string representation is recognized by DB2 or an exit provided by the installation and the other operand is a compatible datetime value. An input string representation of a date or time value with LOCAL specified can be any short string. The following sections describe the string formats that are recognized by DB2.

Datetime values that are represented by strings can appear in contexts requiring values whose data types are date, time, timestamp by using the DATE, TIME, or TIMESTAMP functions.

Date strings: A string representation of a date is a string that starts with a digit and has a length of at least 8 characters. Trailing blanks can be included, leading blanks are not allowed, and leading zeros can be omitted in the month and day portions.

Table 4 shows the valid string formats for dates. Each format is identified by name and includes an associated abbreviation (for use by the CHAR function) and an example of its use. For an installation-defined date string format, the format and length must have been specified when DB2 was installed. They cannot be listed here.

Table 4. Formats for string representations of dates

Format name	Abbreviation	Date format	Example
International Standards Organization	ISO	yyyy-mm-dd	1987-10-12
IBM USA standard	USA	mm/dd/yyyy	10/12/1987
IBM European standard	EUR	dd.mm.yyyy	12.10.1987
Japanese industrial standard Christian era	JIS	yyyy-mm-dd	1987-10-12
Installation-defined	LOCAL	Any installation-defined form	—

Note: For LOCAL, the date exit for ASCII data is DSNXVDTA, the date exit for EBCDIC is DSNXVDTX, and the date exit for Unicode is DSNXVDTU.

Time strings: A string representation of a time is a string that starts with a digit, and has a length of at least 4 characters. Trailing blanks can be included, leading blanks are not allowed, and leading zeros can be omitted in the hour part of the time; seconds can be omitted entirely. If you choose to omit seconds, an implicit specification of 0 seconds is assumed. Thus 13.30 is equivalent to 13.30.00.

Table 5 on page 59 shows the valid string formats for times. Each format is identified by name and includes an associated abbreviation (for use by the CHAR function) and an example of its use. In the case of an installation-defined time string format, the format and length must have been specified when your DB2 subsystem was installed. They cannot be listed here.

10. This is an earlier version of the ISO format. JIS can be used to get the current ISO format.

Table 5. Formats for string representations of times

Format name	Abbreviation	Time format	Example
International Standards Organization ¹⁰	ISO	hh.mm.ss	13.30.05
IBM USA standard	USA	hh:mm AM or PM	1:30 PM
IBM European standard	EUR	hh.mm.ss	13.30.05
Japanese industrial standard Christian era	JIS	hh:mm:ss	13:30:05
Installation-defined	LOCAL	Any installation-defined form	—

Note: For LOCAL, the date exit for ASCII data is DSNXVDTA, the date exit for EBCDIC is DSNXVDTX, and the date exit for Unicode is DSNXVDTU.

In the USA format:

- The minutes can be omitted, thereby specifying 00 minutes. For example, 1 PM is equivalent to 1:00 PM.
- The letters A, M, and P can be lowercase.
- A single blank must precede the AM or PM.
- The hour must not be greater than 12 and cannot be 0 except for the special case of 00:00 AM.

Using the ISO format of the 24-hour clock, the correspondence between the USA format and the 24-hour clock is as follows:

- 12:01 AM through 12:59 AM correspond to 00.01.00 through 00.59.00
- 01:00 AM through 11:59 AM correspond to 01.00.00 through 11.59.00
- 12:00 PM (noon) through 11:59 PM correspond to 12.00.00 through 23.59.00
- 12:00 AM (midnight) corresponds to 24.00.00
- 00:00 AM (midnight) corresponds to 00.00.00

Timestamp strings: A string representation of a timestamp is a string that starts with a digit and has a length of at least 16 characters. The complete string representation of a timestamp has the form *yyyy-mm-dd-hh.mm.ss.nnnnnn*. Trailing blanks can be included, leading blanks are not allowed, and leading zeros can be omitted in the month, day, and hour part of the timestamp; trailing zeros can be truncated or omitted entirely from microseconds. If you choose to omit any digit of the microseconds portion, an implicit specification of 0 is assumed. Thus, 1990-3-2-8.30.00.10 is equivalent to 1990-03-02-08.30.00.100000.

Restrictions on the use of local datetime formats

The following rules apply to the character string representation of dates and times:

For input: In distributed operations, DB2 as a server uses its local date or time routine to evaluate host variables and literals. This means that character string representation of dates and times can be:

- One of the standard formats
- A format recognized by the server's local date/time exit

For output: With DRDA access, DB2 as a server returns date and time host variables in the format defined at the server. With DB2 private protocol access, DB2 as a server returns date and time host variables in the format defined at the requesting system. When LOCAL format is defined, the format that is returned is always ISO. To have date and time host variables or columns returned in another

Data types

```
# format, use CHAR(date-expression, XXXX) where XXXX is JIS, EUR, USA, ISO, or  
# LOCAL to explicitly specify the specific format.
```

For BIND PACKAGE COPY: When binding a package using the COPY option, DB2 uses the ISO format for output values unless the SQL statement explicitly specifies a different format. Input values can be specified in the format described above under “For input” on page 59.

Row ID values

A *row ID* is a value that uniquely identifies a row in a table. A column or a host variable can have a row ID data type. A ROWID column enables queries to be written that navigate directly to a row in the table. Each value in a ROWID column must be unique, and DB2 maintains the values permanently, even across table space reorganizations. When a row is inserted into the table, DB2 generates a value for the ROWID column unless one is supplied. If a value is supplied, it must be a valid row ID value that was previously generated by DB2 and the column must be defined as GENERATED BY DEFAULT. Users cannot update the value of a ROWID column.

The internal representation of a row ID value is transparent to the user. The value is never subject to translation because it is considered to contain BIT data. The length of a ROWID column as described in the LENGTH column of catalog table SYSCOLUMNS is the internal length, which is 17 bytes. The length as described in the LENGTH2 column of catalog table SYSCOLUMNS is the external length, which is 40 bytes.

In a distributed data environment, the row ID data type is not supported for DB2 private protocol access. For information about using row IDs, see *DB2 Application Programming and SQL Guide*.

Distinct types

A *distinct type* is a user-defined data type that shares its internal representation with a built-in data type (its *source type*), but is considered to be a separate and incompatible data type for most operations. For example, the semantics for a picture type, a text type, and an audio type that all use the built-in data type BLOB for their internal representation are quite different. A distinct type is created with the SQL statement CREATE DISTINCT TYPE.

For example, the following statement creates a distinct type named AUDIO:

```
CREATE DISTINCT TYPE AUDIO AS BLOB (1M);
```

Although AUDIO has the same representation as the built-in data type BLOB, it is a separate data type that is not comparable to a BLOB or to any other data type. This inability to compare AUDIO to other data types allows functions to be created specifically for AUDIO and assures that these functions cannot be applied to other data types.

The name of a distinct type is qualified with a schema name. The implicit schema name for an unqualified name depends on the context in which the distinct type appears. If an unqualified distinct type name is used:

- In a CREATE DISTINCT TYPE or the object of DROP, COMMENT ON, GRANT, or REVOKE statement, DB2 uses the normal process of qualification by authorization ID to determine the schema name.

- In any other context, DB2 uses the SQL path to determine the schema name. DB2 searches the schemas in the path, in sequence, and selects the first schema in the path such that the distinct type exists in the schema and the user has authorization to use the data type. For a description of the SQL path, see “Schemas and the SQL path” on page 40.

A distinct type does not automatically acquire the functions and operators of its source type because they might not be meaningful. (For example, it might make sense for a “length” function of the AUDIO type to return the length in seconds rather than in bytes.) Instead, distinct types support *strong typing*. Strong typing ensures that only the functions and operators that are explicitly defined on a distinct type can be applied to that distinct type. However, a function or operator of the source type can be applied to the distinct type by creating an appropriate user-defined function. The user-defined function must be sourced on the existing function that has the source type as a parameter. For example, the following series of SQL statements shows how to create a distinct type named MONEY based on data type DECIMAL(9,2), how to define the + operator for the distinct type, and how the operator might be applied to the distinct type:

```
CREATE DISTINCT TYPE MONEY AS DECIMAL(9,2) WITH COMPARISONS;

CREATE FUNCTION "+"(MONEY,MONEY)
RETURNS MONEY
SOURCE SYSIBM."+"(DECIMAL(9,2),DECIMAL(9,2));

CREATE TABLE SALARY_TABLE
(SALARY MONEY,
 COMMISSION MONEY);

SELECT SALARY + COMMISSION FROM SALARY_TABLE;
```

A distinct type is subject to the same restrictions as its source type. For example, a table can only have one ROWID column. Therefore, a table with a ROWID column cannot also have a column with distinct type that is sourced on a row ID.

The comparison operators are automatically generated for distinct types, except those that are sourced on a CLOB, DBCLOB, or BLOB. In addition, DB2 automatically generates functions for every distinct type that support casting from the source type to the distinct type and from the distinct type to the source type. For example, for the AUDIO type created above, these are generated cast functions:

```
FUNCTION schema-name.BLOB (schema-name.AUDIO) RETURNS SYSIBM.BLOB (1M)
FUNCTION schema-name.AUDIO (SYSIBM.BLOB (1M)) RETURNS schema-name.AUDIO
```

In a distributed data environment, distinct types are not supported for DB2 private protocol access.

Promotion of data types

Data types can be classified into groups of related data types. Within such groups, an order of precedence exists in which one data type is considered to precede another data type. This precedence enables DB2 to support the *promotion* of one data type to another data type that appears later in the precedence order. For example, DB2 can promote the data type CHAR to VARCHAR and the data type INTEGER to DOUBLE PRECISION; however, DB2 cannot promote a CLOB to a VARCHAR.

DB2 considers the promotion of data types when:

- Performing function resolution (see “Function resolution” on page 107)

Promotion of data types

- Casting distinct types (see “Casting between data types”)
- Assigning distinct types to built-in data types (see “Distinct type assignments” on page 71)

For each data type, Table 6 shows the precedence list (in order) that DB2 uses to determine the data types to which the data type can be promoted. The table indicates that the best choice is the same data type and not promotion to another data type.

Table 6. Precedence of data types

Data type ^{1,2}	Data type precedence list (in best-to-worst order)
CHAR or GRAPHIC	CHAR or GRAPHIC, VARCHAR or VARGRAPHIC, CLOB or DBCLOB
VARCHAR or VARGRAPHIC	VARCHAR or VARGRAPHIC, CLOB or DBCLOB
CLOB or DBCLOB	CLOB or DBCLOB
BLOB	BLOB
SMALLINT	SMALLINT, INTEGER, decimal, real, double
INTEGER	INTEGER, decimal, real, double
decimal	decimal, real, double
real	real, double
double	double
DATE	DATE
TIME	TIME
TIMESTAMP	TIMESTAMP
ROWID	ROWID
A distinct type	The same distinct type

Notes:

1. The data types in lowercase letters represent the following data types:

decimal DECIMAL(p,s) or NUMERIC(p,s)

real REAL or FLOAT(*n*) where *n* is not greater than 21

double DOUBLE, DOUBLE PRECISION, FLOAT or FLOAT(*n*) where *n* is greater than 21

2. Other synonyms for the listed data types are considered to be the same as the synonym listed.

Casting between data types

There are many occasions when a value with a given data type needs to be *cast* (changed) to a different data type or to the same data type with a different length, precision, or scale. Data type promotion, as described in “Promotion of data types” on page 61, is one example of when a value with one data type needs to be cast to a new data type. A data type that can be changed to another data type is *castable* from the source data type to the target data type.

The casting of one data type to another can occur implicitly or explicitly. You can use the function notation syntax or CAST specification syntax to explicitly cast a data type. DB2 might implicitly cast data types during assignments that involve a distinct type (see “Distinct type assignments” on page 71). In addition, when you

create a sourced user-defined function, the data types of the parameters of the source function must be castable to the data types of the function that you are creating (see “CREATE FUNCTION” on page 529).

If truncation occurs when a character or graphic string is cast to another data type, a warning occurs if any non-blank characters are truncated. This truncation behavior is unlike the assignment of character or graphic strings to a target when an error occurs if any non-blank characters are truncated.

For casts that involve a distinct type as either the data type to be cast to or from, Table 7 shows the supported casts. For casts between built-in data types, Table 8 on page 64 shows the supported casts.

Table 7. Supported casts when a distinct type is involved

Data type ...	Is castable to data type ...
Distinct type <i>DT</i>	Source data type of distinct type <i>DT</i>
Source data type of distinct type <i>DT</i>	Distinct type <i>DT</i>
Distinct type <i>DT</i>	Distinct type <i>DT</i>
Data type <i>A</i>	Distinct type <i>DT</i> where <i>A</i> is promotable to the source data type of distinct type <i>DT</i> (see “Promotion of data types” on page 61)
INTEGER	Distinct type <i>DT</i> if <i>DT</i> 's source data type is SMALLINT
DOUBLE	Distinct type <i>DT</i> if <i>DT</i> 's source data type is REAL
VARCHAR	Distinct type <i>DT</i> if <i>DT</i> 's source data type is CHAR
VARGRAPHIC	Distinct type <i>DT</i> if <i>DT</i> 's source data type is GRAPHIC

When a distinct type is involved in a cast, a cast function that was generated when the distinct type was created is used. How DB2 chooses the function depends on whether function notation or CAST specification syntax is used. (For details, see “Function resolution” on page 107 and “CAST specification” on page 126, respectively.) Function resolution is similar for both. However, in CAST specification, when an unqualified distinct type is specified as the target data type, DB2 first resolves the schema name of the distinct type and then uses that schema name to locate the cast function.

Casting between data types

Table 8. Supported casts between built-in data types

To data type ¹ →										V	A	R		T	IM	E	S	R
Cast from data type ↓	S	M	I	D		D	V	G	G									
	M	A	N	E	D	O	A	R	R	R	D							
	L	T	C	R	U	C	C	C	P	P	C	B	D	T	I	S	T	
	L	E	I	R	B	H	H	L	H	H	L	L	A	T	M	A	W	
	I	G	M	E	B	H	A	O	I	I	O	O	T	E	M	M	I	
	N	E	A	A	L	A	A	B	C	C	B	B	E	E	E	M	P	
	T	R	L	L	E	R	R	C	B	C	B	B					D	
	SMALLINT	Y	Y	Y	Y	Y	Y	-	-	-	-	-	-	-	-	-	-	
	INTEGER	Y	Y	Y	Y	Y	Y	Y	-	-	-	-	-	-	-	-	-	
	DECIMAL	Y	Y	Y	Y	Y	Y	Y	-	-	-	-	-	-	-	-	-	
REAL	Y	Y	Y	Y	Y	Y	Y	Y	-	-	-	-	-	-	-	-	-	
DOUBLE	Y	Y	Y	Y	Y	Y	Y	Y	-	-	-	-	-	-	-	-	-	
CHAR	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y ²	Y	Y	Y	Y	Y	
VARCHAR	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y ²	Y	Y	Y	Y	Y	
CLOB	-	-	-	-	-	Y	Y	Y	Y	Y	Y	Y ²	Y	-	-	-	-	
GRAPHIC	-	Y ²	Y ³	Y ³	Y ³	Y	Y	Y	Y	Y	Y ²	Y ²	-					
VARGRAPHIC	-	Y ²	Y ³	Y ³	Y ³	Y	Y	Y	Y	Y	Y ²	Y ²	-					
DBCLOB	-	-	-	-	-	Y ³	Y ³	Y ³	Y	Y	Y	Y	Y	-	-	-	-	
BLOB	-	-	-	-	-	-	-	-	-	-	-	-	Y	-	-	-	-	
DATE	-	-	-	-	-	Y	Y	-	-	-	-	-	Y	-	-	-	-	
TIME	-	-	-	-	-	Y	Y	-	-	-	-	-	Y	-	-	-	-	
TIMESTAMP	-	-	-	-	-	Y	Y	-	-	-	-	-	Y	Y	Y	-	-	
ROWID	-	-	-	-	-	Y	Y	-	-	-	-	-	Y	-	-	-	Y	

Note:

1. Other synonyms for the listed data types are considered to be the same as the synonym listed. Some exceptions exist when the cast involves character string data if the subtype is FOR BIT DATA.
2. These data types are castable between each other only if the data is Unicode.
3. These data types are castable between each other only if the data is Unicode. The result length for these casts is $3 * \text{LENGTH}(\text{graphic string})$.

Assignment and comparison

The basic operations of SQL are assignment and comparison. Assignment operations are performed during the execution of INSERT, UPDATE, FETCH, SELECT INTO, SET *assignment*, and VALUES INTO statements. In addition, when a function is invoked or a stored procedure is called, the arguments of the function or stored procedure are assigned. Comparison operations are performed during the execution of statements that include predicates and other language elements such as MAX, MIN, DISTINCT, GROUP BY, and ORDER BY.

The basic rule for both operations is that data types of the operands must be compatible. The compatibility rule also applies to other operations that are described under “Rules for result data types” on page 77. Table 9 on page 65 shows the compatibility matrix for data types.

Assignment and comparison

Table 9. Compatibility of data types for assignments and comparisons. Y indicates that the data types are compatible. N indicates no compatibility. For any number in a column, read the corresponding note at the bottom of the table.

Operands	Binary integer	Decimal number	Floating point	Character string	Graphic string	Binary string	Date	Time	Time-stamp	Row ID	Distinct type
Binary Integer	Y	Y	Y	N	N	N	N	N	N	N	2
Decimal Number	Y	Y	Y	N	N	N	N	N	N	N	2
Floating Point	Y	Y	Y	N	N	N	N	N	N	N	2
I Character String	N	N	N	Y	N ^{4,5}	N ³	1	1	1	N	2
I Graphic String	N	N	N	N ^{4,5}	Y	N	1,4	1,4	1,4	N	2
Binary String	N	N	N	N ³	N	Y	N	N	N	N	2
I Date	N	N	N	1	1,4	N	Y	N	N	N	2
I Time	N	N	N	1	1,4	N	N	Y	N	N	2
I Time-stamp	N	N	N	1	1,4	N	N	N	Y	N	2
Row ID	N	N	N	N	N	N	N	N	N	Y	2
Distinct Type	2	2	2	2	2	2	2	2	2	2	Y ²

Notes:

1. The compatibility of datetime values is limited to assignment and comparison:
 - Datetime values can be assigned to string columns and to string variables, as explained in “Datetime assignments” on page 70.
 - A valid string representation of a date can be assigned to a date column or compared to a date.
 - A valid string representation of a time can be assigned to a time column or compared to a time.
 - A valid string representation of a timestamp can be assigned to a timestamp column or compared to a timestamp.
2. A value with a distinct type is comparable only to a value that is defined with the same distinct type. In general, DB2 supports assignments between a distinct type value and its source data type. For additional information, see “Distinct type assignments” on page 71.
3. All character strings, even those with subtype FOR BIT DATA, are not compatible with binary strings.
4. These data types are compatible if the graphic string is Unicode UTF-16. On assignment and comparison from Graphic to Character, the resulting length is 3 * (LENGTH(graphic string)).
5. Character strings with subtype FOR BIT DATA are not compatible with Unicode UTF-16 Graphic Data.

Compatibility with a column that has a field procedure is determined by the data type of the column, which applies to the decoded form of its values.

A basic rule for assignment operations is that a null value cannot be assigned to a column that cannot contain null values, nor to a host variable that does not have an associated indicator variable. For a host variable that does have an associated indicator variable, a null value is assigned by setting the indicator variable to a negative value. See “References to host variables” on page 100 for a discussion of indicator variables.

Assignment and comparison

Numeric assignments

The basic rule for numeric assignments is that the whole part of a decimal or integer number cannot be truncated. If necessary, the fractional part of a decimal number is truncated.

Decimal or integer to floating-point

Because floating-point numbers are only approximations of real numbers, the result of assigning a decimal or integer number to a floating-point column or variable might not be identical to the original number.

Floating-point or decimal to integer

When a single precision floating-point number is converted to integer, rounding occurs on the seventh significant digit, zeros are added to the end of the number, if necessary, starting from the seventh significant digit, and the fractional part of the number is eliminated.

When a double precision floating-point or decimal number is converted to integer, the fractional part of the number is eliminated.

The following examples show a single precision floating-point number converted to an integer:

Example 1:

The floating-point number assigned to an integer column or host variable is:	2.0000045E6
	2000000

Example 2:

The floating-point number assigned to an integer column or host variable is:	2.00000555E8
	200001000

The following examples show a double precision floating-point number converted to an integer:

Example 1:

The floating-point number assigned to an integer column or host variable is:	2.0000045E6
	2000004

Example 2:

The floating-point number assigned to an integer column or host variable is:	2.00000555E8
	200000555

The following examples show a decimal number converted to an integer:

Example 1:

The decimal number assigned to an integer column or host variable is:	2000004.5
	2000004

Example 2:

The decimal number assigned to an integer column or host variable is:	200000555.0
	200000555

Decimal to decimal

When a decimal number is assigned to a decimal column or variable, the number is converted, if necessary, to the precision and the scale of the target. The necessary number of leading zeros is added or eliminated, and, in the fractional part of the number, the necessary number of trailing zeros is added, or the necessary number of trailing digits is eliminated.

Integer to decimal

When an integer is assigned to a decimal column or variable, the number is converted first to a temporary decimal number and then, if necessary, to the precision and scale of the target. The precision and scale of the temporary decimal number is 5,0 for a small integer or 11,0 for a large integer.

Floating-point to floating-point

When a single precision System/390 floating-point number is assigned to a double precision floating-point column or variable, the single precision data is padded with eight hex zeros.

When a double precision System/390 floating-point number is assigned to a single precision floating-point column or variable, the double precision data is converted and rounded up on the seventh hex digit.

In assembler, C, or C++ applications that are precompiled with the FLOAT(IEEE) option, floating-point constants and values in host variables are assumed to have IEEE floating-point format. All floating-point data is stored in DB2 in System/390 floating-point format. Therefore, when the FLOAT(IEEE) precompiler option is in effect, DB2 performs the following conversions:

- When a number in short or long IEEE floating-point format is assigned to a single-precision or double-precision floating-point column, DB2 converts the number to System/390 floating-point format.
- When a single-precision or double-precision floating-point column value is assigned to a short or long floating-point host variable, DB2 converts the column value to IEEE floating-point format.

Floating-point to decimal

When a single precision floating-point number is assigned to a decimal column or variable, the number is first converted to a temporary decimal number of precision 6 by rounding on the seventh decimal digit. Twenty five zeros are then appended to the number to bring the precision to 31. Because of rounding, a number less than 0.5×10^{-6} is reduced to 0.

When a double precision floating-point number is assigned to a decimal column or variable, the number is first converted to a temporary decimal number of precision 15, and then, if necessary, truncated to the precision and scale of the target. In this conversion, zeros are added to the end of the number, if necessary, to bring the precision to 16. The number is then rounded (using floating-point arithmetic) on the sixteenth decimal digit to produce a 15-digit number. Because of rounding, a number less in magnitude than 0.5×10^{-15} is reduced to 0. If the decimal number requires more than 15 digits to the left of the decimal point, an error is reported. Otherwise, the scale is given the largest possible value that allows the whole part of the number to be represented without loss of significance.

The following examples show the effect of converting a double precision floating-point number to decimal:

Example 1:

Assignment and comparison

The floating-point number	.123456789098765E-05
in decimal notation is:	.00000123456789098765 +5
Rounding adds 5 in the 16th position	.00000123456789148765
and truncates the result to	.000001234567891
Zeros are then added to the end of a 31-digit result:	.00000123456789100000000000000000

Example 2:

The floating-point number	1.2339999999999E+01
in decimal notation is:	12.33999999999900 +5
Rounding adds 5 in the 16th position	12.3399999999905
and truncates the result to	12.339999999990
Zeros are then added to the end of a 31-digit result:	12.3399999999900000000000000000000

To COBOL integers

Assignment to COBOL integer variables uses the full size of the integer. Thus, the value placed in the COBOL data item might be out of the range of values.

Example 1: If COL1 contains a value of 12345, the following statements cause the value 12345 to be placed in A, even though A has been defined with only 4 digits:

```
01 A PIC S9999 BINARY.  
EXEC SQL SELECT COL1  
      INTO :A  
      FROM TABLEX  
END-EXEC.
```

Example 2: The following COBOL statement results in 2345 being placed in A:

```
MOVE 12345 TO A.
```

String assignments

The following rules apply when both the source and the target are strings. When a datetime data type is involved, see “Datetime assignments” on page 70. For the special considerations that apply when a distinct type is involved in an assignment, especially to a host variable, see “Distinct type assignments” on page 71.

There are two types of string assignments:

- *Storage assignment* is when a value is assigned to a column or a parameter of a function or stored procedure.
- *Retrieval assignment* is when a value is assigned to a host variable.

The rules differ for storage and retrieval assignment.

Storage assignment

The basic rule is that the length of the string assigned to a column or parameter of a function or stored procedure must not be greater than the length attribute of the column or the parameter. Trailing blanks are included in the length of the string. When the length of the string is greater than the length attribute of the column or the parameter, the following actions occur:

- If all of the trailing characters that must be truncated to make a string fit the target are blanks and the string is a character or graphic string, the string is truncated and assigned without warning.
- Otherwise, the string is not assigned and an error occurs to indicate that at least one of the excess characters is non-blank.

When a string is assigned to a fixed-length column or parameter and the length of the string is less than the length attribute of the target, the string is padded to the right with the necessary number of SBCS or DBCS blanks. The pad character is always a blank even for columns or parameters defined with the FOR BIT DATA attribute.

Retrieval Assignment

The length of a string assigned to a host variable can be greater than the length attribute of the host variable. When the length of the string is greater than the length of the host variable, the string is truncated on the right by the necessary number of SBCS or DBCS characters. When this occurs, the value W is assigned to the SQLWARN1 field of the SQLCA. Furthermore, if an indicator variable is provided, it is set to the original length of the string.

When a character string is assigned to a fixed-length variable and the length of the string is less than the length attribute of the target, the string is padded to the right with the necessary number of blanks. The pad character is always a blank even for strings defined with the FOR BIT DATA attribute.

Assignments involving mixed data strings

A mixed data string that contains DBCS characters cannot be assigned to an SBCS column, SBCS parameter, or SBCS host variable. The following rules apply when a mixed data string is assigned to a host variable and the string is longer than the length attribute of the variable:

- If the string is not well-formed mixed data, it is truncated as if it were BIT or graphic data.
- If the string is well-formed mixed data, it is modified on the right such that it is well-formed mixed data with a length that is the same as the length attribute of the variable and the number of characters lost is minimal.

Assignments involving C NUL-terminated strings

A C NUL-terminated string variable referenced in a CONNECT statement need not contain a NUL (X'00'). Otherwise, DB2 enforces the convention that the value of a NUL-terminated string variable, either character or graphic, is NUL-terminated. An input host variable that does not contain a NUL will cause an error. A value assigned to an output variable will always be NUL-terminated even if a character must be truncated to make room for the NUL.

When a string of length n is assigned to a C NUL-terminated string variable with a length greater than $n+1$, the rules depend on whether the source string is a value of a fixed-length string or a varying-length string:

Assignment and comparison

- If the source is a fixed-length string, the string is padded on the right with $x-n-1$ blanks, where x is the length of the variable. The padded string is then assigned to the variable and a NUL is placed in the last byte of the variable.
- If the source is a varying-length string, the string is assigned to the first n bytes of the variable and a NUL is placed in the next byte.

Conversion rules for string assignment

A character or graphic string assigned to a column or host variable is first converted, if necessary, to the coded character set of the target. Conversion is necessary only if all the following are true:

- The CCSIDs of string and target are different.
- Neither CCSID is X'FFFF' (neither the string nor the target is defined as BIT data).
- The string is neither null nor empty.

An error occurs if:

- The SYSSTRINGS table is used but contains no information about the pair of CCSIDs and DB2 cannot do the conversion through OS/390 support for Unicode or Language Environment.
- A character of the string cannot be converted and the operation is assignment to a column or to a host variable that has no indicator variable.
- A mixed data string that contains DBCS characters is assigned to an SBCS column.

A warning occurs if:

- A character of the string is converted to a *substitution character*. A *substitution character* is the character that is used when a character of the source character set is not part of the target character set. For example, if the source character set includes Katakana characters and the target character set does not, a Katakana character is converted to the EBCDIC SUB X'3F'.
- A character of the string cannot be converted and the operation is assignment to a host variable that has an indicator variable. For example, a DBCS character cannot be converted if the host variable has an SBCS CCSID. In this case, the string is not assigned to the host variable and the indicator variable is set to -2.

Datetime assignments

A DATE, TIME, or TIMESTAMP value can only be assigned to a column with a matching data type (whether DATE, TIME, or TIMESTAMP), a string column, or a string variable. The string column or string variable can be fixed or varying-length, but must not be a BLOB, CLOB, or DBCLOB column or variable. A value assigned to a DATE, TIME, or TIMESTAMP column must have a matching data type (whether DATE, TIME, or TIMESTAMP) or must be a valid string representation of the matching data type. The string representation must not be a BLOB, CLOB, or DBCLOB. A datetime value cannot be assigned to a column that has a field procedure.

When a datetime value is assigned to a string variable or column, it is converted to its string representation. Leading zeros are not omitted from any part of the date, time, or timestamp. The required length of the target varies depending on the format of the string representation. If the length of the target is greater than required, it is padded on the right with blanks. If the length of the target is less than required, the result depends on the type of datetime value involved, and on the type of target.

- If the target is a string column (except for BLOB, CLOB, or DBCLOB), truncation is not allowed. The length of the column must be at least 10 for a date, 8 for a time, and 19 for a timestamp.
- When the target is a host variable, the following rules apply:
 - For a DATE:** The length of the variable must not be less than 10.
 - For a TIME:** If the USA format is used, the length of the variable must not be less than 8. This format does not include seconds.
 - If the ISO, EUR, or JIS format is used, the length of the variable must not be less than 5. If the length is 5, 6, or 7, the seconds part of the time is omitted from the result and SQLWARN1 is set to 'W'. In this case, the seconds part of the time is assigned to the indicator variable if one is provided, and, if the length is 6 or 7, the value is padded with blanks so that it is a valid string representation of a time.
 - For a TIMESTAMP:** The length of the variable must not be less than 19. If the length is between 19 and 25, the timestamp is truncated like a string, causing the omission of one or more digits of the microsecond part. If the length is 20, the trailing decimal point is replaced by a blank so that the value is a valid string representation of a timestamp.

Row ID assignments

A row ID value can only be assigned to a column, parameter, or host variable with a row ID data type. For the value of the ROWID column, the column must be defined as GENERATED BY DEFAULT and the column must have a unique, single-column index. The value that is specified for the column must be a valid row ID value that was previously generated by DB2.

Distinct type assignments

The rules that apply to the assignments of distinct types to host variables are different than the rules for all other assignments that involve distinct types.

Assignments to host variables: The assignment of distinct type to a host variable is based on the source data type of the distinct type. Therefore, the value of a distinct type is assignable to a host variable only if the source data type of the distinct type is assignable to the host variable.

Example: Assume that distinct type AGE was created with the following SQL statement:

```
CREATE DISTINCT TYPE AGE AS SMALLINT WITH COMPARISONS;
```

When the statement was executed, DB2 also generated these cast functions:

```
AGE (SMALLINT) RETURNS AGE
AGE (INTEGER) RETURNS AGE
SMALLINT (AGE) RETURNS SMALLINT
```

Next, assume that column STU_AGE was defined in table STUDENTS with distinct type AGE. Now, consider this valid assignment of a student's age to host variable HV_AGE, which has an INTEGER data type:

```
SELECT STU_AGE INTO :HV_AGE FROM STUDENTS WHERE STU_NUMBER = 200;
```

The distinct type value is assignable to host variable HV_AGE because the source data type of the distinct type (SMALLINT) is assignable to the host variable (INTEGER). If distinct type AGE had been sourced on a character data type such as CHAR(5), the above assignment would be invalid because a character type cannot be assigned to an integer type.

Assignment and comparison

Assignments other than to host variables: A distinct type can be the source or target of an assignment. Assignment is based on whether the data type of the value to be assigned is castable to the data type of the target. (Table 7 on page 63 shows which casts are supported when a distinct type is involved). Therefore, a distinct type value can be assigned to any target other than a host variable when:

- The target of the assignment has the same distinct type, or
- The distinct type is castable to the data type of the target

Any value can be assigned to a distinct type when:

- The value to be assigned has the same distinct type as the target, or
- The data type of the assigned value is castable to the target distinct type

Example: Assume that the source data type for distinct type AGE is SMALLINT:

```
CREATE DISTINCT TYPE AGE AS SMALLINT WITH COMPARISONS
```

Next, assume that two tables TABLE1 and TABLE2 were created with four identical column descriptions:

```
AGECOL    AGE  
SMINTCOL  SMALLINT  
INTCOL    INTEGER  
DECCOL    DEC(6,2)
```

Using the following SQL statement and substituting various values for X and Y to insert values into various columns of TABLE1 from TABLE2, Table 10 shows whether the assignments are valid. DB2 uses assignment rules in this INSERT statement to determine if X can be assigned to Y.

```
INSERT INTO TABLE1 (Y)  
SELECT X FROM TABLE2;
```

Table 10. Assessment of various assignments for example INSERT statement

X (column in TABLE2)	Y (column in TABLE1)	Valid	Reason
AGECOL	AGECOL	Yes	Source and target are same distinct type
SMINTCOL	AGECOL	Yes	SMALLINT can be cast to AGE
INTCOL	AGECOL	Yes	INTEGER can be cast to AGE (because AGE's source type is SMALLINT)
DECCOL	AGECOL	No	DECIMAL cannot be cast to AGE
AGECOL	SMINTCOL	Yes	AGE can be cast to its source type of SMALLINT
AGECOL	INTCOL	No	AGE cannot be cast to INTEGER
AGECOL	DECCOL	No	AGE cannot be cast to DECIMAL

Numeric comparisons

Numbers are compared algebraically, that is, with regard to sign. For example, -2 is less than +1.

If one number is an integer and the other is decimal, the comparison is made with a temporary copy of the integer, which has been converted to decimal.

When decimal numbers with different scales are compared, the comparison is made with a temporary copy of one of the numbers that has been extended with trailing zeros so that its fractional part has the same number of digits as the other number.

If one number is double precision floating-point and the other is integer, decimal, or single precision floating-point, the comparison is made with a temporary copy of the other number which has been converted to double precision floating-point. However, if a single precision floating-point number is compared with a floating-point constant, the comparison is made with a single precision form of the constant.

Two floating-point numbers are equal only if the bit configurations of their normalized forms are identical.

String comparisons

Two strings are compared by comparing the corresponding bytes of each string. If the strings do not have the same length, the comparison is made with a temporary copy of the shorter string that has been padded on the right with blanks so that it has the same length as the other string.

Two strings are equal if they are both empty or if all corresponding bytes are equal. An empty string is equal to a blank string. If two strings are not equal, their relationship (that is, which has the greater value) is determined by the comparison of the first pair of unequal bytes from the left end of the strings. This comparison is made according to the collating sequence associated with the encoding scheme of the data. For ASCII data, characters A through Z (both upper and lowercase) have a greater value than characters 0 through 9. For EBCDIC data, characters A through Z (both upper and lowercase) have a lesser value than characters 0 through 9.

Varying-length strings with different lengths are equal if they differ only in the number of trailing blanks. In operations that select one value from a collection of such values, the value selected is arbitrary. The operations that can involve such an arbitrary selection are DISTINCT, MAX, MIN, and references to a grouping column. See the description of GROUP BY for further information about the arbitrary selection involved in references to a grouping column.

String comparisons with field procedures

If a column with a field procedure is compared with the value of a variable or a constant, the variable or constant is encoded by the field procedure before the comparison is made. If the comparison operator is LIKE, the variable or constant is not encoded and the column value is decoded.

If a column with a field procedure is compared with another column, that column must have the same field procedure. The comparison is performed on the encoded form of the values in the columns. If the encoded values are numeric, their data types must be identical; if they are strings, their data types must be compatible.

If two encoded strings of different lengths are compared, the shorter is temporarily padded with encoded blanks so that it has the same length as the other string.

In a CASE expression, if a column with a field procedure is used as the *result-expression* in a THEN or ELSE clause, all other columns that are used as *result-expressions* must have the same field procedure. Otherwise, no column used in a *result-expression* may name a field procedure.

Conversion rules for string comparison

When two strings are compared, one of the strings is first converted, if necessary, to the coded character set of the other string. Conversion is necessary only if all of the following are true:

- The CCSIDs of the two strings are different.

Assignment and comparison

- Neither CCSID is X'FFFF' (neither string is defined as BIT data or is a BLOB string).
- The string selected for conversion is neither null nor empty.
- For Unicode data only, the following conversion tables (Table 11 or Table 12) indicate that conversion is necessary.

The conversion that occurs when SBCS data is compared with mixed data depends on the encoding scheme. For ASCII and EBCDIC data, the conversion depends on the value of the field MIXED DATA on installation panel DSNTIPF at the DB2 that does the comparison:

- If this value is YES, the SBCS operand is converted to MIXED.
- If this value is NO, the MIXED operand is converted to SBCS.

For comparison of two Unicode strings, the following table shows which operand is selected for conversion.

Table 11. Operand that supplies the CCSID for character conversion for the CCSID for Unicode data

First operand	Second operand		
	SBCS Data	Mixed UTF-8 Data	DBCS UTF-16 Data
SBCS Data	See next table.	second	second
Mixed UTF-8 Data	first	See next table.	second
DBCS UTF-16 Data	first	first	See next table.

In other cases, the string selected for conversion depends on the type of the operands. The following table shows which operand supplies the target CCSID, given the operand types.

Table 12. Operand that supplies the CCSID for character conversion

First operand	Second operand				
	Column value	String constant	Special register	Derived value	Host variable
Column Value	first	first	first	first	first
String Constant	second	first	first	first	first
Special Register	second	first	first	first	first
Derived Value	second	second	second	first	first
Host Variable	second	second	second	second	first/second ¹

Note: 1. Both operands are converted, if necessary, to the system CCSID of the server for the encoding scheme in effect.

For example, assume a comparison of the form:

```
string-constant = derived-value
```

Here, the relevant table entry is in the second row and fourth column. The value for this entry shows that the first operand (*string-constant*) supplies the target CCSID. Thus, the derived value is converted, if necessary, to the coded character set of the string constant.

An error occurs if a character of the string cannot be converted, the SYSSTRINGS table is used but contains no information about the pair of CCSIDs of the operands

| being compared, or DB2 cannot do the conversion through OS/390 support for
| Unicode or Language Environment. A warning occurs if a character of the string is
| converted to a substitution character.

Datetime comparisons

A DATE, TIME, or TIMESTAMP value can be compared either with another value of the same data type or with a string representation of that data type. All comparisons are chronological, which means the further a point in time is from January 1, 0001, the greater the value of that point in time.

Comparisons involving TIME values and string representations of time values always include seconds. If the string representation omits seconds, zero seconds are implied.

Comparisons involving TIMESTAMP values are chronological without regard to representations that might be considered equivalent. Thus, the following predicate is true:

```
TIMESTAMP('1990-02-23-00.00.00') > '1990-02-22-24.00.00'
```

Row ID comparisons

A value with a row ID type can only be compared to another row ID value. The comparison of the row ID values is based on their internal representations. The maximum number of bytes that are compared is 17 bytes, which is the number of bytes in the internal representation. Therefore, row ID values that differ in bytes beyond the 17th byte are considered to be equal.

Distinct type comparisons

A value with a distinct type can only be compared to another value with exactly the same type because distinct types have strong typing, which means that a distinct type is compatible only with its own type. Therefore, to compare a distinct type to a value with a different data type, the distinct type value must be cast to the data type of the comparison value or the comparison value must be cast to the distinct type. For example, because constants are built-in data types, a constant can be compared to a distinct type value only if it is first cast to the distinct type or vice versa.

Table 13 shows examples of valid and invalid comparisons, assuming the following SQL statements were used to define two distinct types AGE_TYPE and CAMP_DATE and table CAMP_ROSTER table.

```
CREATE DISTINCT TYPE AGE_TYPE AS INTEGER WITH COMPARISONS;
CREATE DISTINCT TYPE CAMP_DATE AS DATE WITH COMPARISONS;

CREATE TABLE CAMP_ROSTER
( NAME          VARCHAR(20),
  ATTENDEE_NUMBER  INTEGER NOT NULL,
  AGE            AGE_TYPE,
  FIRST_CAMP_DATE CAMP_DATE,
  LAST_CAMP_DATE  CAMP_DATE,
  BIRTHDATE      DATE);
```

Table 13. Examples of valid and invalid comparisons involving distinct types

SQL statement	Valid	Reason
Distinct types with distinct types		
SELECT * FROM CAMP_ROSTER WHERE FIRST_CAMP_DATE < LAST_CAMP_DATE;	Yes	Both values are the same distinct type.

Assignment and comparison

Table 13. Examples of valid and invalid comparisons involving distinct types (continued)

SQL statement	Valid	Reason
Distinct types with columns of the same source data type		
SELECT * FROM CAMP_ROSTER WHERE AGE > ATTENDEE_NUMBER;	No	A distinct type cannot be compared to integer.
SELECT * FROM CAMP_ROSTER WHERE INTEGER(AGE) > ATTENDEE_NUMBER;	Yes	The distinct type is cast to an integer, making the comparison of two integers.
SELECT * FROM CAMP_ROSTER WHERE CAST(AGE AS INTEGER) > ATTENDEE_NUMBER;		
SELECT * FROM CAMP_ROSTER WHERE AGE > AGE_TYPE(ATTENDEE_NUMBER);	Yes	Integer ATTENDEE_NUMBER is cast to the distinct type AGE_TYPE, making both values the same distinct type.
SELECT * FROM CAMP_ROSTER WHERE AGE > CAST(ATTENDEE_NUMBER as AGE_TYPE);		
Distinct types with constants		
SELECT * FROM CAMP_ROSTER WHERE AGE IN (15,16,17);	No	A distinct type cannot be compared to a constant.
SELECT * FROM CAMP_ROSTER WHERE INTEGER(AGE) IN (15,16,17);	Yes	The distinct type is cast to the data type of constants, making all the values in the comparison integers.
SELECT * FROM CAMP_ROSTER WHERE AGE IN (AGE_TYPE(15),AGE_TYPE(16),AGE_TYPE(17));	Yes	Constants are cast to distinct type AGE_TYPE, making all the values in the comparison the same distinct type.
SELECT * FROM CAMP_ROSTER WHERE FIRST_CAMP_DATE > '06/12/99';	No	A distinct type cannot be compared to a constant.
SELECT * FROM CAMP_ROSTER WHERE FIRST_CAMP_DATE > CAST('06/12/99' AS CAMP_DATE);	No	The string constant '06/12/99', a VARCHAR data type, cannot be cast directly to distinct type CAMP_DATE, which is sourced on a DATE data type. As illustrated in the next row, the constant must be cast to a DATE data type and then to the distinct type.
SELECT * FROM CAMP_ROSTER WHERE FIRST_CAMP_DATE > CAST(DATE('06/12/1999') AS CAMP_DATE);	Yes	The string constant '06/12/99' is cast to the distinct type CAMP_DATE, making both values the same distinct type. To cast a string constant to a distinct type that is sourced on a DATE, TIME, or TIMESTAMP data type, the string constant must first be cast to a DATE, TIME, or TIMESTAMP data type.
Distinct types with host variables		
SELECT * FROM CAMP_ROSTER WHERE AGE BETWEEN :HV_INTEGER AND :HV_INTEGER2;	No	The host variables have integer data types. A distinct type cannot be compared to an integer.
SELECT * FROM CAMP_ROSTER WHERE AGE BETWEEN CAST(:HV_INTEGER AS AGE_TYPE) AND AGE_TYPE(:HV_INTEGER2);	Yes	The host variables are cast to distinct type AGE_TYPE, making all the values the same distinct type.
SELECT * FROM CAMP_ROSTER WHERE FIRST_CAMP_DATE > :HV_VARCHAR;	No	The host variable has a VARCHAR data type. A distinct type cannot be compared to a VARCHAR.

Table 13. Examples of valid and invalid comparisons involving distinct types (continued)

SQL statement	Valid	Reason
SELECT * FROM CAMP_ROSTER WHERE FIRST_CAMP_DATE > CAST(DATE(:HV_VARCHAR) AS CAMP_DATE);	Yes	The host variable is cast to the distinct type CAMP_DATE, making both values the same distinct type. To cast a VARCHAR host variable to a distinct type that is sourced on a DATE, TIME, or TIMESTAMP data type, the host variable must first be cast to a DATE, TIME, or TIMESTAMP data type.

Rules for result data types

Rules that are applied to the operands of an operation determine the data type of the result. This section explains when those rules apply and lists them by the possible data types of operands.

The rules apply to:

- Corresponding columns in UNION or UNION ALL operations
- Result expressions of a CASE expression
- Arguments of the scalar functions COALESCE, IFNULL, and VALUE
- Expression values of the IN list of an IN predicate

The rules are applied subject to other restrictions on long strings for the various operations.

Evaluation of the operands of an operation determines the data type of the result. If an operation has more than one pair of operands, DB2 determines the result type of the first pair, uses this result type with the next operand to determine the next result type, and so on. The last intermediate result type and the last operand determine the result type of the operation.

String operands

Character and graphic strings are compatible with other character and graphic strings as long as there is a conversion between their corresponding CCSIDs.

Table 14. Result data types with string operands

One operand	Other operand	Data type of the result
CHAR(x)	CHAR(y)	CHAR(z) where z = max(x,y)
CHAR(x)	GRAPHIC(y)	GRAPHIC(z) where z = max(x,y)
VARCHAR(x)	CHAR(y) or VARCHAR(y)	VARCHAR(z) where z = max(x,y)
VARCHAR(x)	GRAPHIC(y) or VARCHAR(y)	VARGRAPHIC(z) where z = max(x,y)
CLOB(x)	CHAR(y), VARCHAR(y), or CLOB(y)	CLOB(z) where z = max(x,y)
CLOB(x)	VARGRAPHIC(y), GRAPHIC(y), or DBCLOB(y)	DBCLOB(z) where z = max(x,y)
VARGRAPHIC(x)	VARGRAPHIC(y), GRAPHIC(y), VARCHAR(y), or CHAR(y)	VARGRAPHIC(z) where z = max(x,y)

Rules for result data types

Table 14. Result data types with string operands (continued)

One operand	Other operand	Data type of the result
DBCLOB(x)	DBCLOB(y), VARGRAPHIC(y), GRAPHIC(y), CLOB(y), VARCHAR(y), or CHAR(y)	DBCLOB(z) where z = max(x,y)

Character string subtypes are determined as indicated in the following table:

Table 15. Result data types with character string operands

One operand	Other operand	Data type of the result
Bit data	Mixed, SBCS, or bit data	Bit data
Mixed data	Mixed or SBCS data	Mixed data
SBCS data	SBCS data	SBCS data

Binary string operands

Binary strings (BLOBs) are compatible only with other binary strings (BLOB)s. The data type of the result is a BLOB. Other data types can be treated as a BLOB data type by using the BLOB scalar function to cast the data type to a BLOB. The length of the result BLOB is the largest length of all the data types.

Table 16. Result data types with binary string operands

One operand	Other operand	Data type of the result
BLOB(x)	BLOB(y)	BLOB(z) where z = max(x,y)

Numeric operands

Numeric types are compatible only with other numeric types.

Table 17. Result data types with numeric operands

One operand	Other operand	Data type of the result
SMALLINT	SMALLINT	SMALLINT
INTEGER	INTEGER	INTEGER
INTEGER	SMALLINT	INTEGER
DECIMAL(w,x)	SMALLINT	DECIMAL(p,x) where $p = x + \max(w-x, 5)^1$
DECIMAL(w,x)	INTEGER	DECIMAL(p,x) where $p = x + \max(w-x, 11)^1$
DECIMAL(w,x)	DECIMAL(y,z)	DECIMAL(p,s) where $p = \max(x,z) + \max(w-x, y-z)^1$ $s = \max(x,z)$
REAL	REAL	REAL
REAL	DECIMAL, INTEGER, or SMALLINT	DOUBLE
DOUBLE	any numeric	DOUBLE

Note:

- Precision cannot exceed 31.

Datetime operands

A DATE type is compatible with another DATE type or any string expression that contains a valid string representation of a date. A string representation must not be a BLOB, CLOB, or DBCLOB and must have an actual length that is not greater than 255 bytes. The data type of the result is DATE.

A TIME type is compatible with another TIME type or any string expression that contains a valid string representation of a time. A string representation must not be a BLOB, CLOB, or DBCLOB and must have an actual length that is not greater than 255 bytes. The data type of the result is TIME.

A TIMESTAMP type is compatible with another TIMESTAMP type or any string expression that contains a valid string representation of a timestamp. A string representation must not be a BLOB, CLOB, or DBCLOB and must have an actual length that is not greater than 255 bytes. The data type of the result is TIMESTAMP.

Row ID operands

A row ID data type is compatible only with itself. The result has a row ID data type.

Distinct type operands

A distinct type is compatible only with itself. The data type of the result is the distinct type.

Nullable attribute of a result

With the exception of the COALESCE and VALUE functions, the result of an operation can be null unless the operands do not allow nulls.

Constants

A *constant* (also called a *literal*) specifies a value. Constants are classified as string constants or numeric constants. Numeric constants are further classified as integer, floating-point, or decimal. String constants are classified as character or graphic.

All constants have the attribute NOT NULL. A negative sign in a numeric constant with a value of zero is ignored.

Constants have a built-in data type. Therefore, an operation that involves a constant and a distinct type requires that the distinct type be cast to the built-in data type of the constant or the constant be cast to the distinct type. For example, see Table 13 on page 75, which contains an example of casting data types to compare a constant to a distinct type.

Integer constants

An *integer constant* specifies a binary integer as a signed or unsigned number that has a maximum of 10 significant digits and no decimal point. If the value is not within the range of a large integer, the constant is interpreted as a decimal constant. The data type of an integer constant is large integer.

Examples:

64 -15 +100 32767 720176

In syntax diagrams, the term *integer* is used for an integer constant that must not include a sign.

Constants

Floating-point constants

A *floating-point constant* specifies a floating-point number as two numbers separated by an E. The first number can include a sign and a decimal point. The second number can include a sign but not a decimal point. The value of the constant is the product of the first number and the power of 10 specified by the second number. It must be within the range of floating-point numbers. The number of characters in the constant must not exceed 30. Excluding leading zeros, the number of digits in the first number must not exceed 17 and the number of digits in the second must not exceed 2. The data type of a floating-point constant is double precision floating-point.

Examples: The following floating-point constants represent the numbers 150, 200000, -0.22, and 500:

15E1 2.E5 -2.2E-1 +5.E+2

Decimal constants

A *decimal constant* specifies a decimal number as a signed or unsigned number of no more than 31 digits and either includes a decimal point or is not within the range of binary integers. The precision is the total number of digits, including those, if any, to the right of the decimal point. The total includes all leading and trailing zeros. The scale is the number of digits to the right of the decimal point, including trailing zeros.

Examples: The following decimal constants have, respectively, precisions and scales of 5 and 2; 4 and 0; 2 and 0; 23 and 2:

025.50 1000. -15. +37589333333333333333.33

Character string constants

A *character string constant* specifies a varying-length character string. There are two forms of character string constant:

- A sequence of characters that starts and ends with a string delimiter, which is either an apostrophe ('') or a quotation mark (""). For the factors that determine which is applicable, see “Apostrophes and quotation marks in string delimiters” on page 149. This form of string constant specifies the character string contained between the string delimiters. The number of bytes between the delimiters must not be greater than 255. Two consecutive string delimiters are used to represent one string delimiter within the character string.
- An X followed by a sequence of characters that starts and ends with a string delimiter. This form of a character string constant is also called a *hexadecimal constant*. The characters between the string delimiters must be an even number of hexadecimal digits. The number of hexadecimal digits must not exceed 254. A hexadecimal digit is a digit or any of the letters A through F (uppercase or lowercase). Under the conventions of hexadecimal notation, each pair of hexadecimal digits represents a character. A hexadecimal constant allows you to specify characters that do not have a keyboard representation.

Examples:

'12/14/1985' '32' 'DON''T CHANGE' X'FFFF' ''

The rightmost string in the example ("") represents an empty character string constant, which is a string of zero length.

A character string constant is classified as mixed data if it includes a DBCS substring. In all other cases, a character string constant is classified as SBCS data. The CCSID assigned to the constant is the appropriate system CCSID of the database server. A mixed string constant can be continued from one line to the next only if the break occurs between single byte characters. A Unicode string is always considered mixed regardless of the content of the string.

For Unicode, character constants can be assigned to UTF-8 and UTF-16. The form of the constant does not matter. Typically, character string constants are used only with character strings, but they also can be used with graphic UTF-16 data. However, hexadecimal constants are just character data. Thus, hexadecimal constants being used to insert data into UTF-16 data strings should be in UTF-8 format, not UTF-16 format. For example, if you wanted to insert the number 1 into a UTF-16 column, you would use X'31', not X'0031'. Even though X'0031' is a UTF-16 value, DB2 treats it as two separate UTF-8 codepoints. Thus, X'0031' would become X'00000031'.

Datetime constants

A *datetime constant* is a character string constant of a particular format. Character string constants are described under the previous heading, “Character string constants” on page 80. For information about the valid string formats, see “String representations of datetime values” on page 57.

Graphic string constants

A *graphic string constant* specifies a varying-length graphic string. (Shift-in and shift-out characters for EBCDIC data are discussed in “Character strings” on page 49.)

In EBCDIC environments, the forms of graphic string constants are¹¹:

11. The PL/I form of graphic string constants is supported only in static SQL statements.

Constants

Context	Graphic String Constant	Empty String	Example
PL/I	$\$_0'`dbcs-string`G^{s_1}$ $\$_0'`dbcs-string`{s_1} G$	$\$_0``G^{s_1}$ $\$_0``{s_1} G$	$\$_0'元気`G^{s_1}$
	<p>G represents a DBCS G (X'42C7')</p> <p>$'$ represents a DBCS apostrophe (X'427D')</p>		
All other contexts	G'_0dbcs-string^{s_1}'$	G'_0{s_1}'$ G''	G'_0元気{s_1}'$
		g'_0{s_1}'$ g''	
	N'_0dbcs-string^{s_1}'$	N'_0{s_1}'$ N''	
		n'_0{s_1}'$ n''	

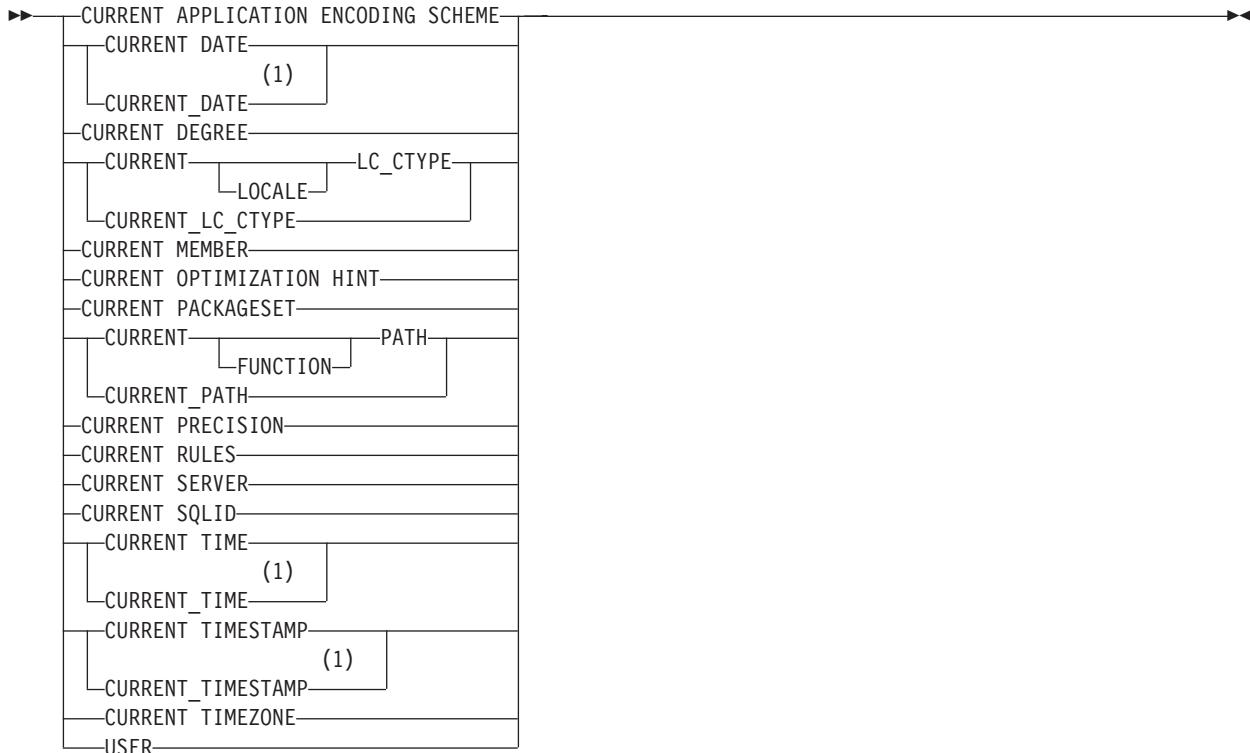
In SQL statements and in host language statements in a source program, graphic string constants cannot be continued from one line to the next. The maximum number of DBCS characters in a graphic string constant is 124.

| For Unicode, graphic constants can be assigned to UTF-8 and UTF-16. The form of the constant does not matter. Typically, graphic string constants are used only with graphic strings; however, they also can be used with character UTF-8 data.

Special registers

A special register is a storage area defined for a process by DB2. Wherever its name appears in an SQL statement, the name is replaced by the register's value when the statement is executed. Thus, the name acts like a function that has no arguments. The form of a special register is as follows:

special registers



Notes:

- 1 The SQL standard uses the form with the underline.

General rules for special registers

Following these general rules for special registers, each special register is described individually.

Changing register values: A commit operation might cause special registers to be re-initialized. Whether a special register is affected by a commit depends on whether the special register has been explicitly set within the application process. For example, assume that the PATH special register has not been explicitly set with a SET PATH statement in the application process. After a commit, the value of PATH is re-initialized. For information on the initialization of PATH, which can take the current value of CURRENT_SQLID into consideration, see “CURRENT_SQLID” on page 91.

A rollback operation has no effect on the values of special registers. Nor does any SQL statement, with the following exceptions:

- SQL SET statements can change the values of CURRENT APPLICATION ENCODING SCHEME, CURRENT DEGREE, CURRENT LOCALE LC_CTYPE,

Special registers

CURRENT OPTIMIZATION HINT, CURRENT PACKAGESET, CURRENT PATH, CURRENT PRECISION, CURRENT RULES, and CURRENT SQLID¹².

- SQL CONNECT statements can change the value of CURRENT SERVER.

CCSIDs for register values: The values of certain special registers are character strings. The registers with string values are:

- CURRENT APPLICATION ENCODING SCHEME
- CURRENT DEGREE
- CURRENT LOCALE LC_CTYPE
- CURRENT MEMBER
- CURRENT OPTIMIZATION HINT
- CURRENT PACKAGESET
- CURRENT PATH
- CURRENT PRECISION
- CURRENT RULES
- CURRENT SERVER
- CURRENT SQLID
- USER

The CCSID that is associated with these registers is the one named in the ASCII CODED CHAR SET, EBCDIC CODED CHAR SET, or the UNICODE CCSID field on installation panel DSNTIPF at the server executing the statement. The CCSID that is used depends on whether the SQL statement in which the special register is referenced involves data in ASCII, EBCDIC, or Unicode tables; if no table is involved, the CCSID for the default encoding scheme for your system is used. Field DEF ENCODING SCHEME on installation panel specifies whether the default encoding scheme is EBCDIC, ASCII, or UNICODE. For the SET *host-variable* and VALUES INTO statements, the application encoding scheme bind option determines the desired encoding scheme.

Datetime special registers: The datetime registers are named CURRENT DATE, CURRENT TIME, and CURRENT TIMESTAMP. Datetime special registers are stored in an internal format. When two or more of these registers are implicitly or explicitly specified in a single SQL statement, they represent the same point in time. A datetime special register is implicitly specified when it is used to provide the default value of a datetime column.

If the SQL statement in which a datetime special register is used is in a user-defined function or stored procedure that is within the scope of a trigger, DB2 uses the timestamp for the triggering SQL statement to determine the special register value.

The values of these special registers are based on:

- The time-of-day clock of the processor for the server executing the SQL statement
- The MVS TIMEZONE parameter for this processor. The TIMEZONE parameter is in SYS1.PARMLIB(CLOCKXX).

To evaluate the references when the statement is being executed, a single reading from the time-of-day clock is incremented by the number of hours, minutes, and seconds specified by the TIMEZONE parameter. The values derived from this are

12. If the SET CURRENT SQLID statement is executed in a stored procedure or user-defined function package that has a dynamic SQL behavior other than run behavior, the SET CURRENT SQLID statement does not affect the authorization ID that is used for dynamic SQL statements in the package. The dynamic SQL behavior determines the authorization ID. For more information, see the discussion of DYNAMICRULES in Chapter 2 of *DB2 Command Reference*.

assumed to be the local date, time, or timestamp, where local means local to the DB2 that executes the statement. This assumption is correct if the clock is set to local time and the MVS TIMEZONE parameter is zero or the clock is set to GMT and the MVS TIMEZONE parameter gives the difference from GMT. Universal time, coordinated (UTC) is another name for Greenwich Mean Time (GMT).

Since the datetime special registers and the CURRENT TIMEZONE special register depend on the MVS parameter PARMTZ(SYS1.PARMLIB(CLOCKXX)), their values are affected if the MVS local time at the server is changed by the MVS system command SET CLOCK. The values of the CURRENT DATE and CURRENT TIMESTAMP special registers might be affected if the MVS local date at the server is changed by the MVS system command SET DATE¹³.

Where special registers are processed: In distributed applications, CURRENT APPLICATION ENCODING SCHEME, CURRENT SERVER, and CURRENT PACKAGESET are processed locally. All other special registers are processed at the server.

CURRENT APPLICATION ENCODING SCHEME

CURRENT APPLICATION ENCODING SCHEME specifies which encoding scheme is to be used for dynamic statements. It allows an application to indicate the encoding scheme that is used to process data. This register is not supported in REXX applications or in stored procedures written in REXX.

The initial value of CURRENT APPLICATION ENCODING SCHEME is determined by the value of the ENCODING bind option if the bind option is specified. If the bind option was not specified, then the initial value is the value of field DEFAULT APPLICATION ENCODING SCHEME on installation panel DSNTIPF. You can change the value of the register by executing the statement SET CURRENT APPLICATION ENCODING SCHEME. For a description of the statement, see “SET CURRENT APPLICATION ENCODING SCHEME” on page 906.

The value contained in the special register is a character representation of a CCSID. Although you can use the values ASCII, EBCDIC, or UNICODE to set the special register, what is stored in the special register is a character representation of the numeric CCSID that corresponds to the value used in the SET CURRENT APPLICATION ENCODING SCHEME statement. The value ASCII, EBCDIC, or UNICODE is not stored. The CCSID_ENCODING scalar function can be used to get a value of ASCII, EBCDIC, or UNICODE from a numeric CCSID value.

The data type is CHAR(8). If necessary, the value is padded on the right with blanks so that its length is 8 bytes.

For stored procedures and user-defined functions, the initial value of the CURRENT APPLICATION ENCODING SCHEME special register is determined by the value of the ENCODING bind option for the package that is associated with the procedure or function. If the bind option was not specified, then the initial value is the value of the field DEFAULT APPLICATION ENCODING SCHEME field on installation panel DSNTIPF.

13. Whether the SET DATE command affects these special registers depends on the MVS system level and the program temporary fix (PTF) level of the system.

Special registers

For triggers, the initial value of the CURRENT APPLICATION ENCODING SCHEME special register is the value of field DEFAULT APPLICATION ENCODING SCHEME on installation panel DSNTIPF.

Example: The CURRENT APPLICATION ENCODING SCHEME special register can be used like any other special register:

```
EXEC SQL VALUES(CURRENT APPLICATION ENCODING SCHEME) INTO :HV1;
EXEC SQL INSERT INTO T1 VALUES (CURRENT APPLICATION ENCODING SCHEME);
EXEC SQL SET :HV1 = CURRENT APPLICATION ENCODING SCHEME;
EXEC SQL SELECT C1 FROM T1 WHERE C1 = CURRENT APPLICATION ENCODING SCHEME;
```

CURRENT DATE

CURRENT DATE, or equivalently CURRENT_DATE, specifies the current date. The data type is DATE. The value of CURRENT DATE in a user-defined function or stored procedure is inherited according to the rules in Table 19 on page 93. For other applications, the date is derived by the DB2 that executes the SQL statement that refers to the special register. For a description of how the date is derived, see “Datetime special registers” on page 84.

Example: Display the average age of employees.

```
SELECT AVG(YEAR(CURRENT DATE - BIRTHDATE))
   FROM DSN8710.EMP;
```

CURRENT DEGREE

CURRENT DEGREE specifies the degree of parallelism for the execution of queries that are dynamically prepared by the application process. The data type of the register is CHAR(3) and the only valid values are 1 (padded on the right with two blanks) and ANY.

If the value of CURRENT DEGREE is 1 when a query is dynamically prepared, the execution of that query will not use parallelism. If the value of CURRENT DEGREE is ANY when a query is dynamically prepared, the execution of that query can involve parallelism. See Part 5 (Volume 2) of *DB2 Administration Guide* for a description of query parallelism.

The initial value of CURRENT DEGREE is determined by the value of field CURRENT DEGREE on installation panel DSNTIP4. The default for the initial value of that field is 1 unless your installation has changed it to be ANY by modifying the value in that field. The initial value of CURRENT DEGREE in a user-defined function or stored procedure is inherited according to the rules in Table 19 on page 93.

You can change the value of the register by executing the statement SET CURRENT DEGREE. For details about this statement, see “SET CURRENT DEGREE” on page 907.

CURRENT DEGREE is a register at the database server. Its value applies to queries that are dynamically prepared at that server and to queries that are dynamically prepared at another DB2 subsystem as a result of the use of a DB2 private connection between that server and that DB2 subsystem.

Example: The following statement inhibits parallelism:

```
SET CURRENT DEGREE = '1';
```

CURRENT LOCALE LC_CTYPE

CURRENT LOCALE LC_CTYPE specifies the LC_CTYPE locale that will be used to execute SQL statements that use a built-in function that references a locale. Functions LCASE, UCASE, and TRANSLATE (with a single argument) refer to the locale when they are executed. The data type is CHAR(50). If necessary, the value is padded on the right with blanks so that its length is 50 bytes.

The initial value of CURRENT LOCALE LC_CTYPE is determined by the value of field LOCALE LC_CTYPE on installation panel DSNTIPF. The default for the initial value of that field is blank unless your installation has changed the value of that field. The initial value of CURRENT LOCALE LC_CTYPE in a user-defined function or stored procedure is inherited according to the rules in Table 19 on page 93.

You can change the value of the register by executing the statement SET CURRENT LOCALE LC_CTYPE. For details about this statement, see “SET CURRENT LOCALE LC_CTYPE” on page 909.

Example: Save the value of current register CURRENT LOCALE LC_CTYPE in host variable HV1, which is defined as VARCHAR(50).

```
EXEC SQL VALUES(CURRENT LOCALE LC_CTYPE) INTO :HV1;
```

CURRENT MEMBER

CURRENT MEMBER specifies the member name of a current DB2 data sharing member on which a statement is executing. The value of CURRENT MEMBER is a character string. The data type is CHAR(8). If necessary, the member name is padded on the right with blanks so that its length is 8 bytes.

The value of CURRENT MEMBER is a string of blanks when the application process is connected to a DB2 subsystem that is not a member of a data sharing group.

THE SQL SET statement cannot change the value of CURRENT MEMBER.

Example: Set the host variable MEM to the name of the current DB2 member.

```
EXEC CURRENT MEMBER :MEM = CURRENT MEMBER;
or
EXEC VALUES (CURRENT MEMBER) INTO :MEM;
```

CURRENT OPTIMIZATION HINT

CURRENT OPTIMIZATION HINT specifies the user-defined optimization hint that DB2 should use to generate the access path for dynamic statements. The data type is CHAR(8).

The value of the register identifies the rows in auth.PLAN_TABLE that DB2 uses to generate the access path. DB2 uses information in the rows in auth.PLAN_TABLE for which the value of the OPTHINT column matches the value of the CURRENT OPTIMIZATION special register. If the value of the register is all blanks, DB2 uses normal optimization and ignores optimization hints. If the value of the register includes any non-blank characters and DB2 was installed without optimization hints enabled (field OPTIMIZATION HINTS on installation panel DSNTIP4), a warning occurs.

The initial value of CURRENT OPTIMIZATION HINT is the value of the OPTHINT bind option. The initial value of CURRENT OPTIMIZATION HINT in a user-defined

Special registers

function or stored procedure is inherited according to the rules in Table 19 on page 93. You can change the value of the special register by executing the statement SET CURRENT OPTIMIZATION HINT. For details about this statement, see “SET CURRENT OPTIMIZATION HINT” on page 911.

Example: Set the CURRENT OPTIMIZATION HINT special register so that DB2 uses the optimization plan hint that is identified by host variable NOHYB when generating the access path for dynamic statements.

```
SET CURRENT OPTIMIZATION HINT = :NOHYB
```

For more information about telling DB2 how to generate access paths, see Part 6 of *DB2 Application Programming and SQL Guide*.

CURRENT PACKAGESET

CURRENT PACKAGESET specifies a string of blanks or the collection ID of the package or packages that will be used to execute SQL statements. The data type is CHAR(18). If necessary, the collection ID is padded on the right with blanks so that its length is 18 bytes.

The initial value of CURRENT PACKAGESET is blanks. The value is a collection ID only if the application process has explicitly specified a collection ID by means of the SET CURRENT PACKAGESET statement. For details about this statement, see “SET CURRENT PACKAGESET” on page 912. The initial value of CURRENT PACKAGESET in a user-defined function or stored procedure is inherited according to the rules in Table 19 on page 93.

Example: Before passing control to another program, identify the collection ID for its package as ALPHA.

```
EXEC SQL SET CURRENT PACKAGESET = 'ALPHA';
```

CURRENT PATH

CURRENT PATH, or equivalently CURRENT_PATH, specifies the SQL path used to resolve unqualified data type names and function names in dynamically prepared SQL statements. It is also used to resolve unqualified procedure names that are specified as host variables in SQL CALL statements (CALL *host-variable*). The data type is VARCHAR(254).

The CURRENT PATH special register contains a list of one or more schema names, where each schema name is enclosed in delimiters and separated from the following schema by a comma (any delimiters within the string are repeated as they are in any delimited identifier). The delimiters and commas are included in the 254 character length.

For information on when the SQL path is used to resolve unqualified names in both dynamic and static SQL statements and the effect of its value, see “Schemas and the SQL path” on page 40.

The initial value of the CURRENT PATH special register is:

- The value of the PATH bind option, or
- “SYSIBM”, “SYSPROC”, “*value of CURRENT SQLID special register*” if the PATH bind option was not specified

If the value of the CURRENT SQLID special register changes after the initial value of the CURRENT PATH special register is established, the value of the CURRENT PATH special register is unaffected when CURRENT SQLID is

```
# updated. However, if a commit occurs later and a SET PATH statement has not
# been processed, the value of the CURRENT PATH special register is
# re-initialized with the current value of the CURRENT SQLID special register
# taken into consideration.
```

| The initial value of CURRENT PATH in a user-defined function or stored procedure
| is inherited according to the rules in Table 19 on page 93.

| You can change the value of the register by executing the statement SET PATH.
| For details about this statement, see "SET PATH" on page 921.

Example: Set the special register so that schema SMITH is searched before
schemas SYSIBM, SYSFUN, and SYSPROC.

```
SET PATH = SMITH, SYSIBM, SYSFUN, SYSPROC;
```

CURRENT PRECISION

```
# CURRENT PRECISION specifies the rules to be used when both operands in a
# decimal operation have precisions of 15 or less. The data type of the register is
# CHAR(5). Valid values for the CURRENT PRECISION special register include
# 'DEC15,' 'DEC31,' and 'Dpp.s' where 'pp' is either 15 or 31 and 's' is a number
# between 1 and 9. DEC15 specifies the rules that do not allow a precision greater
# than 15 digits, and DEC31 specifies the rules that allow a precision of up to 31
# digits. The rules for DEC31 are always used if either operand has a precision
# greater than 15. If the form 'Dpp.s' is used, 'pp' represents the precision that will be
# used as the rules where DEC15 and DEC31 rules are used, and 's' represents the
# minimum divide scale to use for division operations. The separator used in the form
# 'Dpp.s' may be either the '.' or ',' character, regardless of the setting of the default
# decimal point.
```

| The initial value of CURRENT PRECISION is determined by the value of field
| DECIMAL ARITHMETIC on installation panel DSNTIP4. The default for the initial
| value is DEC15 unless your installation has changed it to be DEC31 by modifying
| the value in that field. The initial value of CURRENT PRECISION in a user-defined
| function or stored procedure is inherited according to the rules in Table 19 on page
| 93.

| You can change the value of the register by executing the statement SET
| CURRENT PRECISION. For details about this statement, see "SET CURRENT
| PRECISION" on page 914.

```
# CURRENT PRECISION only affects dynamic SQL. When an SQL statement is
# dynamically prepared and the value of CURRENT PRECISION is DEC15 or D15.s,
# where s is a number between 1 and 9, DEC15 rules will apply. When an SQL
# statement is dynamically prepared and the value of CURRENT PRECISION is
# DEC31 or D31.s, where s is a number between 1 and 9, DEC31 rules will apply.
# Preparation of a statement with DEC31 instead of DEC15 is more likely to result in
# an error, especially for division operations. Specifications of CURRENT PRECISION
# in the form 'Dpp.s' where 'pp' represents the precision, 15 or 31, and 's' represents
# the minimum divide scale will in some cases make division errors less likely when
# 'pp' is set to 31. For more information, see "Arithmetic with two decimal operands"
# on page 114.
```

Example: Set CURRENT PRECISION so that subsequent statements that are
prepared use DEC31 rules for decimal arithmetic:

```
SET CURRENT PRECISION = 'DEC31';
```

Special registers

```
# Example 2: Set CURRENT PRECISION so that subsequent statements that are
# prepared use DEC31 rules for decimal arithmetic with a minimum divide scale of 3:
#
#     SET CURRENT PRECISION = 'DEC31,3';
```

CURRENT RULES

CURRENT RULES specifies whether certain SQL statements are executed in accordance with DB2 rules or the rules of the SQL standard. The data type of the register is CHAR(3), and the only valid values are 'DB2' and 'STD'.

CURRENT RULES is a register at the database server. If the server is not the local DB2, the initial value of the register is 'DB2'. Otherwise, the initial value is the same as the value of the SQLRULES bind option. The initial value of CURRENT RULES in a user-defined function or stored procedure is inherited according to the rules in Table 19 on page 93.

You can change the value of the register by executing the statement SET CURRENT RULES. For details about this statement, see "SET CURRENT RULES" on page 915.

CURRENT RULES affects the statements listed in Table 18. The table summarizes when the statements are affected and shows where to find detailed information. CURRENT RULES also affects whether DB2 issues an *existence error* (SQLCODE -204) or an *authorization error* (SQLCODE -551) when an object does not exist.

Table 18. Summary of statements affected by CURRENT RULES

Statement	What is affected	Details on page
ALTER TABLE	Enforcement of check constraints added.	447
	Default value of the delete rule for referential constraints.	
	Whether DB2 creates LOB table spaces, auxiliary tables, and indexes on auxiliary tables for added LOB columns.	
	Whether DB2 creates an index for an added ROWID column that is defined with GENERATED BY DEFAULT.	
CREATE TABLE	Default value of the delete rule for referential constraints.	653
	Whether DB2 creates LOB table spaces, auxiliary tables, and indexes on auxiliary tables for LOB columns.	
	Whether DB2 creates an index for a ROWID column that is defined with GENERATED BY DEFAULT.	
DELETE	Authorization requirements for searched DELETE.	742
GRANT	Granting privileges to yourself.	803
REVOKE	Revoking privileges from authorization IDs	865
UPDATE	Authorization requirements for searched UPDATE.	928

Example: Set CURRENT RULES so that a later ALTER TABLE statement is executed in accordance with the rules of the SQL standard:

```
SET CURRENT RULES = 'STD';
```

CURRENT SERVER

CURRENT SERVER specifies the location name of the current server. The data type is CHAR(16). If necessary, the location name is padded on the right with blanks so that its length is 16 bytes.

The initial value of CURRENT SERVER depends on the CURRENTSERVER bind option. If CURRENTSERVER X is specified on the bind subcommand, the initial value is X. If the option is not specified, the initial value is the location name of the local DB2. The initial value of CURRENT SERVER in a user-defined function or stored procedure is inherited according to the rules in Table 19 on page 93. The value of CURRENT SERVER is changed by the successful execution of a CONNECT statement.

The value of CURRENT SERVER is a string of blanks when:

- The application process is in the unconnected state, or
- The application process is connected to a local DB2 subsystem that does not have a location name.

Example: Set the host variable CS to the location name of the current server.

```
EXEC SQL SET :CS = CURRENT SERVER;
```

CURRENT SQLID

CURRENT SQLID specifies the SQL authorization ID of the process. The data type is CHAR(8). If necessary, the authorization ID is padded on the right with blanks so that its length is 8 bytes.

The initial value of CURRENT SQLID can be provided by the connection or sign-on exit routine. If not, the initial value is the primary authorization ID of the process.

The initial value of CURRENT SQLID in a user-defined function or stored procedure is inherited according to the rules in Table 19 on page 93.

CURRENT SQLID can only be referred to in an SQL statement that is executed by the current server.

Example: Set the SQL authorization ID to 'GROUP34' (one of the authorization IDs of the process).

```
SET CURRENT SQLID = 'GROUP34';
```

CURRENT TIME

CURRENT TIME, or equivalently CURRENT_TIME, specifies the current time. The data type is TIME.

The time is derived by the DB2 that executes the SQL statement that refers to the special register. For a description of how the time is derived, see "Datetime special registers" on page 84. The value of CURRENT TIME in a user-defined function or stored procedure is inherited according to the rules in Table 19 on page 93.

Example: Display information about all project activities and include the current date and time in each row of the result.

Special registers

```
SELECT DSN8710.PROJECT.* , CURRENT DATE, CURRENT TIME  
      FROM DSN8710.PROJECT;
```

CURRENT TIMESTAMP

CURRENT TIMESTAMP, or equivalently CURRENT_TIMESTAMP, specifies the current timestamp. The data type is TIMESTAMP.

The timestamp is derived by the DB2 that executes the SQL statement that refers to the special register. For a description of how the timestamp is derived, see “Datetime special registers” on page 84. The value of CURRENT TIMESTAMP in a user-defined function or stored procedure is inherited according to the rules in Table 19 on page 93.

Example: Display information about the full image copies that were taken in the last week.

```
SELECT * FROM SYSIBM.SYSCOPY  
      WHERE TIMESTAMP > CURRENT TIMESTAMP - 7 DAYS;
```

CURRENT TIMEZONE

CURRENT TIMEZONE specifies the MVS TIMEZONE parameter in the form of a time duration. The data type is DECIMAL(6,0).

The time duration is derived by the DB2 that executes the SQL statement that refers to the special register. The seconds part of the time duration is always zero. An error occurs if the hours portion of the MVS TIMEZONE parameter is not between -24 and 24. The value of CURRENT TIMEZONE in a user-defined function or stored procedure is inherited according to the rules in Table 19 on page 93.

Example: Display information from SYSCOPY, but with the TIMESTAMP converted to GMT. This example is based on the assumption that the installation sets the clock to GMT and the MVS TIMEZONE parameter to the difference from GMT.

```
SELECT DBNAME, TSNAME, DSNUM, ICTYPE, TIMESTAMP - CURRENT TIMEZONE  
      FROM SYSIBM.SYSCOPY;
```

USER

USER specifies the primary authorization ID of the process. The data type is CHAR(8). If necessary, the authorization ID is padded on the right with blanks so that its length is 8 bytes.

If USER is referred to in an SQL statement that is executed at a remote DB2 and the primary authorization ID has been translated to a different authorization ID, USER specifies the translated authorization ID. For an explanation of authorization ID translation, see Part 3 (Volume 1) of *DB2 Administration Guide*. The value of USER in a user-defined function or stored procedure is determined according to the rules in Table 19 on page 93.

Example: Display information about tables, views, and aliases that are owned by the primary authorization ID of the process.

```
SELECT * FROM SYSIBM.SYSTABLES WHERE CREATOR = USER;
```

Inheriting special registers in a user-defined function or a stored procedure

Table 19 shows information you need when you use special registers in a user-defined function or stored procedure.

Table 19. Characteristics of special registers in a user-defined function or a stored procedure

Special register	Initial value when INHERIT SPECIAL REGISTERS option is specified	Initial value when DEFAULT SPECIAL REGISTERS option is specified	Routine can use SET statement to modify?
CURRENT APPLICATION ENCODING SCHEME	The value of bind option ENCODING for the user-defined function or stored procedure package	The value of bind option ENCODING for the user-defined function or stored procedure package	Yes
CURRENT DATE	New value for each SQL statement in the user-defined function or stored procedure package ¹	New value for each SQL statement in the user-defined function or stored procedure package ¹	Not applicable ⁴
CURRENT DEGREE	Inherited from invoker ²	The value of field CURRENT DEGREE on installation panel DSNTIP4	Yes
CURRENT LOCALE LC_CTYPE	Inherited from invoker	The value of field CURRENT LC_CTYPE on installation panel DSNTIPF	Yes
CURRENT OPTIMIZATION HINT	The value of bind option OPTHINT for the user-defined function or stored procedure package or inherited from invoker ⁵	The value of bind option OPTHINT for the user-defined function or stored procedure package	Yes
CURRENT PACKAGESET	Inherited from invoker ³	Inherited from invoker ³	Yes
CURRENT PATH	The value of bind option PATH for the user-defined function or stored procedure package or inherited from invoker ⁵	The value of bind option PATH for the user-defined function or stored procedure package	Yes
CURRENT PRECISION	Inherited from invoker	The value of field DECIMAL ARITHMETIC on installation panel DSNTIP4	Yes
CURRENT RULES	Inherited from invoker	The value of bind option SQLRULES for the plan that invokes a user-defined function or stored procedure	Yes
CURRENT SERVER	Inherited from invoker	Inherited from invoker	Yes

Special registers

Table 19. Characteristics of special registers in a user-defined function or a stored procedure (continued)

Special register	Initial value when INHERIT SPECIAL REGISTERS option is specified	Initial value when DEFAULT SPECIAL REGISTERS option is specified	Routine can use SET statement to modify?
CURRENT SQLID	The primary authorization ID of the application process or inherited from invoker ⁶	The primary authorization ID of the application process	Yes ⁷
CURRENT TIME	New value for each SQL statement in the user-defined function or stored procedure package ¹	New value for each SQL statement in the user-defined function or stored procedure package ¹	Not applicable ⁴
CURRENT TIMESTAMP	New value for each SQL statement in the user-defined function or stored procedure package ¹	New value for each SQL statement in the user-defined function or stored procedure package ¹	Not applicable ⁴
CURRENT TIMEZONE	Inherited from invoker	Inherited from invoker	Not applicable ⁴
USER	Primary authorization ID of the application process	Primary authorization ID of the application process	Not applicable ⁴

Note:

1. If the user-defined function or stored procedure is invoked within the scope of a trigger, DB2 uses the timestamp for the triggering SQL statement as the timestamp for all SQL statements in the package.
2. DB2 allows parallelism at only one level of a nested SQL statement. If you set the value of the CURRENT DEGREE special register to ANY, and parallelism is disabled, DB2 ignores the CURRENT DEGREE value.
3. If the user-defined function or stored procedure definer specifies a value for COLLID in the CREATE FUNCTION statement, DB2 sets CURRENT PACKAGESET to the value of COLLID.
4. Not applicable because no SET statement exists for the special register.
5. If a program within the scope of the invoking program issues a SET statement for the special register before the user-defined function or stored procedure is invoked, the special register inherits the value from the SET statement. Otherwise, the special register contains the value that is set by the bind option for the user-defined function or stored procedure package.
6. If a program within the scope of the invoking program issues a SET CURRENT SQLID statement before the user-defined function or stored procedure is invoked, the special register inherits the value from the SET statement. Otherwise, CURRENT SQLID contains the authorization ID of the application process.
7. If the user-defined function or stored procedure package uses a value other than RUN for the DYNAMICRULES bind option, the SET CURRENT SQLID statement can be executed but does not affect the authorization ID that is used for the dynamic SQL statements in the package. The DYNAMICRULES value determines the authorization ID that is used for dynamic SQL statements. For more information, see the discussion of DYNAMICRULES in Chapter 2 of *DB2 Command Reference*.

Column names

The meaning of a column name depends on its context. A column name can be used to:

- Declare the name of a column, as in a CREATE TABLE statement or in a CREATE FUNCTION statement that defines a table function.
- Identify a column, as in a CREATE INDEX statement.
- Specify values of the column, as in the following contexts:
 - In a *column function*, a column name specifies all values of the column in the group or intermediate result table to which the function is applied. (Groups and intermediate result tables are explained in Chapter 4, “Queries,” which begins on page 347.) For example, MAX(SALARY) applies the function MAX to all values of the column SALARY in a group.
 - In a *GROUP BY* or *ORDER BY clause*, a column name specifies all values in the intermediate result table to which the clause is applied. For example, ORDER BY DEPT orders an intermediate result table by the values of the column DEPT.
 - In an *expression*, a *search condition*, or a *scalar function*, a column name specifies a value for each row or group to which the construct is applied. For example, when the search condition CODE = 20 is applied to some row, the value specified by the column name CODE is the value of the column CODE in that row.
- Temporarily, rename a column, as in the *correlation-clause* of a table referenced in a FROM clause.

Qualified column names

A qualifier for a column name can be a table name, a view name, an alias name, a synonym, or a correlation name.

Whether a column name can be qualified depends, like its meaning, on its context:

- In some forms of the COMMENT and LABEL ON statements, a column name must be qualified. This is shown in the syntax diagrams.
- Where the column name specifies values of the column, a column name can be qualified at the user’s option.
- In all other contexts, a column name must not be qualified. This rule will be mentioned in the discussion of each statement to which it applies.

Where a qualifier is optional, it can serve two purposes. See “Column name qualifiers to avoid ambiguity” on page 96 and “Column name qualifiers in correlated references” on page 97 for details.

Correlation names

A *correlation name* can be defined in the FROM clause of a query and in the first clause of an UPDATE or DELETE statement. For example, the clause FROM X.MYTABLE Z establishes Z as a correlation name for X.MYTABLE.

With Z defined as a correlation name for table X.MYTABLE, only Z should be used to qualify a reference to a column of X.MYTABLE in that SELECT statement.

A correlation name is associated with a table, view, nested table expression or table function only within the context in which it is defined. Hence, the same correlation

Column names

name can be defined for different purposes in different statements. In a nested table expression or table function, a correlation name is required.

As a qualifier, a correlation name can be used to avoid ambiguity or to establish a correlated reference. It can also be used merely as a shorter name for a table or view. In the example, Z might have been used merely to avoid having to enter X.MYTABLE more than once.

The use of a correlation name in the FROM clause also allows the option of specifying a list of column names to be associated with the columns of the result table. As with a correlation name, the listed column names should be the names that are used to reference the columns in that SELECT statement. For example, assume that the name of the first column in the DEPT table is DEPTNO. Given this FROM clause:

```
FROM DEPT D (NUM,NAME,MGR,ANUM,LOC)
```

You should use D.NUM instead of D.DEPTNO to reference the first column of the table.

Column name qualifiers to avoid ambiguity

In the context of a function, a GROUP BY clause, an ORDER BY clause, an expression, or a search condition, a column name refers to values of a column in some table, view, nested table expression, or table function. The tables, views, nested table expression, or table function reference that might contain the column are called the *object tables* of the context. Two or more object tables might contain columns with the same name. One reason for qualifying a column name is to name the table from which the column comes.

Table designators: A qualifier that names a specific object table is called a *table designator*. The clause that identifies the object tables also establishes the table designators for them. For example, the object tables of an expression in a SELECT clause are named in the FROM clause that follows it, as in this statement:

```
SELECT DISTINCT Z.EMPNO, EMPTIME, PHONENO  
  FROM DSN8710.EMP Z, DSN8710.EMPPROJECT  
 WHERE WORKDEPT = 'D11'  
   AND EMPTIME > 0.5  
   AND Z.EMPNO = DSN8710.EMPPROJECT.EMPNO;
```

This example illustrates how to establish table designators in the FROM clause:

- A correlation name that follows a table name, view name, nested table expression, or table function is a table designator. Thus, Z is a table designator and qualifies the first column name after SELECT.
- A table name or view name that is *not* followed by a correlation name is a table designator. Thus, the qualified table name, DSN8710.EMPPROJECT is a table designator and qualifies the EMPNO column.

Avoiding undefined or ambiguous references in DB2 SQL: When a column name refers to values of a column, exactly one object table must include a column with that name. The following situations are considered errors:

- No object table contains a column with the specified name. The reference is undefined.
- The column name is qualified by a table designator, but the table named does not include a column with the specified name. Again, the reference is undefined.
- The name is unqualified and more than one object table includes a column with that name. The reference is ambiguous.

Avoid ambiguous references by qualifying a column name with a uniquely defined table designator. If the column is contained in several object tables with different names, the table names can be used as designators.

Two or more object tables can be instances of the same table. A FROM clause that includes n references to the same table should include at least $n - 1$ unique correlation names.

For example, in the following FROM clause X and Y are defined to refer, respectively, to the first and second instances of the table EMP.

```
SELECT X.LASTNAME, Y.LASTNAME
  FROM DSN8710.EMP X, DSN8710.EMP Y
 WHERE Y.JOB = 'MANAGER'
   AND X.WORKDEPT = Y.WORKDEPT
   AND X.JOB <> 'MANAGER';
```

Column name qualifiers in correlated references

A *subselect* and a *fullselect* are forms of a query that can be used as a component of various SQL statements. Refer to Chapter 4, “Queries,” on page 347 for more information on subselects and fullselects. A subselect or a fullselect used within a search condition of any statement is called a *subquery*.

A subquery can include search conditions of its own, and these search conditions can, in turn, include subqueries. Thus, an SQL statement can contain a hierarchy of subqueries. Those elements of the hierarchy that contain subqueries are said to be at a higher level than the subqueries they contain.

Every element of the hierarchy has a clause that establishes one or more table designators. This is the FROM clause, except in the highest level of an UPDATE or DELETE statement. A search condition of a subquery can reference not only columns of the tables identified by the FROM clause of its own element of the hierarchy, but also columns of tables identified at any level along the path from its own element to the highest level of the hierarchy. A reference to a column of a table identified at a higher level is called a *correlated reference*.

A correlated reference to column C of table T can be of the form C, T.C, or Q.C, if Q is a correlation name defined for T. However, **a correlated reference in the form of an unqualified column name is not good practice**. The following explanation is based on the assumption that a correlated reference is always in the form of a qualified column name and that the qualifier is a correlation name.

A qualified column name, Q.C, is a correlated reference only if these three conditions are met:

- Q.C is used in a search condition or in a select list of a subquery.
- Q does not name a table used in the FROM clause of that subquery.
- Q does name a table used at some higher level.

Q.C refers to column C of the table or view at the level where Q is used as the table designator of that table or view. Because the same table or view can be identified at many levels, unique correlation names are recommended as table designators. If Q is used to name a table at more than one level, Q.C refers to the lowest level that contains the subquery that includes Q.C.

For example, in the following statement, the correlated reference X.WORKDEPT (in the last line) refers to the value of WORKDEPT in table DSN8710.EMP at the level

Column names

of the first FROM clause (which establishes X as a correlation name for DSN8710.EMP.). The statement lists employees who make less than the average salary for their department.

```
SELECT EMPNO, LASTNAME, WORKDEPT
      FROM DSN8710.EMP X
            WHERE SALARY < (SELECT AVG(SALARY)
                                FROM DSN8710.EMP
                                WHERE WORKDEPT = X.WORKDEPT);
```

The following example shows a correlated reference in the select list of the subquery.

```
SELECT T1.KEY1
      FROM BP1TBL T1
            GROUP BY T1.KEY1
            HAVING MAX(T1.KEY1) = (SELECT MIN(T1.KEY1) + MIN(T2.KEY1)
                                FROM BP2TBL T2);
```

Resolution of column name qualifiers and column names

Names in a FROM clause are either *exposed* or *non-exposed*. A correlation name for a table name, view name, nested table expression, or reference to a function reference is always exposed. A table name or a view name that is not followed by a correlation name is also exposed.

In IBM SQL and ANSI/ISO SQL, the exposed names in a FROM clause must be unique, and the qualifier of a column name must be an exposed name.

The rules for finding the referent of a column name qualifier are as follows:

1. Let Q be a one-, two-, or three-part name, and let Q.C denote a column name in subselect S. Q must designate a table or view identified in the statement that includes S and that table or view must have a column named C. An additional requirement differs for two cases:
 - If Q.C is *not* in a search-condition or S is *not* a subquery, Q must designate a table or view identified in the FROM clause of S. For example, if Q.C is in a SELECT clause, Q refers to a table or view in the following FROM clause.
 - If Q.C is in a *search-condition* and S is a subquery, Q must designate a table or view identified either in the FROM clause of S or in a FROM clause of a subselect that directly or indirectly includes S. For example, if Q.C is in a WHERE clause and S is the only subquery in the statement, the table or view that Q refers to is either in the FROM clause of S or the FROM clause of the subselect that includes S.
2. The same table or view can be identified more than once in the same statement. The particular occurrence of the table or view that Q refers to is determined by a procedure equivalent to the following steps:
 - a. The one- and two-part names in every FROM clause and the one- and two-part qualifiers of column names are expanded into a fully-qualified form.
For example, if a dynamic SQL statement uses FROM Q and DYNAMICRULES run behavior (RUN) is in effect, Q is expanded to S.A.Q, where S is the value of CURRENT SERVER and A is the value of CURRENT SQLID. (If DYNAMICRULES bind behavior is in effect instead, A is the plan or package qualifier as determined during the bind process.) We refer to this step later as “name completion”. An error occurs if the first part of every name (the location) is not the same.
 - b. Q, now a three-part name, is compared with every name in the FROM clause of S. If Q.C is in a *search-condition* and S is a subquery, Q is next compared with every name in the FROM clause of the subselect that

contains S. If that subselect is a subquery, Q is then compared with every name in the FROM clause of the subselect containing that subquery, and so on. If a FROM clause includes multiple names, the comparisons in that clause are made in order from left to right.

- c. The referent of Q is selected by these rules:
 - If Q matches exactly one name, that name is selected.
 - If Q matches more than one name, but only one exposed name, that exposed name is selected.
 - If Q matches more than one exposed name, the first of those names is selected.
 - If Q matches more than one name, none of which are exposed names, the first of those names is selected.

If Q does not match any name, or if the table or view designated by Q does not include a column named C, an error occurs.

- d. Otherwise, Q.C is resolved to column C of the occurrence of the table or view identified by the selected name.
3. A warning occurs for any of these cases:
 - The selected name is not an exposed name.
 - The selected name is an exposed name that has an unexposed duplicate that appears before the selected name in the ordered list of names to which Q is compared.
 - The selected name is an exposed name that has an exposed duplicate in the same FROM clause.
 - Another name would have been selected had the matching been performed before name completion.

The warnings indicate cases of ambiguous references in which the referent selected might not be the same one that would have been selected in releases of DB2 before Version 2 Release 3.

The rules for resolving column name qualifiers apply to every SQL statement that includes a subselect and are applied before synonyms and aliases are resolved. In the case of a searched UPDATE or DELETE statement, the first clause of the statement identifies the table or view to be updated or deleted. That clause can include a correlation name and, with regard to name resolution, is equivalent to the first FROM clause of a SELECT statement. For example, a subquery in the search condition of an UPDATE statement can include a correlated reference to a column of the updated rows.

The rules for column names in the ORDER BY clause are the same as other clauses.

References to variables

A *variable* is a *host variable* or an *SQL variable* that is referenced in an SQL statement. Host variables are defined by statements of a host language. SQL variables are defined by an SQL compound-statement in an SQL procedure. Variables cannot be referenced in dynamic SQL statements; parameter markers must be used instead. In this book, unless otherwise noted, the term *host variable* in syntax diagrams is used to indicate where a host-variable, SQL variable, or parameter marker can be used.

References to variables

For more information on host variables, see “References to host variables.” For more information on SQL variables, see “compound-statement” on page 952. For more information on parameter markers, see “Parameter markers” on page 852.

References to host variables

A *host variable* is either of these items that is referred to in an SQL statement:

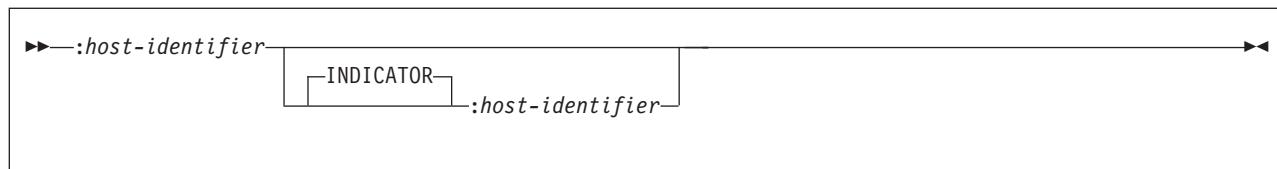
- A variable in a host language such as a PL/I variable, C variable, Fortran variable, COBOL data item, or Assembler language storage area
- A host language construct that was generated by an SQL precompiler from a variable declared using SQL extensions

Host variables are defined directly by statements of the host language or indirectly by SQL extensions as described in Part 2 of *DB2 Application Programming and SQL Guide*. Host variables cannot be referenced in dynamic SQL statements.

In PL/I, C, and COBOL, host variables can be referred to in ways that do not apply to Fortran and Assembler language. This is explained in “Host structures in PL/I, C, and COBOL” on page 103. The following applies to all host languages.

The term *host-variable*, as used in the syntax diagrams, shows a reference to a host variable. In a SET *assignment* statement and the INTO clause of a FETCH, SELECT INTO, or VALUES INTO statement, a host variable is an output variable to which a value is assigned by DB2. In all other contexts, a host variable is an input variable which provides a value to DB2.

The general form of a host variable reference is:



Each host identifier must be declared in the source program, except in a program written in REXX. The first host identifier designates the main variable; the second host identifier designates its indicator variable. The variable designated by the second host identifier must be a small integer. The purposes of the indicator variable are to:

- Specify the null value. A negative value of the indicator variable specifies the null value. A -2 null indicates a numeric conversion or arithmetic expression error occurred in the SELECT list of an outer SELECT statement.
- Record the original length of a truncated string.
- Indicate that a character could not be converted.
- Record the seconds portion of a time if the time is truncated on assignment to a host variable.

For example, if :V1:V2 is used to specify an insert or update value, and if V2 is negative, the value specified is the null value. If V2 is not negative, the value specified is the value of V1.

Similarly, if :V1:V2 is specified in a FETCH or SELECT INTO statement, and if the value returned is null, V1 is not changed and V2 is set to -1 or -2. It is set to -1 if

the value selected was actually null. It is set to -2 if the null value was returned because of numeric conversion errors or arithmetic expression errors in the SELECT list of an outer SELECT statement. It is also set to -2 as the result of a character conversion error. If the value returned is not null, that value is assigned to V1, and V2 is set to zero (unless the assignment to V1 requires string truncation, in which case V2 is set to the original length of the string). If an assignment requires truncation of the seconds part of a time, V2 is set to the number of seconds.

If the second host identifier is omitted, the host variable does not have an indicator variable: the value specified by the host variable :V1 is always the value of V1 and null values cannot be assigned to the variable. Thus, this form should not be used in an INTO clause unless the corresponding result column cannot contain null values. If this form is used for an output host variable and the value returned is null, DB2 will generate an error at run time.

An SQL statement that refers to host variables must be within the scope of the declaration of those host variables. For host variables referred to in the SELECT statement of a cursor, that rule applies to the OPEN statement rather than to the DECLARE CURSOR statement.

All references to host variables must be preceded by a colon. (There is one exception to this rule. A colon is not required if the host variable is used with an SQLDA descriptor.) If an SQL statement references a host variable without a preceding colon, the precompiler issues an error for the missing colon or interprets the host variable as an unqualified column name, which might lead to unintended results. The interpretation of a host variable without a colon as a column name occurs when the host variable is referenced in a context in which a column name can also be referenced.

Host variables in dynamic SQL

In dynamic SQL statements, parameter markers are used instead of host variables. A parameter marker is a question mark (?) that represents a position in a dynamic SQL statement where the application will provide a value; that is, where a host variable would be found if the statement string were a static SQL statement. The following examples show a static SQL statement that uses host variables and a dynamic statement that uses parameter markers:

```
INSERT INTO DEPT VALUES (:HV_DEPTNO, :HV_DEPTNAME, :HV_MGRNO, :HV_ADMRDEPT)  
INSERT INTO DEPT VALUES (?, ?, ?, ?)
```

For more information on parameter markers, see “Parameter markers” on page 852 under the PREPARE statement.

References to LOB host variables

Regular LOB variables (CLOB, DBCLOB, and BLOB) and LOB locator variables (see “References to LOB locator variables” on page 102) can be defined in all host languages, except in REXX. Where LOBs are allowed, the term *host-variable* in a syntax diagram can refer to a regular host variable or a locator variable. Since these variables are not native data types in host programming languages, SQL extensions are used and the precompilers generate the host language constructs necessary to represent each variable.

When it is possible to define a host variable that is large enough to hold an entire LOB value and the performance benefit of delaying the transfer of data from the server is not required, a LOB locator is not needed. However, host language

References to Host Variables

restrictions, storage restrictions, or performance often dictate against storing an entire LOB value in temporary storage. When it is preferable not to store an entire LOB value, a LOB locator can be used to refer to the LOB value and portions of the LOB value can be selected into or updated from host variables that contain only a portion of the LOB value.

Like all other host variables, a LOB locator variable can have an associated indicator variable. Indicator variables for LOB locator variables behave in the same way as indicator variables for other data types. When a null value is returned from the database, the indicator variable is set and the locator host variable is unchanged. This means a locator can never represent a null value.

References to LOB locator variables

A LOB locator variable is a host variable that contains the locator representing a LOB value on the database server.

A locator variable in an SQL statement must identify a LOB locator variable described in the program according to the rules for declaring locator variables. This is always indirectly through an SQL statement. For example, in C:

```
static volatile SQL TYPE IS CLOB_LOCATOR *loc1;
```

The term *locator-variable*, as used in the syntax diagrams, shows a reference to a LOB locator variable. The meta-variable *locator-variable* can be expanded to include a *host-identifier* the same as that for *host-variable*.

Like all other host variables, a LOB locator variable can have an associated indicator variable. Indicator variables for LOB locator variables behave in the same way as indicator variables for other data types. When a null value is returned from the database, the indicator variable is set and the locator host variable is unchanged. This means a locator can never represent a null value. However, when the indicator variable associated with a LOB locator is null, the value of the referenced LOB value is null.

If a locator variable does not currently represent any value, an error occurs when the locator variable is referenced.

At transaction commit, all LOB locators that were acquired by the transaction are released unless a HOLD LOCATOR statement was issued for the LOB locator. At transaction termination, all LOB locators are released.

References to stored procedure result sets

When an application needs to access a result set returned from a stored procedure, the invoking application must first define a result set locator to access the result set. An ASSOCIATE LOCATOR statement defines a result set locator, which identifies the stored procedure that returns the result set. The DESCRIBE PROCEDURE statement can be used to determine the number of result sets that a stored procedure returns, and the DESCRIBE CURSOR statement can be used to get information about a result set. The ALLOCATE CURSOR statement is used to define a cursor and associate it with a result set locator. The application then issues FETCH statements to retrieve rows from the result set.

References to result set locator variables

A result set locator variable is a host variable that contains the locator that identifies a stored procedure result set.

A result set locator variable in an SQL statement must identify a result set locator variable described in the program according to the rules for declaring result set locator variables. This is always indirectly through an SQL statement. For example, in C:

```
static volatile SQL TYPE IS RESULT_SET_LOCATOR *loc1;
```

The term *rs-locator-variable*, as used in the syntax diagrams, shows a reference to a result set locator variable. The meta-variable *rs-locator-variable* can be expanded to include a *host-identifier* the same as that for *host-variable*.

When the indicator variable associated with a result set locator is null, the referenced result set is not defined.

If a result set locator variable does not currently represent any stored procedure result set, an error occurs when the locator variable is referenced.

A commit operation destroys all open cursors that were declared in the stored procedure without the WITH HOLD operation and the result set locators that are associated with those cursors. Otherwise, a cursor and its associated result set locator persist past the commit.

Host structures in PL/I, C, and COBOL

A *host structure* is a PL/I structure, C structure, or COBOL group that is referred to in an SQL statement. Host structures are defined by statements of the host language, as explained in Part 2 of *DB2 Application Programming and SQL Guide*. As used here, the term “host structure” does not include an SQLCA or SQLDA.

The form of a host structure reference is identical to the form of a host variable reference. The reference :S1:S2 is a host structure reference if S1 names a host structure. If S1 designates a host structure, S2 must be a small integer variable or an array of small integer variables. S1 is the host structure and S2 is its indicator array.

A host structure can be referred to in any context where a list of host variables can be referenced. A host structure reference is equivalent to a reference to each of the host variables contained within the structure in the order which they are defined in the host language structure declaration. The *n*th variable of the indicator array is the indicator variable for the *n*th variable of the host structure.

In PL/I, for example, if V1, V2, and V3 are declared as the variables within the structure S1, the statement:

```
EXEC SQL FETCH CURSOR1 INTO :S1;
```

is equivalent to:

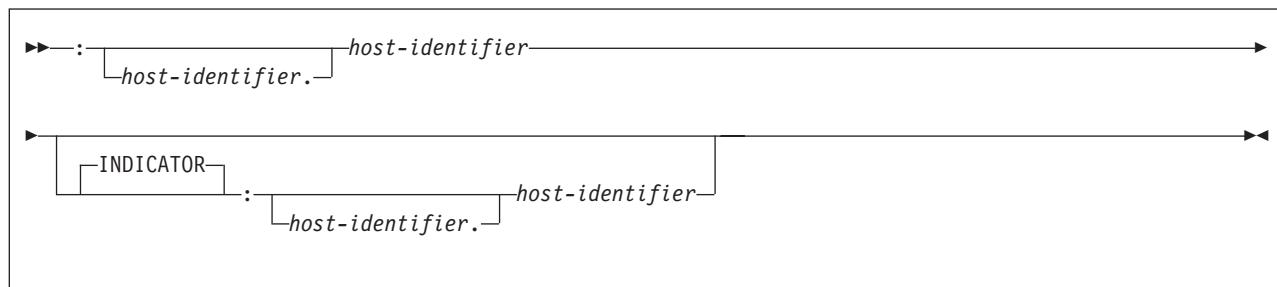
```
EXEC SQL FETCH CURSOR1 INTO :V1, :V2, :V3;
```

If the host structure has *m* more variables than the indicator array, the last *m* variables of the host structure do not have indicator variables. If the host structure has *m* fewer variables than the indicator array, the last *m* variables of the indicator array are ignored. These rules also apply if a reference to a host structure includes an indicator variable or a reference to a host variable includes an indicator array. If an indicator array or variable is not specified, no variable of the host structure has an indicator variable.

Host Structures in PL/I, C, and COBOL

In addition to structure references, individual host variables or indicator variables in PL/I, C, and COBOL can be referred to by qualified names. The qualified form is a host identifier followed by a period and another host identifier. The first host identifier must name a structure, and the second host identifier must name a host variable within that structure.

In PL/I, C, and COBOL, the syntax of *host-variable* is:



In general, a *host-variable* in an expression must identify a host variable (not a structure) described in the program according to the rules for declaring host variables. However, there are a few SQL statements that allow a host variable in an expression to identify a structure, as specifically noted in the descriptions of the statements.

The following examples show references to host variables and host structures:

:V1 :S1.V1 :S1.V1:V2 :S1.V2:S2.V4

Functions

A *function* is an operation denoted by a function name followed by zero or more operands that are enclosed in parentheses. It represents a relationship between a set of input values and a set of result values. The input values to a function are called *arguments*. For example, a function can be passed two input arguments that have date and time data types and return a value with a timestamp data type as the result.

Types of functions

There are several ways to classify functions. One way to classify functions is as built-in functions, user-defined functions, or cast functions that are generated for distinct types.

Built-in functions

Built-in functions are IBM-supplied functions that come with DB2 for OS/390 and z/OS and are in the SYSIBM schema. Built-in functions include operator functions such as "+", column functions such as AVG, and scalar functions such as SUBSTR. Although both column and scalar functions return a single value, their arguments differ. The argument of a column function is a set of like values. Each argument of a scalar function is a single value.

The built-in functions are in schema SYSIBM. A built-in function can be invoked with or without its schema name. Regardless of whether a schema name qualifies the function name, DB2 uses function resolution to determine which function to use. For more information on the process of function resolution, see "Function resolution" on page 107. For specific built-in functions, see function description in Chapter 3, "Functions," on page 155.

User-defined functions

User-defined functions are functions that are registered to DB2 in catalog table SYSIBM.SYSROUTINES using the CREATE FUNCTION statement. These functions allow users to extend the function of the database system by adding their own or third party vendor function definitions.

A user-defined function can be an *external*, *sourced*, or SQL scalar function. An external function is defined to the database with a reference to a load module that is executed when the function is invoked. A sourced function is defined to the database with a reference to a built-in function or another user-defined function. Sourced functions are useful for supporting the use of built-in column and scalar functions for distinct types. An SQL function is defined in the RETURN clause of the function.

A user-defined function resides in the schema in which it was registered. The schema cannot be SYSIBM. In addition to being external or sourced, user-defined functions can be further categorized as scalar, column, or table functions.

To help you define and implement user-defined functions, sample user-defined functions are supplied with DB2. You can also use these sample user-defined functions in your application program just as you would any other user-defined function if the appropriate installation job has been run. For a list of the sample user-defined functions, see Appendix G, “Sample user-defined functions,” on page 1155. For more information on creating and using user-defined functions, see Part 2 of *DB2 Application Programming and SQL Guide*.

```
#  
#  
#  
#  
# One set of user-defined functions that DB2 provides are DB2 MQSeries functions,  
# which integrate MQSeries messaging operations within SQL statements. The  
# functions help you integrate MQSeries messaging with database applications. You  
# can use the functions to access MQSeries messaging from within SQL statements  
# and to combine MQSeries messaging with DB2 database access.
```

```
#  
#  
#  
#  
#  
#  
#  
#  
# The MQSeries functions are installed into DB2 and provide access to the MQSeries  
# server using AMI (Application Messaging Interface). AMI supports the use of an  
# external configuration file (the AMI repository) to store configuration information. AMI  
# uses two key concepts: service point and policy. A service point is a logical endpoint  
# from which a message may be sent or received. Policy defines the quality of  
# service option that should be used for a given messaging operation. The MQSeries  
# functions can read, receive, or send messages to the queue specified by the  
# service and policy in the AMI repository. The functions can be scalar or table  
# functions. For more information on MQSeries functions, see the function  
# descriptions in Chapter 3, “Functions,” on page 155.
```

Cast functions

Cast functions are automatically generated by DB2 when a distinct type is created using the CREATE DISTINCT TYPE statement. These functions support casting from the distinct type to the source type and from the source type to the distinct type. The ability to cast between the data types is important because a distinct type is compatible only with itself.

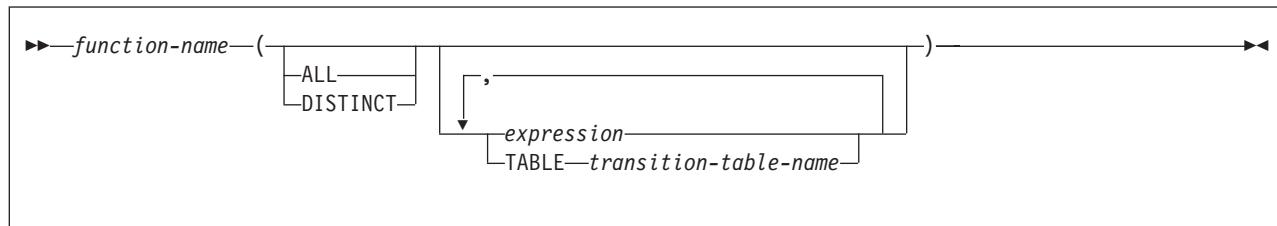
The generated cast functions reside in the same schema as the distinct type for which they were created. The schema cannot be SYSIBM. For more information on the functions that are generated for a distinct type, see “CREATE DISTINCT TYPE” on page 522.

Additional way to classify functions

Another way to classify functions is as column, scalar, or table functions, depending on the input data values and result values. For a list of the column, scalar, and table functions and information on these functions, see Chapter 3, “Functions,” on page 155.

- A *column function* returns a single-value result for the argument it receives. The argument is a set of like values (such as the values of a column). Column functions are sometimes called *aggregating functions*. Built-in functions and user-defined sourced functions can be column functions. An external user-defined function cannot be a column function.
- A *scalar function* also returns a single-value result for the arguments it receives. Each argument is a single value. Built-in functions and user-defined functions, both external and sourced, can be scalar functions. The functions that are created for distinct types are also scalar functions.
- A *table function* returns a table for the set of arguments it receives. Each argument is a single value. A table function can only be referenced in the FROM clause of a subselect. Table functions can be used to apply SQL language processing power to data that is not DB2 data or to convert such data into a DB2 table. For example, a table function can take a file and convert it to a table, get data from the World Wide Web and tabularize it, or access a Lotus® Notes™ database and return information about mail messages. Only external user-defined functions can be table functions.

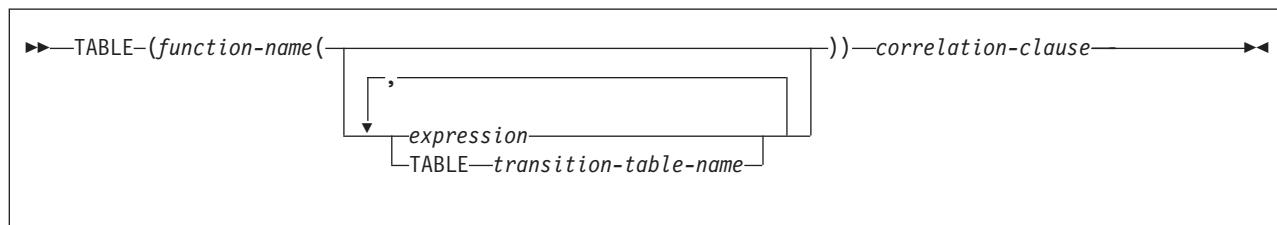
Each reference to a scalar or column function (either built-in or user-defined) conforms to the following syntax:



In the above syntax, *expression* cannot include a column function. See “Expressions” on page 111 for other rules for *expression*.

The ALL or DISTINCT keyword can only be specified for a column function or a user-defined function that is sourced on a column function. The TABLE keyword can only be used in a trigger body.

Each reference to a table function conforms to the following syntax:



In the above syntax, *expression* is the same as it is for a scalar or column function. For more details on referencing a table function, see the description of the FROM clause on page 352, the only place where a table function can be referenced.

Function resolution

A function is invoked by its function name, which is implicitly or explicitly qualified with a schema name, followed by parentheses that enclose the arguments to the function. Within the database, each function is uniquely identified by its *function signature*, which is its schema name, function name, the number of parameters, and the data types of the parameters. Thus, a schema can contain several functions that have the same name but each of which have a different number of parameters or parameters with different data types. Also, a function with the same name, number of parameters, and types of parameters can exist in multiple schemas.

Because multiple functions with the same name can exist in the same schema or different schemas, DB2 must determine which function to execute. The process of choosing the function is called *function resolution*.

Function resolution is similar for functions that are invoked with a qualified or unqualified function name with the exception that for an unqualified name, DB2 needs to search more than one schema.

Qualified function resolution: When a function is invoked with a schema name and a function name, DB2 only searches the specified schema to resolve which function to execute. DB2 finds the appropriate function instance when all of the following conditions are true:

- The name of the function instance matches the name in the function invocation.
- The number of input parameters in the function instance matches the number of function arguments in the function invocation.
- The invoker of the function is authorized to execute the function instance.
- The data type of each input argument of the function invocation matches or is *promotable* to the data type of the corresponding parameter of the function instance.

For a function invocation that passes a transition table, the data type, length, precision, and scale of each column in the transition table must match exactly the data type, length, precision, and scale of each column of the table that is named in the function instance definition.

If the function invocation contains no untyped parameter markers, the comparison of data types results in one best fit, which is the choice for execution (see “Method of finding the best fit” on page 108). For information on the promotion of data types, see “Promotion of data types” on page 61.

For a function invocation that contains untyped parameter markers, the data types of those parameter markers are considered to match or be promotable to the data types of the parameters in the function instance.

- The create timestamp for the function must be older than the bind timestamp for the package or plan in which the function is invoked.

If a function invoked from a trigger body receives a transition table, and the invocation occurs during an automatic rebind, the form of the invoked function used for function selection includes only the columns of the table that existed at the time of the original BIND or REBIND package or plan for the invoking program.

If the function invocation contains untyped parameter markers, the comparison can result in more than one best fit. In that case, DB2 returns an error.

If DB2 authorization checking is in effect, and DB2 performs an automatic rebind on a plan or package that contains a user-defined function invocation, any

Functions

user-defined functions that were created after the original BIND or REBIND of the invoking plan or package are not candidates for execution.

If you use an access control authorization exit routine, some user-defined functions that were not candidates for execution before the original BIND or REBIND of the invoking plan or package might become candidates for execution during the automatic rebind of the invoking plan or package. See Appendix B (Volume 2) of *DB2 Administration Guide* for information about function resolution with access control authorization exit routines.

If no function in the schema meets these criteria, an error occurs. If a function is selected, its successful use depends on it being invoked in a context in which the returned result is allowed. For example, if the function returns an integer data type where a character data type is required, or returns a table where a table function is not allowed, an error occurs.

Unqualified function resolution: When a function is invoked with only a function name and no schema name, DB2 needs to search more than one schema to resolve the function instance to execute. DB2 uses these steps to choose the function:

1. The *SQL path* contains the list of schemas to search. For each schema in the path, DB2 selects a candidate function based on the same criteria described immediately above for qualified function resolution. However, if no function in the schema meets the criteria, an error does not occur, and a candidate function is not selected for that schema.
2. After identifying the candidate functions for the schemas in the path, DB2 selects the candidate with the best fit as the function to execute. If more than one schema contains the function instance with the best fit (the function signatures are identical except for the schema name), DB2 selects the function whose schema is earliest in the SQL path. If no function in any schema in the SQL path meets the criteria, an error occurs.

The create timestamp of a user-defined function must be older than the timestamp resulting from an explicit bind for the plan or package containing the function invocation. During autobind, built-in functions introduced in a later DB2 release than the DB2 release that was used to explicitly bind the package or plan are not considered for function resolution.

For more information on user-defined functions, such as how you can simplify function resolution or use the DSN_FUNCTION_TABLE to see how DB2 resolves a function, see *DB2 Application Programming and SQL Guide*.

Method of finding the best fit

More than one function instance with the same name might be a candidate for execution. In that case, DB2 compares the argument and parameter data types to determine which function is the best fit for the invocation.

If the data types of all the parameters for a given function are the same as those of the arguments in the function invocation, that function is the best fit. If there is no exact match, DB2 compares the data types in the parameter lists from left to right, using this method:

1. DB2 compares the data types of the first argument in the function invocation to the data type of the first parameter in each function. Any length, precision, scale, subtype, and encoding scheme attributes of the data types are not considered in the comparison.

2. For this argument, if one function has a data type that fits the function invocation better than the data types in the other functions, that function is the best fit. The precedence list for the promotion of data types in Table 6 on page 62 shows the data types that fit each data type, in best-to-worst order.
3. If the data types of the first parameter for all the candidate functions fit the function invocation equally well, DB2 repeats this process for the next argument of the function invocation. DB2 continues this process for each argument until a best fit is found.

Examples of function resolution: The following examples illustrate function resolution.

Example 1: Assume that MYSHEMA contains two functions, both named FUNA, that were registered with these partial CREATE FUNCTION statements.

1. CREATE FUNCTION MYSHEMA.FUNA (VARCHAR(10), INT, DOUBLE) ...
2. CREATE FUNCTION MYSHEMA.FUNA (VARCHAR(10), REAL, DOUBLE) ...

Also assume that a function with three arguments of data types VARCHAR(10), SMALLINT, and DECIMAL is invoked with a qualified name:

```
MYSHEMA.FUNA(VARCHARCOL, SMALLINTCOL, DECIMALCOL)
```

The data types of the first parameter for the two function instances in the schema, which are both VARCHAR(10), fit the data type of the first argument of the function invocation, which is VARCHAR(10), equally well. However, for the second parameter, the data type of the first function (INT) fits the data type of the second argument (SMALLINT) better than the data type of second function (REAL). Therefore, DB2 selects Function 1 as the function instance to execute.

Example 2: Assume that these functions were registered with these partial CREATE FUNCTION statements:

1. CREATE FUNCTION SMITH.ADDIT (CHAR(5), INT, DOUBLE) ...
2. CREATE FUNCTION SMITH.ADDIT (INT, INT, DOUBLE) ...
3. CREATE FUNCTION SMITH.ADDIT (INT, INT, DOUBLE, INT) ...
4. CREATE FUNCTION JOHNSON.ADDIT (INT, DOUBLE, DOUBLE) ...
5. CREATE FUNCTION JOHNSON.ADDIT (INT, INT, DOUBLE) ...
6. CREATE FUNCTION TODD.ADDIT (REAL) ...
7. CREATE FUNCTION TAYLOR.SUBIT (INT, INT, DECIMAL) ...

Also assume that the SQL path at the time an application invokes a function is "TAYLOR" "JOHNSON", "SMITH". The function is invoked with three data types (INT, INT, DECIMAL) as follows:

```
SELECT ... ADDIT(INTCOL1, INTCOL2, DECIMALCOL) ...
```

Function 5 is chosen as the function instance to execute based on the following evaluation:

- Function 6 is eliminated as a candidate because schema TODD is not in the SQL path.
- Function 7 in schema TAYLOR is eliminated as a candidate because it does not have the correct function name.
- Both Function 4 and 5 in schema JOHNSON are candidates because the data types of their parameters match or are promotable to the data types of the arguments. However, Function 5 is chosen as the better candidate because although the data types of the first parameter of both functions (INT) match the first argument (INT), the data type of the second parameter of Function 5 (INT) is a better match of the second argument (INT) than Function 4 (DOUBLE).

Functions

- Function 1 and 3 in schema SMITH are eliminated as candidates. The CHAR data type of the first parameter of Function 1 is not promotable to INT. Function 3 has the wrong number of parameters. Function 2 is a candidate because the data types of its parameters match or are promotable to the data types of the arguments.
- Of the remaining candidates, Function 2 and 5, DB2 selects Function 5 because schema JOHNSON comes before schema SMITH in the SQL path.

SQL path considerations for built-in functions

Function resolution applies to all functions, including built-in functions. The built-in functions are in schema SYSIBM. If a built-in function is invoked without its schema name, the SQL path is searched. If SYSIBM is not first in the path, it is possible that DB2 will select another function instead of the intended function. If schema SYSIBM or SYSPROC is not explicitly specified in the SQL path, the schema is implicitly assumed at the front of the path. DB2 adds implicitly assumed schemas in the order of SYSIBM and SYSPROC. See “Schemas and the SQL path” on page 40 for information on how to specify the path so that the intended function is selected when it is invoked with an unqualified name.

Function invocation

Once the function is selected, there are still possible reasons why the use of the function may not be permitted. Each function is defined to return a result with a specific data type. If this result data type is not compatible with the context in which the function is invoked, an error occurs. For example, assume functions named STEP are defined with different data types:

```
STEP(SMALLINT) returns CHAR(5)  
STEP(DOUBLE) returns INTEGER
```

Assume also that the function is invoked with the following function reference (where S is a SMALLINT column):

```
SELECT ... 3+STEP(S) ...
```

Because there is an exact match on argument type, the first STEP is chosen. An error occurs on the statement because the result type is CHAR(5) instead of a numeric type as required for an argument of the addition operator.

An error also occurs in the following examples:

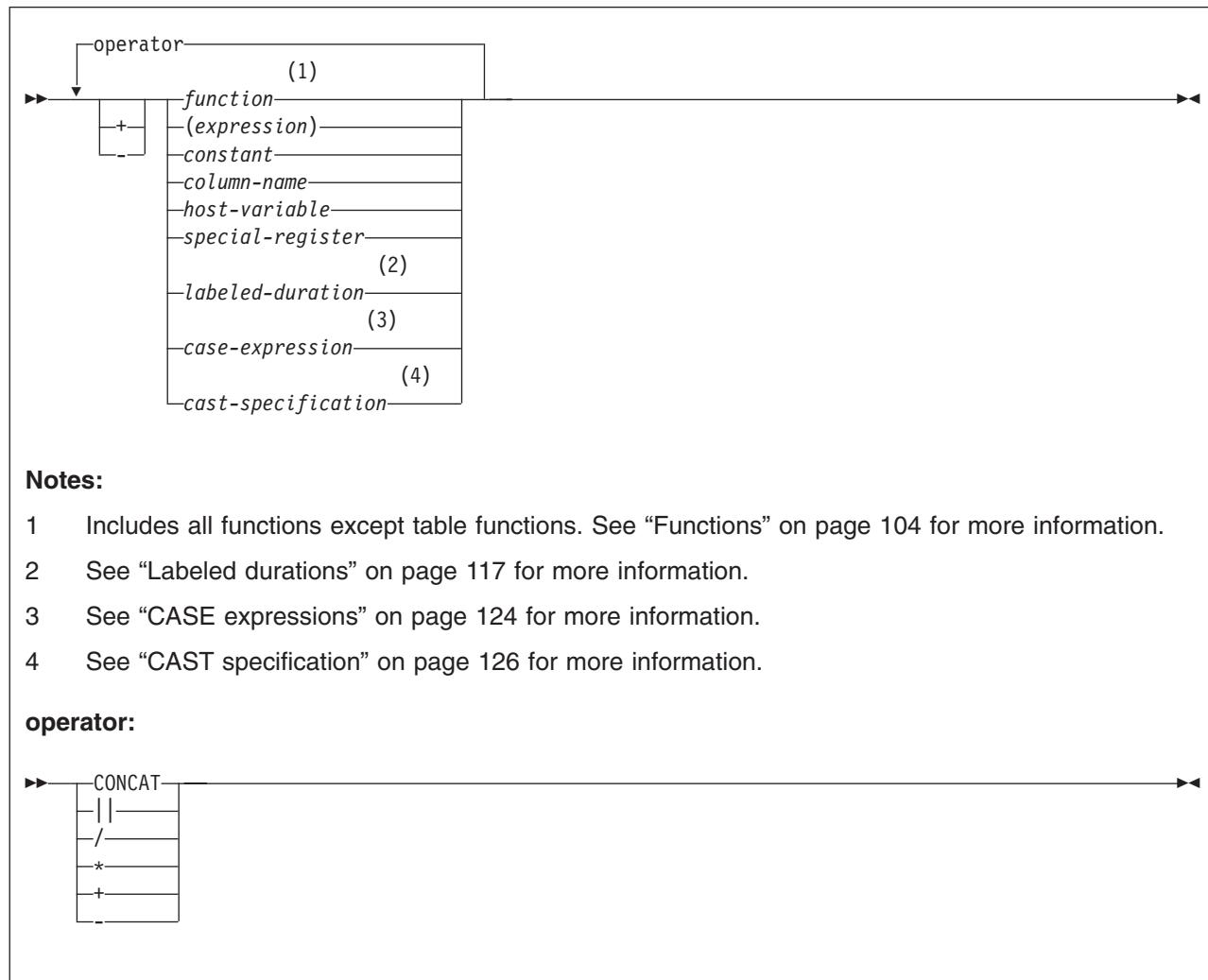
- The function is referenced in a FROM clause, but the function selected by the function resolution step is a scalar or column function.
- The function calls for a scalar or column function, but the function resolution step is a table function.

In cases where the arguments of the function invocation are not an exact match to the data types of the parameters of the selected function, the arguments are converted to the data type of the parameter at execution using the same rules as assignment to columns. See “Assignment and comparison” on page 64. Problems with conversions can also occur when precision, scale, length, or the encoding scheme differs between the argument and the parameter. Conversion might occur for a character string argument when the corresponding parameter of the function has a different encoding scheme or CCSID. For example, an error occurs on function invocation when mixed data that actually contains DBCS characters is specified as an argument and the corresponding parameter of the function is declared with an SBCS subtype.

Additionally, a character FOR BIT DATA argument cannot be passed as input for a parameter that is not defined as character FOR BIT DATA. Likewise, a character argument that is not FOR BIT DATA cannot be passed as input for a parameter that is defined as character FOR BIT DATA.

Expressions

An *expression* specifies a value. The form of an expression is as follows:



Notes:

- 1 Includes all functions except table functions. See “Functions” on page 104 for more information.
- 2 See “Labeled durations” on page 117 for more information.
- 3 See “CASE expressions” on page 124 for more information.
- 4 See “CAST specification” on page 126 for more information.

operator:



Without operators

If no operators are used, the result of the expression is the specified value.

Examples:

SALARY :SALARY 'SALARY' MAX(SALARY)

With the concatenation operator

Both CONCAT and the vertical bars (||) represent the concatenation operator. Vertical bars (or the characters that must be used in place of vertical bars in some countries¹⁴) can cause parsing errors in statements passed from one DBMS to another. The problem occurs if the statement undergoes character conversion with certain combinations of source and target CCSIDs¹⁴. Thus, CONCAT is the preferable concatenation operator.

When two strings operands are concatenated, the result of the expression is a string. The operands of concatenation must be compatible strings. A binary string cannot be concatenated with a character string, including character strings that are defined as FOR BIT DATA (for more information on the compatibility of data types, see the compatibility matrix in Table 9 on page 65). A distinct type that is sourced on a string type can be concatenated only if an appropriate user-defined function is created, as explained at the end of this section.

If either operand can be null, the result can be null, and if either is null, the result is the null value. Otherwise, the result consists of the first operand string followed by the second.

Table 20 shows how the string operands determine the data type and the length attribute of the result (the order in which the operands are concatenated has no effect on the result).

Table 20. Data type and length of concatenated operands

One operand	Other operand	Combined length attribute	Result ¹
CHAR(A)	CHAR(B)	<256	CHAR(A+B) ²
	CHAR(B)	>255	VARCHAR(A+B)
	VARCHAR(B)	-	VARCHAR(A+B)
VARCHAR(A)	VARCHAR(B)	-	VARCHAR(A+B)
CLOB(A)	CHAR(B)	-	CLOB(MIN(A+B, 2G))
	VARCHAR(B)	-	CLOB(MIN(A+B, 2G))
	CLOB(B)	-	CLOB(MIN(A+B, 2G))
GRAPHIC(A) UTF-16 UNICODE DATA ⁴	CHAR(B)	-	VARGRAPHIC(A+B) ³
	VARCHAR(B)	-	VARGRAPHIC(A+B) ³
VARGRAPHIC(A) UTF-16 UNICODE DATA ⁴	CHAR(B)	-	VARGRAPHIC(A+B) ³
	VARCHAR(B)	-	VARGRAPHIC(A+B) ³
DBCLOB(A)	GRAPHIC(B)	-	DBCLOB(MIN(A+B, 1G))
	VARGRAPHIC(B)	-	DBCLOB(MIN(A+B, 1G))
	DBCLOB(B)	-	DBCLOB(MIN(A+B, 1G))
BLOB(A)	BLOB(B)	-	BLOB(MIN(A+B, 2G))

14. DB2 supports code point combinations X'4F4F', X'BBBB', and X'5A5A' to mean concatenation. X'BBBB' and X'5A5A' are interpreted to mean concatenation only on single byte character set DB2 subsystems.

Table 20. Data type and length of concatenated operands (continued)

One operand	Other operand	Combined length attribute	Result ¹
Notes:			
1.	2G represents 2 147 483 647 bytes 1G represents 1 073 741 823 double-byte characters		
2.	Neither CHAR(A) nor CHAR(B) must contain mixed data. If either operand contains mixed data, the result is VARCHAR(A+B).		
3.	CHAR(B) or VARCHAR(B) is converted to UTF-16 and the result is concatenated to GRAPHIC(A) or VARGRAHPI(A)		
4.	UTF-16 data is the only graphic data that can be used with a character operand..		

As Table 20 on page 112 shows, the length of the result is the sum of the lengths of the operands. However, the length of the result is two bytes less if redundant shift code characters are eliminated from the result. Redundant shift code characters exist when both character strings are EBCDIC mixed data, and the first string ends with a “shift-in” character (X'0F') and the second operand begins with a “shift-out” character (X'0E'). These two shift code characters are removed from the result.

The CCSID of the result is determined by the rules set forth in “Character conversion in unions and concatenations” on page 366. Some consequences of those rules are the following:

- If either operand is BIT data, the result is BIT data.
- The conversion that occurs when SBCS data is compared with mixed data depends on the encoding scheme. If the encoding scheme is Unicode, the SBCS operand is converted to MIXED. Otherwise, the conversion depends on the field MIXED DATA on installation panel DSNTIPP for the DB2 that does the comparison:
 - Mixed data if the MIXED DATA option at the server is YES¹⁵
 - SBCS data if the MIXED DATA option at the server is NO.¹⁶

If an operand is a string from a column with a field procedure, the operation applies to the decoded form of the value. The result does not inherit the field procedure.

One operand of concatenation can be a parameter marker. When one operand is a parameter marker, its data type and length attributes are considered to be the same as those for the operand that is not a parameter marker. The order of concatenation operations must be considered to determine these attributes in the case of nested concatenation.

No operand of concatenation can be a distinct type even if the distinct type is sourced on a character data type. To concatenate a distinct type, create a user-defined function that is sourced on the CONCAT operator. For example, if distinct types TITLE and TITLE_DESCRIPTION were both sourced on data type VARCHAR(25), the following user-defined function, named ATTACH, could be used to concatenate the two distinct types:

```
CREATE FUNCTION ATTACH (TITLE, TITLE_DESCRIPTION)
RETURNS VARCHAR(50) SOURCE CONCAT_(VARCHAR(), VARCHAR())
```

15. The result is not necessarily well-formed mixed data.

16. If the mixed data cannot be converted to pure SBCS data, an error occurs.

Expressions

Alternatively, the concatenation operator could be overloaded by using a user-defined function to add the distinct types:

```
CREATE FUNCTION "||" (TITLE, TITLE_DESCRIPTION)
RETURNS VARCHAR(50) SOURCE CONCAT (VARCHAR(), VARCHAR())
```

With arithmetic operators

If arithmetic operators are used, the result of the expression is a number derived from the application of the operators to the values of the operands. The result of the expression can be null. If any operand has the null value, the result of the expression is the null value. Arithmetic operators (except unary plus, which is meaningless) must not be applied to strings. For example, USER+2 is invalid. Multiplication and division operators must not be applied to datetime values, which can only be added and subtracted.

The prefix operator + (*unary plus*) does not change its operand. The prefix operator - (*unary minus*) reverses the sign of a nonzero operand. If the data type of A is *small integer*, the data type of -A is *large integer*. The first character of the token following a prefix operator must not be a plus or minus sign.

The *infix operators* +, -, *, and / specify addition, subtraction, multiplication, and division, respectively. The value of the second operand of division must not be zero.

Arithmetic with two integer operands

If both operands of an arithmetic operator are integers, the operation is performed in binary and the result is a large integer. Any remainder of division is lost. The result of an integer arithmetic operation (including unary minus) must be within the range of large integers.

Arithmetic with an integer and a decimal operand

If one operand is an integer and the other is decimal, the operation is performed in decimal using a temporary copy of the integer that has been converted to a decimal number with zero scale and precision as defined in the following table:

Operand	Precision of decimal copy
Column or variable: large integer	11
Column or variable: small integer	5
Constant: more than five digits (including leading zeros)	Same as the number of digits in the constant
Constant: five digits or fewer	5

Arithmetic with two decimal operands

If both operands are decimal, the operation is performed in decimal. The result of any decimal arithmetic operation is a decimal number with a precision and scale that depend on two factors:

The precision and scale of the operands

In the discussion of operations with two decimal operands, the precision and scale of the first operand are denoted by p and s, that of the second operand by p' and s'. Thus, for a division, the dividend has precision p and scale s, and the divisor has precision p' and scale s'.

Whether DEC31 or DEC15 is in effect for the operation

DEC31 and DEC15 specify the rules to be used when both operands in a decimal operation have precisions of 15 or less. DEC15 specifies the rules

which do not allow a precision greater than 15 digits, and DEC31 specifies the rules which allow a precision of up to 31 digits. The rules for DEC31 are always used if either operand has a precision greater than 15.

For static SQL statements, the value of the field DECIMAL ARITHMETIC on installation panel DSNTIP4 or the precompiler option DEC determines whether DEC15 or DEC31 is used.

For dynamic SQL statements, the value of the field DECIMAL ARITHMETIC on installation panel DSNTIP4, the precompiler option DEC, or the special register CURRENT PRECISION determines whether DEC15 or DEC31 is used according to these rules:

- Field DECIMAL ARITHMETIC applies if either of these conditions is true:
 - DYNAMICRULES run behavior applies and the application has not set CURRENT PRECISION.
For a list of the DYNAMICRULES bind option values that specify run, bind, define, or invoke behavior, see Table 2 on page 44.
 - DYNAMICRULES bind, define, or invoke behavior applies; the value of installation panel field USE FOR DYNAMICRULES is YES; and the application has not set CURRENT PRECISION.
- Precompiler option DEC applies if DYNAMICRULES bind, define, or invoke behavior is in effect, the value of installation panel field USE FOR DYNAMICRULES is NO, and the application has not set CURRENT PRECISION.
- Special register CURRENT PRECISION applies if the application sets the register.

The value of DECIMAL ARITHMETIC is the default value for the precompiler option and the special register. SQL statements executed using SPUFI use the value in DECIMAL ARITHMETIC.

Decimal addition and subtraction

If the operation is addition or subtraction and the operands do not have the same scale, the operation is performed with a temporary copy of one of the operands that has been extended with trailing zeros so that its fractional part has the same number of digits as the other operand.

The precision of the result is the minimum of n and the quantity $\text{MAX}(p-s,p'-s')+1$. The scale is $\text{MAX}(s,s')$. n is 31 if DEC31 is in effect or if the precision of at least one operand is greater than 15. Otherwise, n is 15.

Decimal multiplication

For multiplication, the precision of the result is $\text{MIN}(n,p+p')$, and the scale is $\text{MIN}(n,s+s')$. n is 31 if DEC31 is in effect or if the precision of at least one operand is greater than 15. Otherwise, n is 15.

If both operands have a precision greater than 15, the operation is performed using a temporary copy of the operand with the smaller precision. If the operands have the same precision, the second operand is selected. If more than 15 significant digits are needed for the integral part of the copy, the statement's execution is ended and an error occurs. Otherwise, the copy is converted to a number with precision 15, by truncating the copy on the right. The truncated copy has a scale of $\text{MAX}(0,S-(P-15))$, where P and S are the original precision and scale. If, in the process of truncation, one or more nonzero digits are removed, SQLWARN7 in SQLCA is set to W, indicating loss of precision.

Expressions

When both operands have a precision greater than 15, the foregoing formulas for the precision and scale of the result still apply, with one change: for the operand selected as the copy, use the precision and scale of the truncated copy; that is, use 15 as the precision and $\text{MAX}(0, S - (P - 15))$ for the scale.

Let n denote the value of the operand with the greater precision or the first operand in the case of operands with the same precision. The number of leading zeros in a 31-digit representation of n must be greater than the precision of the other operand. This is always the case if the precision of the operand is 15 or less. With greater precisions, overflow can occur even if the precision of the result is less than 31. For example, the expression:

will cause overflow because the number of leading zeros in the 31-digit representation of the large number and the precision of the small number are both 5 (see “Arithmetic with an integer and a decimal operand” on page 114).

Decimal division

The rules for a specific decimal division depend on three factors:

- Whether the DEC31 option is in effect for the operation
 - Whether p is greater than 15
 - Whether p' is greater than 15

The following table shows how the precision and scale of the result depend on these factors. In that table, the occurrence of “N/A” in a row implies that the indicated factor is not relevant to the case represented by the row.

Table 21. Precision (p) and scale (s) of the result of a decimal division

DEC31	p	p'	P	S
Not in effect	≤ 15	≤ 15	15	$15 - (p - s + s')$
In effect	≤ 15	≤ 15	31	$N - (p - s + s')$, where N is $30 - p'$ if p' is odd. N is $29 - p'$ if p' is even.
N/A	> 15	≤ 15	31	$N - (p - s + s')$, where N is $30 - p'$ if p' is odd. N is $29 - p'$ if p' is even.
N/A	N/A	> 15	31	$15 - (p - s + x)$, where x is $\text{MAX}(0, s' - (p' - 15))$ (See Note 2 below)

Notes on decimal division:

1. If the calculated value of 's' is negative, an error occurs. If a minimum divide result scale is specified, this error does not occur. A minimum divide result scale of 3 can be specified using the MINIMUM DIVIDE SCALE field on the installation panel DSNTIPF. A minimum divide scale result between 1 and 9 can be specified using the DECIMAL ARITHMETIC OPTION of the form 'Dpp.s' where 'pp' is 15 or 31 and represents the precision and 's' represents the minimum divide scale, as a number between 1 and 9. Such a specification overrides the MINIMUM DIVIDE SCALE. When a minimum divide result scale is specified, the formula MAX(s,s'), where s represents the scale derived from the above table and s' represents the value specified by the minimum divide result scale, is applied and a new scale is derived. The newly derived scale is the scale of the result and overrides any scale derived using the table above.

2. If p' is greater than 15, the division is performed using a temporary copy of the divisor. If more than 15 significant digits are needed for the integral part of the divisor, the statement's execution is ended, and an error occurs. Otherwise, the copy is converted to a number with precision 15, by truncating the copy on the right. The truncated copy has a scale of $\text{MAX}(0, s' - (p' - 15))$, which is the formula for x that appears in row 4 of Table 21. If, in the process of truncation, one or more nonzero digits are removed, SQLWARN7 in SQLCA is set to W, indicating loss of precision.

Arithmetic with floating-point operands

If either operand of an arithmetic operator is floating-point, the operation is performed in floating-point. If necessary, the operands are first converted to double precision floating-point numbers. Thus, if any element of an expression is a floating-point number, the result of the expression is a double precision floating-point number.

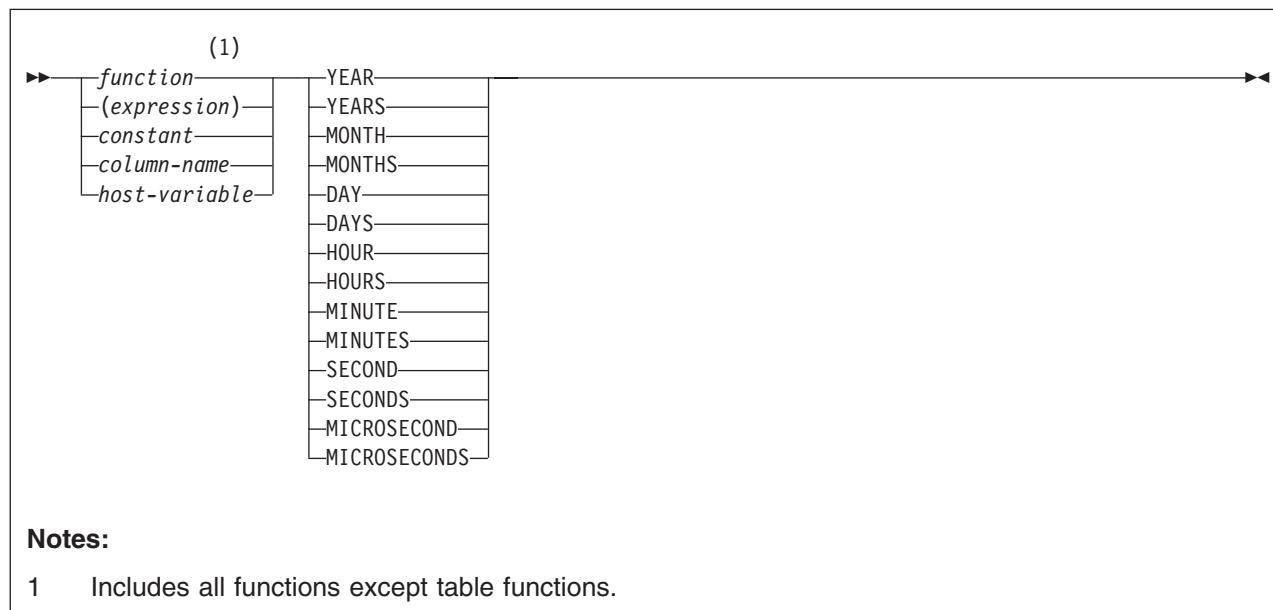
An operation involving a floating-point number and an integer is performed with a temporary copy of the integer that has been converted to double precision floating-point. An operation involving a floating-point number and a decimal number is performed with a temporary copy of the decimal number that has been converted to double precision floating-point. The result of a floating-point operation must be within the range of floating-point numbers.

Datetime operands and durations

Datetime values can be incremented, decremented, and subtracted. These operations may involve decimal numbers called durations. A *duration* is a number representing an interval of time. There are four types of durations:

Labeled durations

The form a labeled duration is as follows:



A *labeled duration* represents a specific unit of time as expressed by a number (which can be the result of an expression) followed by one of the

Expressions

seven duration keywords:¹⁷ YEARS, MONTHS, DAYS, HOURS, MINUTES, SECONDS, or MICROSECONDS. The number specified is converted as if it were assigned to a DECIMAL(15,0) number.

A labeled duration can only be used as an operand of an arithmetic operator, and the other operand must have a data type of DATE, TIME, or TIMESTAMP. Thus, the expression HIREDATE + 2 MONTHS + 14 DAYS is valid, whereas the expression HIREDATE + (2 MONTHS + 14 DAYS) is not. In both of these expressions, the labeled durations are 2 MONTHS and 14 DAYS.

Date duration

A *date duration* represents a number of years, months, and days expressed as a DECIMAL(8,0) number. To be properly interpreted, the number must have the format *yyyymmdd*, where *yyyy* represents the number of years, *mm* the number of months, and *dd* the number of days. The result of subtracting one DATE value from another, as in the expression HIREDATE - BIRTHDATE, is a date duration.

Time duration

A *time duration* represents a number of hours, minutes, and seconds expressed as a DECIMAL(6,0) number. To be properly interpreted, the number must have the format *hhmmss*, where *hh* represents the number of hours, *mm* the number of minutes, and *ss* the number of seconds. The result of subtracting one TIME value from another is a time duration.

Timestamp duration

A *timestamp duration* represents a number of years, months, days, hours, minutes, seconds, and microseconds expressed as a DECIMAL(20,6) number. To be properly interpreted, the number must have the format *yyyxxddhhmmsszzzzz*, where *yyyy*, *xx*, *dd*, *hh*, *mm*, *ss* and *zzzzzz* represent, respectively, the number of years, months, days, hours, minutes, seconds, and microseconds. The result of subtracting one TIMESTAMP value from another is a timestamp duration.

Datetime arithmetic in SQL

The only arithmetic operations that can be performed on datetime values are addition and subtraction. If a datetime value is the operand of addition, the other operand must be a duration. The specific rules governing the use of the addition operator with datetime values follow.

- If one operand is a date, the other operand must be a date duration or labeled duration of years, months, or days.
- If one operand is a time, the other operand must be a time duration or a labeled duration of hours, minutes, or seconds.
- If one operand is a timestamp, the other operand must be a duration. Any type of duration is valid.
- Neither operand of the addition operator can be a parameter marker. For a discussion of parameter markers, see Parameter markers in “PREPARE” on page 846.

The rules for the use of the subtraction operator on datetime values are not the same as those for addition because a datetime value cannot be subtracted from a duration, and because the operation of subtracting two datetime values is not the

17. The singular form of these keywords is also acceptable: YEAR, MONTH, DAY, HOUR, MINUTE, SECOND, and MICROSECOND.

same as the operation of subtracting a duration from a datetime value. The specific rules governing the use of the subtraction operator with datetime values follow.

- If the first operand is a date, the second operand must be a date, a date duration, a string representation of a date, or a labeled duration of years, months, or days.
- If the second operand is a date, the first operand must be a date, or a string representation of a date.
- If the first operand is a time, the second operand must be a time, a time duration, a string representation of a time, or a labeled duration of hours, minutes, or seconds.
- If the second operand is a time, the first operand must be a time, or string representation of a time.
- If the first operand is a timestamp, the second operand must be a timestamp, a string representation of a timestamp, or a duration.
- If the second operand is a timestamp, the first operand must be a timestamp or a string representation of a timestamp.
- Neither operand of the subtraction operator can be a parameter marker.

When an operand in a datetime expression is a string, it may undergo character conversion before it is interpreted and converted to a datetime value. When its CCSID is not that of the default for mixed strings, a mixed string is converted to the default mixed data representation. When its CCSID is not that of the default for SBCS strings, an SBCS string is converted to the default SBCS representation.

Date arithmetic

Dates can be subtracted, incremented, or decremented.

Subtracting dates: The result of subtracting one date (DATE2) from another (DATE1) is a date duration that specifies the number of years, months, and days between the two dates. The data type of the result is DECIMAL(8,0). If DATE1 is greater than or equal to DATE2, DATE2 is subtracted from DATE1. If DATE1 is less than DATE2, however, DATE1 is subtracted from DATE2, and the sign of the result is made negative. The following procedural description clarifies the steps involved in the operation $\text{RESULT} = \text{DATE1} - \text{DATE2}$.

Date subtraction: result = date1 - date2

- If DAY(DATE2) <= DAY(DATE1)
then DAY(RESULT) = DAY(DATE1) - DAY(DATE2).
- If DAY(DATE2) > DAY(DATE1)
then DAY(RESULT) = N + DAY(DATE1) - DAY(DATE2)
where N = the last day of MONTH(DATE2).
MONTH(DATE2) is then incremented by 1.
- If MONTH(DATE2) <= MONTH(DATE1)
then MONTH(RESULT) = MONTH(DATE1) - MONTH(DATE2).
- If MONTH(DATE2) > MONTH(DATE1)
then MONTH(RESULT) = 12 + MONTH(DATE1) - MONTH(DATE2)
and YEAR(DATE2) is incremented by 1.
- YEAR(RESULT) = YEAR(DATE1) - YEAR(DATE2).

For example, the result of DATE('3/15/2000') - '12/31/1999' is 215 (or, a duration of 0 years, 2 months, and 15 days). In this example, notice that the second operand did not need to be converted to a date. According to one of the rules for subtraction, described under "Datetime arithmetic in SQL" on page 118, the second operand can be a string representation of a date if the first operand is a date.

Incrementing and decrementing dates: The result of adding a duration to a date, or of subtracting a duration from a date, is itself a date. (For the purposes of this operation, a month denotes the equivalent of a calendar page. Adding months to a date, then, is like turning the pages of a calendar, starting with the page on which the date appears.) The result must fall between the dates January 1, 0001 and December 31, 9999 inclusive. If a duration of years is added or subtracted, only the year portion of the date is affected. The month is unchanged, as is the day unless the result would be February 29 of a non-leap-year. Here the day portion of the result is set to 28, and the SQLWARN6 field of the SQLCA is set to W, indicating that an end-of-month adjustment was made to correct an invalid date. Part 2 of *DB2 Application Programming and SQL Guide* also describes how SQLWARN6 is set.

Similarly, if a duration of months is added or subtracted, only months and, if necessary, years are affected. The day portion of the date is unchanged unless the result would be invalid (September 31, for example). In this case the day is set to the last day of the month, and the SQLWARN6 field of the SQLCA is set to W to indicate the adjustment.

Adding or subtracting a duration of days will, of course, affect the day portion of the date, and potentially the month and year.

Date durations, whether positive or negative, can also be added to and subtracted from dates. As with labeled durations, the result is a valid date, and SQLWARN6 is set to W to indicate any necessary end-of-month adjustment.

When a positive date duration is added to a date, or a negative date duration is subtracted from a date, the date is incremented by the specified number of years,

months, and days, in that order. Thus, DATE1+X, where X is a positive DECIMAL(8,0) number, is equivalent to the expression:

DATE1 + YEAR(X) YEARS + MONTH(X) MONTHS + DAY(X) DAYS

When a positive date duration is subtracted from a date, or a negative date duration is added to a date, the date is decremented by the specified number of days, months, and years, in that order. Thus, DATE1-X, where X is a positive DECIMAL(8,0) number, is equivalent to the expression:

DATE1 - DAY(X) DAYS - MONTH(X) MONTHS - YEAR(X) YEARS

Adding a month to a date gives the same day one month later unless that day does not exist in the later month. In that case, the day in the result is set to the last day of the later month. For example, January 28 plus one month gives February 28; one month added to January 29, 30, or 31 results in either February 28 or, for a leap year, February 29. If one or more months is added to a given date and then the same number of months is subtracted from the result, the final date is not necessarily the same as the original date.

The order in which labeled date durations are added to and subtracted from dates can affect the results. When you add labeled date durations to a date, specify them in the order of YEARS + MONTHS + DAYS. When you subtract labeled date durations from a date, specify them in the order of DAYS - MONTHS - YEARS. For example, to add one year and one day to a date, specify:

DATE1 + 1 YEAR + 1 DAY

To subtract one year, one month, and one day from a date, specify:

DATE1 - 1 DAY - 1 MONTH - 1 YEAR

Time arithmetic

Times can be subtracted, incremented, or decremented.

Subtracting times: The result of subtracting one time (TIME2) from another (TIME1) is a time duration that specifies the number of hours, minutes, and seconds between the two times. The data type of the result is DECIMAL(6,0). If TIME1 is greater than or equal to TIME2, TIME2 is subtracted from TIME1. If TIME1 is less than TIME2, however, TIME1 is subtracted from TIME2, and the sign of the result is made negative. The following procedural description clarifies the steps involved in the operation RESULT = TIME1 - TIME2.

Time subtraction: result = time1 - time2

- If SECOND(TIME2) <= SECOND(TIME1)
then SECOND(RESULT) = SECOND(TIME1) - SECOND(TIME2).
- If SECOND(TIME2) > SECOND(TIME1)
then SECOND(RESULT) = 60 + SECOND(TIME1) - SECOND(TIME2)
and MINUTE(TIME2) is incremented by 1.
- If MINUTE(TIME2) <= MINUTE(TIME1)
then MINUTE(RESULT) = MINUTE(TIME1) - MINUTE(TIME2).
- If MINUTE(TIME2) > MINUTE(TIME1)
then MINUTE(RESULT) = 60 + MINUTE(TIME1) - MINUTE(TIME2)
and HOUR(TIME2) is incremented by 1.
- HOUR(RESULT) = HOUR(TIME1) - HOUR(TIME2).

For example, the result of TIME('11:02:26') - '00:32:56' is 102930 (a duration of 10 hours, 29 minutes, and 30 seconds). In this example, notice that the second operand did not need to be converted to a time. According to one of the rules for subtraction, described under "Datetime arithmetic in SQL" on page 118, the second operand can be a string representation of a time if the first operand is a time.

Incrementing and decrementing times: The result of adding a duration to a time, or of subtracting a duration from a time, is itself a time. Any overflow or underflow of hours is discarded, thereby ensuring that the result is always a time. If a duration of hours is added or subtracted, only the hours portion of the time is affected. Adding 24 hours to the time '00:00:00' results in the time '24:00:00'. However, adding 24 hours to any other time results in the same time; for example, adding 24 hours to the time '00:00:59' results in the time '00:00:59'. The minutes and seconds are unchanged.

Similarly, if a duration of minutes is added or subtracted, only minutes and, if necessary, hours are affected. The seconds portion of the time is unchanged.

Adding or subtracting a duration of seconds affects the seconds portion of the time and may affect the minutes and hours.

Time durations, whether positive or negative, can also be added to and subtracted from times. The result is a time that has been incremented or decremented by the specified number of hours, minutes, and seconds, in that order. Thus, TIME1 + X, where X is a positive DECIMAL(6,0) number, is equivalent to the expression

TIME1 + HOUR(X) HOURS + MINUTE(X) MINUTES + SECOND(X) SECONDS

Timestamp arithmetic

Timestamps can be subtracted, incremented, or decremented.

Subtracting timestamps: The result of subtracting one timestamp (TS2) from another (TS1) is a timestamp duration that specifies the number of years, months, days, hours, minutes, seconds, and microseconds between the two timestamps. The data type of the result is DECIMAL(20,6). If TS1 is greater than or equal to TS2, TS2 is subtracted from TS1. If TS1 is less than TS2, however, TS1 is

subtracted from TS2 and the sign of the result is made negative. The following procedural description clarifies the steps involved in the operation $\text{RESULT} = \text{TS1} - \text{TS2}$.

Timestamp subtraction: $\text{result} = \text{ts1} - \text{ts2}$

- If $\text{MICROSECOND}(\text{TS2}) \leq \text{MICROSECOND}(\text{TS1})$
then $\text{MICROSECOND}(\text{RESULT}) = \text{MICROSECOND}(\text{TS1}) - \text{MICROSECOND}(\text{TS2})$.
- If $\text{MICROSECOND}(\text{TS2}) > \text{MICROSECOND}(\text{TS1})$
then $\text{MICROSECOND}(\text{RESULT}) = 1000000 + \text{MICROSECOND}(\text{TS1}) - \text{MICROSECOND}(\text{TS2})$
and $\text{SECOND}(\text{TS2})$ is incremented by 1.
- The seconds and minutes part of the timestamps are subtracted as specified in the rules for subtracting times.
- If $\text{HOUR}(\text{TS2}) \leq \text{HOUR}(\text{TS1})$
then $\text{HOUR}(\text{RESULT}) = \text{HOUR}(\text{TS1}) - \text{HOUR}(\text{TS2})$.
- If $\text{HOUR}(\text{TS2}) > \text{HOUR}(\text{TS1})$
then $\text{HOUR}(\text{RESULT}) = 24 + \text{HOUR}(\text{TS1}) - \text{HOUR}(\text{TS2})$
and $\text{DAY}(\text{TS2})$ is incremented by 1.
- The date part of the timestamps is subtracted as specified in the rules for subtracting dates.

Incrementing and decrementing timestamps: The result of adding a duration to a timestamp, or of subtracting a duration from a timestamp, is itself a timestamp. Date and time arithmetic is performed as previously defined, except that an overflow or underflow of hours is carried into the date part of the result, which must be within the range of valid dates. When the result of an operation is midnight, the time portion of the result can be '24.00.00' or '00.00.00'; a comparison of those two values does not result in 'equal'.

Precedence of operations

Expressions within parentheses are evaluated first. When the order of evaluation is not specified by parentheses, prefix operators are applied before multiplication and division, and multiplication, division, and concatenation are applied before addition and subtraction. Operators at the same precedence level are applied from left to right.

Example 1:

1.10 * (SALARY + BONUS) + SALARY / :VAR3

(2) (1) (4) (3)

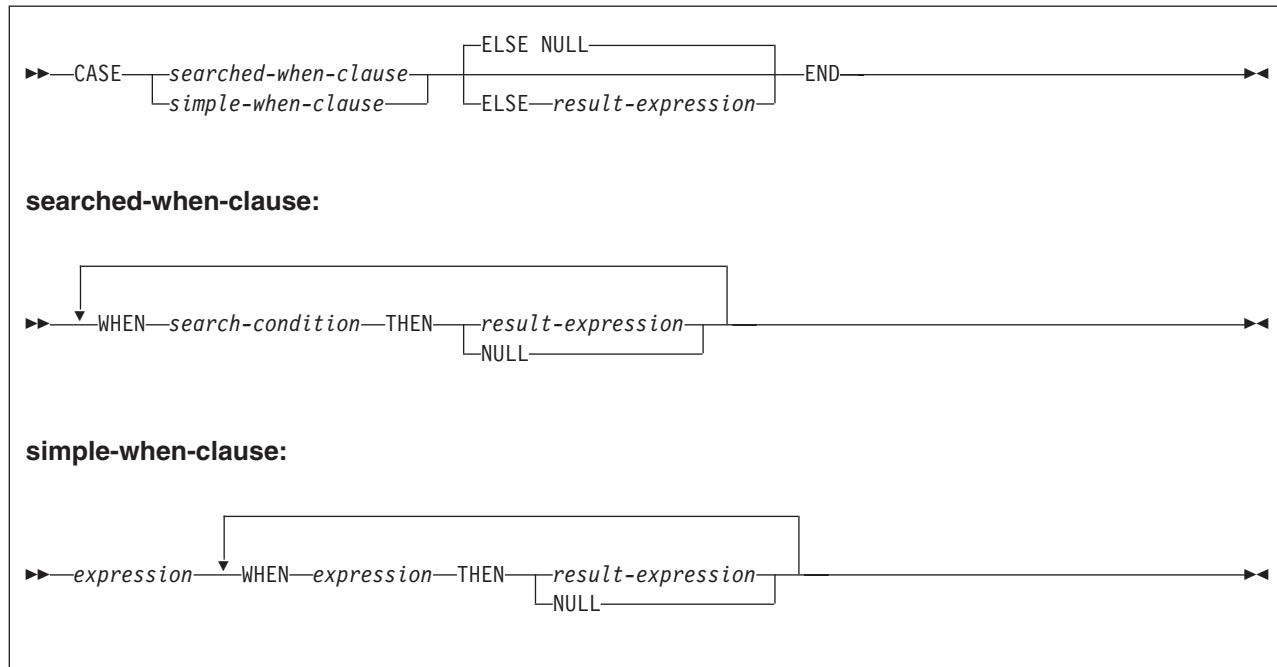
Example 2: In this example, the first operation (CONCAT) combines the character strings in the variables YYYYMM and DD into a string representing a date. The second operation (-) then subtracts that date from the date being processed in DATECOL. The result is a date duration that indicates the time elapsed between the two dates.

Expressions

DATECOL - :YYYYMM CONCAT :DD

(2) (1)

CASE expressions



A CASE expression allows an expression to be selected based on the evaluation of one or more conditions. In general, the value of the case-expression is the value of the *result-expression* following the first (leftmost) case that evaluates to true. If no case evaluates to true and the ELSE keyword is present, the result is the value of the *result-expression* or NULL. If no case evaluates to true and the ELSE keyword is not present, the result is NULL. When a case evaluates to unknown (because of NULLs), the case is NOT true and hence is treated the same way as a case that evaluates to false.

CASE

Begins a *case-expression*.

searched-when-clause

Specifies a *search-condition* that is applied to each row or group of table data presented for evaluation, and the result when that condition is true.

simple-when-clause

Specifies that the value of the *expression* prior to the first WHEN keyword is tested for equality with the value of each *expression* that follows the WHEN keyword. It also specifies the result for when that condition is true.

The data type of the *expression* prior to the first WHEN keyword must be comparable to the data types of each *expression* that follows the WHEN keywords. The data type of any of the expressions cannot be a CLOB, DBCLOB or BLOB. In addition, the *expression* prior to the first WHEN keyword cannot include a function that is nondeterministic or has an external action.

result-expression

Specifies an expression that follows the THEN and ELSE keyword. It specifies the result of a *searched-when-clause* or a *simple-when-clause* that is true, or

the result if no case is true. There must be at least one *result-expression* in the CASE expression with a defined data type. NULL cannot be specified for every case.

All *result-expressions* must be compatible. The attributes of the result are determined according to the rules that are described in “Rules for result data types” on page 77. When the result is a string, its attributes include a CCSID. For the rules on how the CCSID is determined, see “System CCSIDs” on page 23.

search-condition

Specifies a condition that is true, false, or unknown about a row or group of table data. The *search-condition* cannot contain a subselect. If the CASE expression is in a select list or an IN predicate, the search-condition cannot be a quantified predicate, an IN predicate, or an EXISTS predicate.

END

Ends a *case-expression*.

Example 1 (simple-when-clause): Assume that in the EMPLOYEE table the first character of a department number represents the division in the organization. Use a CASE expression to list the full name of the division to which each employee belongs.

```
SELECT EMPNO, LASTNAME,
       CASE SUBSTR(WORKDEPT,1,1)
         WHEN 'A' THEN 'Administration'
         WHEN 'B' THEN 'Human Resources'
         WHEN 'C' THEN 'Design'
         WHEN 'D' THEN 'Operations'
         END
    FROM EMPLOYEE;
```

Example 2 (searched-when-clause): You can also use a CASE expression to avoid “division by zero” errors. From the EMPLOYEE table, find all employees who earn more than 25 percent of their income from commission, but who are not fully paid on commission:

```
SELECT EMPNO, WORKDEPT, SALARY+COMM FROM EMPLOYEE
  WHERE (CASE WHEN SALARY=0 THEN 0
             ELSE COMM/(SALARY+COMM)
             END) > 0.25;
```

Example 3 (searched-when-clause): You can use a CASE expression to avoid “division by zero” errors in another way. The following queries show an accumulation or summing operation. In the first query, DB2 performs the division before performing the CASE statement and an error occurs along with the results.

```
SELECT REF_ID,PAYMT_PAST_DUE_CT,
       CASE
         WHEN PAYMT_PAST_DUE_CT=0 THEN 0
         WHEN PAYMT_PAST_DUE_CT>0 THEN
           SUM(BAL_AMT/PAYMT_PAST_DUE_CT)
         END
    FROM PAY_TABLE
   GROUP BY REF_ID,PAYMT_PAST_DUE_CT;
```

However, if the CASE expression is included in the SUM column function, the CASE expression would prevent the errors. In the following query, the CASE expression screens out the unwanted division because the CASE operation is performed before the division.

Expressions

```
#          SELECT REF_ID,PAYMT_PAST_DUE_CT,
#                  SUM(CASE
#                      WHEN PAYMT_PAST_DUE_CT=0 THEN 0
#                      WHEN PAYMT_PAST_DUE_CT>0 THEN
#                          BAL_AMT/PAYMT_PAST_DUE_CT
#                      END)
#          FROM PAY_TABLE
#         GROUP BY REF_ID,PAYMT_PAST_DUE_CT;
```

Example 4: This example shows how to group the results of a query by a CASE expression without having to re-type the expression. Using the sample employee table, find the maximum, minimum, and average salary. Instead of finding these values for each department, assume that you want to combine some departments into the same group.

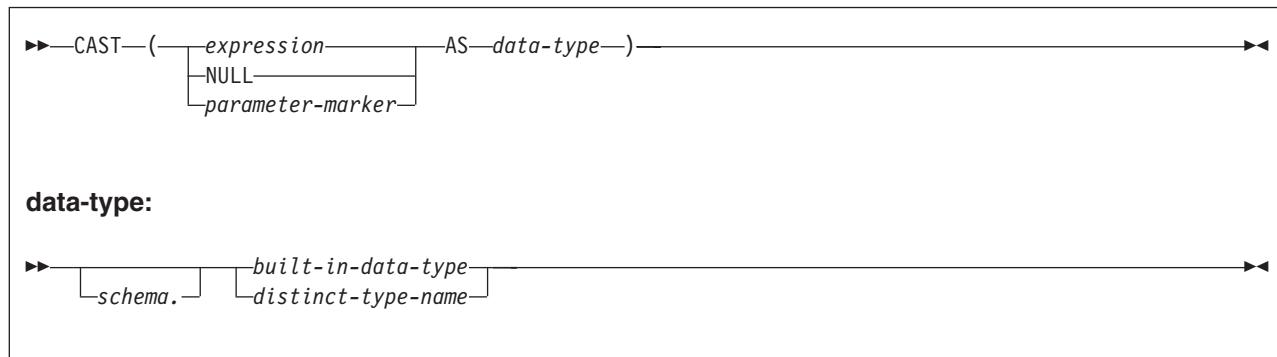
```
SELECT CASE_DEPT,MAX(SALARY),MIN(SALARY),AVG(SALARY)
  FROM (SELECT SALARY,CASE WHEN WORKDEPT = 'A00' OR WORKDEPT = 'E21'
                           THEN 'A00_E21'
                           WHEN WORKDEPT = 'D11' OR WORKDEPT = 'E11'
                           THEN 'D11_E11'
                           ELSE WORKDEPT
                       END AS CASE_DEPT
  FROM DSN8710.EMP) X
 GROUP BY CASE_DEPT;
```

There are two scalar functions, NULLIF and COALESCE, that are specialized to handle a subset of the functionality provided by CASE. Table 22 shows the equivalent expressions using CASE or these functions.

Table 22. Equivalent case expressions

CASE expression	Equivalent expression
CASE WHEN e1=e2 THEN NULL ELSE e1 END	NULLIF(e1,e2)
CASE WHEN e1 IS NOT NULL THEN e1 ELSE e2 END	COALESCE(e1,e2)
CASE WHEN e1 IS NOT NULL THEN e1 ELSE COALESCE(e2,...,eN) END	COALESCE(e1,e2,...,eN)

CAST specification



The CAST specification returns the first operand (the cast operand) converted to the data type that is specified by the second operand. If the data type of either operand is a distinct type, the privilege set must implicitly include EXECUTE authority on the generated cast functions for the distinct type.

expression

Specifies that the cast operand is an expression other than NULL or a parameter marker. The result is the value of the operand value converted to the specified *data type*.

Table 7 on page 63 and Table 8 on page 64 shows the supported casts between data types. If you specify an unsupported cast, an error occurs.

When a character string is cast to a character string with a different length or a graphic string is cast to a graphic string with a different length, a warning occurs if any characters except trailing blanks are truncated. A warning also occurs if any characters are truncated when a BLOB operand is cast.

NULL

Specifies that the cast operand is null. The result is a null value with the specified *data type*.

parameter-marker

A parameter marker, which is normally considered an expression, has a special meaning as a cast operand. When the cast operand is a *parameter-marker*, the *data type* that is specified represents the “promise” that the replacement value for the parameter marker will be assignable to that data type (using “store assignment” rules for strings). Such a parameter marker is considered a *typed parameter marker*. Typed parameter markers are treated like any other typed value for the purpose of function resolution, a DESCRIBE of a select list, or column assignment.

data type

The name of a built-in data type or a distinct type. If a data type has length, precision, or scale attributes, specify the attributes. Only the data type, length, precision, and scale can be specified. (FOR BIT/MIXED/SBCS DATA cannot be specified.) If the attributes are not specified, the default values are used. For example, the default for CHAR is a length of 1, and the default for DECIMAL is a precision of 5 and a scale of 0. For the default values of the other data types, see the description of “built-in-data-type” on page 658 for the CREATE TABLE statement. (For portability across operating systems, when specifying a floating-point data type, use REAL or DOUBLE instead of FLOAT.)

- If the cast operand is *expression*, see “Casting between data types” on page 62 and use any of the target data types that are supported for the data type of the cast operand.
- If the cast operand is NULL, you can use any data type.
- If the cast operand is a *parameter-marker*, you can use any data type. If the data type is a distinct type, the application that uses the parameter marker will use the source data type of the distinct type.

Resolution of cast functions: DB2 uses the schema name and the data type name of the target data type to locate the function to use to convert the first operand to the data type of the second operand. If an unqualified data type name is specified for the second operand, DB2 first resolves the schema name of the data type (by searching the SQL path and selecting the first schema such that the data type exists in the schema and the user has authorization to use the data type). DB2 finds the appropriate cast function when all of the following conditions are true:

- The schema name of the function matches the schema name of the target data type.
- The function name matches the name of the target data type.
- The data type of the expression matches or is *promotable* to the data type of the function’s parameter.

Expressions

This comparison of data types results in one best fit, which is the choice for execution (see “Method of finding the best fit” on page 108). For information on the promotion of data types, see “Promotion of data types” on page 61.

- The user has EXECUTE authority on the function.
- The create timestamp for the function is older than the bind timestamp for the statement in which the CAST specification is used.

If DB2 authorization checking is in effect, and DB2 performs an automatic rebind on a plan or package that contains a CAST specification, any cast functions that were created after the original BIND or REBIND of the invoking plan or package are not candidates for execution.

Result of the CAST: When numeric data is cast to character data, the data type of the result is a fixed-length character string, which is similar to the result that the CHAR function would give. (For more information, see “CHAR” on page 185.) When character data is cast to numeric data, the data type of the result depends on the data type of the specified number. For example, character data that is cast to an integer becomes a large integer, which is similar to the result that the INT function would give. (For more information see “INTEGER or INT” on page 228.)

If the data type of the result is character, the subtype of the result is determined as follows:

- If the *expression* and *data-type* are both character, the subtype of the result is the same as the subtype of *expression*.
- If the field MIXED DATA on installation panel DSNTPF is NO, the subtype of the result is SBCS.
- If the *expression* is a row ID and *data-type* is character, the result has a subtype of FOR BIT DATA, unless the *data-type* is CLOB.
- Otherwise, the subtype of the result is MIXED.

If the data type of the result is a string data type and not character FOR BIT DATA, the encoding scheme of the result is determined as follows:

- If the *expression* and *data-type* are both character, the encoding scheme and CCSID of the result are the same as *expression*. For example, assume CHAR_COL is a character column in the following:
`CAST(CHAR_COL AS VARCHAR(25))`

The result of the CAST is a varying length string with the same encoding scheme and CCSID as the input.

- If the *expression* and *data-type* are both graphic, the encoding scheme and CCSID of the result are the same as *expression*.
- If the result is character, the encoding scheme of the result depends on the value of the field DEF ENCODING SCHEME on installation panel DSNTIPF. The CCSID of the result is the default CCSID for the encoding scheme and subtype of the result.
- If the result is graphic, the encoding scheme of the result depends on the value of the field DEF ENCODING SCHEME on installation panel DSNTIPF. The CCSID of the result is the default CCSID for the encoding scheme of the result.

Alternative syntax for casting distinct types: There is alternative syntax for casting a distinct type to its source data type and vice versa. Assume that a distinct type D_MONEY was defined with the following statement and column MONEY was defined with that data type.

```
CREATE DISTINCT TYPE D_MONEY AS DECIMAL(9,2) WITH COMPARISONS;
```

`DECIMAL(MONEY)` is equivalent syntax to `CAST(MONEY AS DECIMAL(9,2))`. Both forms of the syntax use the cast function that DB2 generated when the distinct type `D_MONEY` was created to convert the distinct type to its source type of `DECIMAL(9,2)`.

However, it is possible that different cast functions may be chosen for the equivalent syntax forms because of the difference in function resolution, particularly the treatment on unqualified names. Although the process of function resolution is similar for both, in the `CAST` specification as described above, DB2 uses the schema name of the target data type to locate the function. Therefore, if an unqualified data type name is specified as the target data type, DB2 uses the SQL path to resolve the schema name of the distinct type and then searches for the function in that schema. In function notation, when an unqualified function name is specified, DB2 searches the schemas in the SQL path to find an appropriate function match, as described under “Function resolution” on page 107. For example, assume that you defined the following distinct types, which implicitly gives you both `USAGE` authority on the distinct types and `EXECUTE` authority on the cast functions that are generated for them:

```
CREATE DISTINCT TYPE SCHEMA1.AGE AS DECIMAL(2,0) WITH COMPARISONS;
  one of the generated cast functions is:
    FUNCTION SCHEMA1.AGE(SYSIBM.DECIMAL(2,0)) RETURNS SCHEMA1.AGE

CREATE DISTINCT TYPE SCHEMA2.AGE AS INTEGER WITH COMPARISONS;
  one of the generated cast functions is:
    FUNCTION SCHEMA2.AGE(SYSIBM.INTEGER) RETURNS SCHEMA2.AGE
```

If `STU_AGE`, an `INTEGER` host variable, is cast to the distinct type with either of the following statements and the SQL path is `SYSIBM`, `SCHEMA1`, `SCHEMA2`:

Syntax 1: `CAST(:STU_AGE AS AGE);`
 Syntax 2: `AGE(:STU_AGE);`

different cast functions are chosen. For syntax 1, DB2 first resolves the schema name of distinct type `AGE` as `SCHEMA1` (the first schema in the path that contains a distinct type named `AGE` for which you have `USAGE` authority). Then it looks for a suitable function in that schema and chooses `SCHEMA1.AGE` because the data type of `STU_AGE`, which is `INTEGER`, is promotable to the data type of the function argument, which is `DECIMAL(2,0)`. For syntax 2, DB2 searches all the schemas in the path for an appropriate function and chooses `SCHEMA2.AGE`. DB2 selects `SCHEMA2.AGE` over `SCHEMA1.AGE` because the data type of its argument (`INTEGER`) is an exact match for `STU_AGE` (`INTEGER`) and, therefore, a better match than the argument for `SCHEMA1.AGE`, which is `DECIMAL(2,0)`.

Example 1: Assume that an application needs only the integer portion of the `SALARY` column, which is defined as `DECIMAL(9,2)` from the `EMPLOYEE` table. The following query for the employee number and the integer value of `SALARY` could be prepared.

```
SELECT EMPNO, CAST(SALARY AS INTEGER) FROM EMPLOYEE;
```

Example 2: Assume that two distinct types exist in schema `SCHEMAX`. Distinct type `D_AGE` was sourced on `SMALLINT` and is the data type for the `AGE` column in the `PERSONNEL` table. Distinct type `D_YEAR` was sourced on `INTEGER` and is the data type for the `RETIRE_YEAR` column in the same table. The following `UPDATE` statement could be prepared.

```
UPDATE PERSONNEL SET RETIRE_YEAR =?
  WHERE AGE = CAST( ? AS SCHEMAX.D_AGE);
```

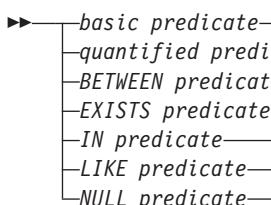
Expressions

The first parameter is an untyped parameter marker that has a data type of RETIRE_YEAR. However, the application will use an integer for the parameter marker. The parameter marker does not need to be cast because the SET is an assignment.

The second parameter marker is a typed parameter marker that is cast to the distinct type D_AGE. Casting the parameter marker satisfies the requirement that comparisons must be performed with compatible data types. The application will use the source data type, SMALLINT, to process the parameter marker.

Predicates

A *predicate* specifies a condition that is true, false, or unknown about a given row or group. The types of predicates are:



The following rules apply to predicates of any type:

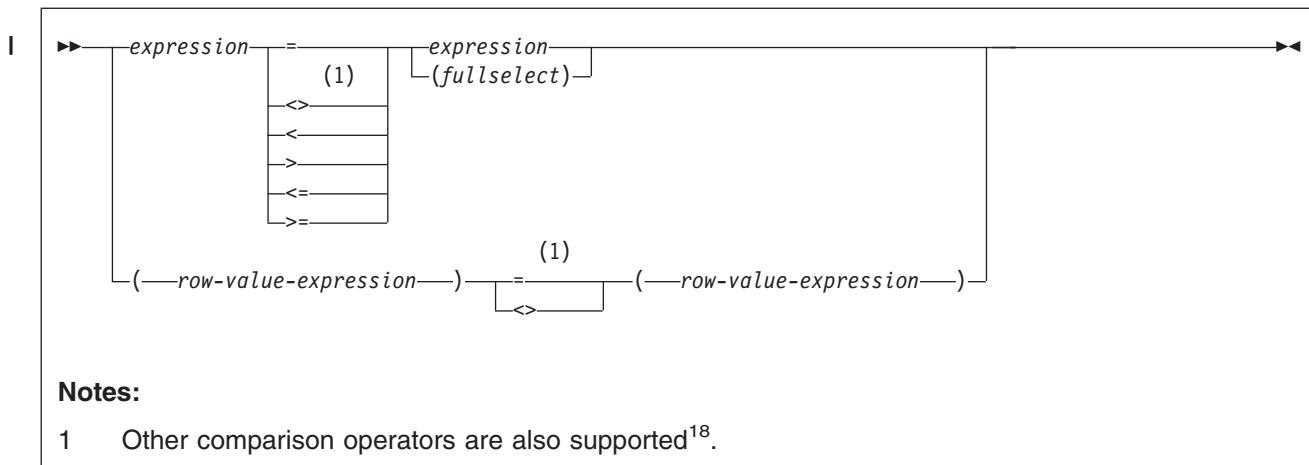
- All values specified in a predicate must be compatible.
- Except for the first operand of LIKE, the operand of a predicate must not be a character string with a maximum length greater than 255 or a graphic string with a maximum length greater than 127.
- Except for EXISTS, a subquery in a predicate must specify a single column.

In addition to the examples of predicates in the following sections, see “Distinct type comparisons” on page 75, which contains several examples of predicates that use distinct types.

Basic predicate

18. The following forms of the comparison operators are also supported in basic and quantified predicates in code pages in which the exclamation point is codepoint X'5A': !=, !<, and !>. In addition, in code pages 437, 819, and 850, the forms ~=, ~<, and ~> are supported. All these product-specific forms of the comparison operators are intended only to support existing SQL statements that use these operators and are not recommended for use when writing new SQL statements.

A not sign (~) or the character that must be used in its place in certain countries, can cause parsing errors in statements passed from one DBMS to another. The problem occurs if the statement undergoes character conversion with certain combinations of source and target CCSIDs. To avoid this problem, substitute an equivalent operator for any operator that includes a not sign. For example, substitute '<>' for '~='; '<=' for '~>', and '>=' for '~<'.



A *basic predicate* compares two values or compares a set of values with another set of values. In the syntax diagram, when *expression* is specified with *fullselect*, the fullselect returns a single result column. If the value of either operand is null or the result of the fullselect is empty, the result of the predicate is unknown. Otherwise, the result is either true or false.

A *row-value-expression* is a set of value expressions. Its form is (*value-expression*, *value expression*, ..., *value expression*). In the syntax diagram for a basic predicate, when a *row-value-expression* is specified on the left side of the = or the <> operator, another *row-value-expression* must be specified on the right side, as in the following two cases:

- If the operator is =, the result is true if all pairs of expressions evaluate to true, and false otherwise.
- If the operator is <>, the result is true if any pair of expressions evaluate to true, and false otherwise.

The use of any other operators or combination of operators when a row value expression appears on the left side of a basic predicate results in an error.

A fullselect in a basic predicate must not return more than one value, whether null or not null.

For values *x* and *y*:

Predicate	Is true if and only if...
<i>x</i> = <i>y</i>	<i>x</i> is equal to <i>y</i>
<i>x</i> <> <i>y</i>	<i>x</i> is not equal to <i>y</i>
<i>x</i> < <i>y</i>	<i>x</i> is less than <i>y</i>
<i>x</i> > <i>y</i>	<i>x</i> is greater than <i>y</i>
<i>x</i> <= <i>y</i>	<i>x</i> is less than or equal to <i>y</i>
<i>x</i> >= <i>y</i>	<i>x</i> is greater than or equal to <i>y</i>

Examples for values *x* and *y*:

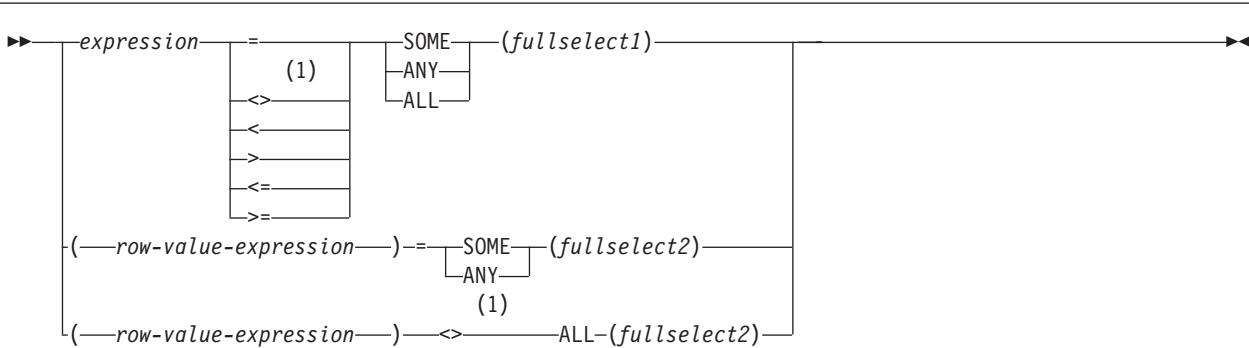
```
EMPNO = '528671'
SALARY < 20000
PRSTAFF <> :VAR1
SALARY >=4 (SELECT AVG(SALARY) FROM DSN8710.EMP)
```

Predicates

Example: List the name, first name, and salary of the employee who is responsible for the 'SECRET' project. This employee may appear in either the PROJA1 or PROJA2 tables. A UNION is used in case the employee appears in both tables to eliminate duplicate RESEMP values.

```
SELECT LASTNAME, FIRSTNAME, SALARY
      FROM DSN8710.EMP X
      WHERE EMPNO = (
      SELECT RESPEMP
            FROM PROJA1 X
            WHERE MAJPROJ = 'SECRET'
      UNION
      SELECT RESPEMP
            FROM PROJA2 X
            WHERE MAJPROJ = 'SECRET');
```

Quantified predicate



Notes:

- 1 Other comparison operators are also supported¹⁸.

A *quantified predicate* compares a value or values with a collection of values. In the syntax diagram, when an *expression* is specified, *fullselect1* returns a single result column and any number of values, whether null or not null. In the diagram, when more than one *row-value-expression* is specified, *fullselect2* returns a number of result columns that is equal to the number of expressions on the left side of = *SOME*, = *ANY*, or <> *ALL*. A *fullselect1* refers to a single column result. A *fullselect2* refers to the number of row-value expressions.

When *ALL* is specified, the result of the predicate is:

- True if the result of the fullselect is empty or if the specified relationship is true for every value returned by the fullselect.
- False if the specified relationship is false for at least one value returned by the fullselect.
- Unknown if the specified relationship is not false for any values returned by the fullselect and at least one comparison is unknown because of a null value.

When *SOME* or *ANY* is specified, the result of the predicate is:

- True if the specified relationship is true for at least one value returned by the fullselect.
- False if the result of the fullselect is empty or if the specified relationship is false for every value returned by the fullselect.

- Unknown if the specified relationship is not true for any of the values returned by the fullselect and at least one comparison is unknown because of a null value.

The use of any other operators or combination of operators when a *row-value-expression* appears on the left side of a quantified predicate results in an error.

Examples: Use the tables below when referring to the following examples. In all examples, “row *n* of TBLA” refers to the row in TBLA for which COLA has the value *n*.

TBLA:	COLA
	1
	2
	3
	4

TBLB:	COLB	COLC
	2	2
	3	--

TBLC:	COLB	COLC
	2	2

Example 1: In the following predicate, the fullselect returns the values 2 and 3. The predicate is false for rows 1, 2, and 3 of TBLA, and is true for row 4.

```
COLA > ALL(SELECT COLB FROM TBLB
UNION
SELECT COLB FROM TBLC)
```

Example 2: In the following predicate, the fullselect returns the values 2 and 3. The predicate is false for rows 1 and 2 of TBLA, and is true for rows 3 and 4.

```
COLA > ANY(SELECT COLB FROM TBLB
UNION
SELECT COLB FROM TBLC)
```

Example 3: In the following predicate, the fullselect returns the values 2 and null. The predicate is false for rows 1 and 2 of TBLA, and is unknown for rows 3 and 4. The result is an empty table.

```
COLA > ALL(SELECT COLC FROM TBLB
UNION
SELECT COLC FROM TBLC)
```

Example 4: In the following predicate, the fullselect returns the values 2 and null. The predicate is unknown for rows 1 and 2 of TBLA, and is true for rows 3 and 4.

```
COLA > SOME(SELECT COLC FROM TBLB
UNION
SELECT COLC FROM TBLC)
```

Example 5: In the following predicate, the fullselect returns an empty result column. Hence, the predicate is true for all rows of TBLA.

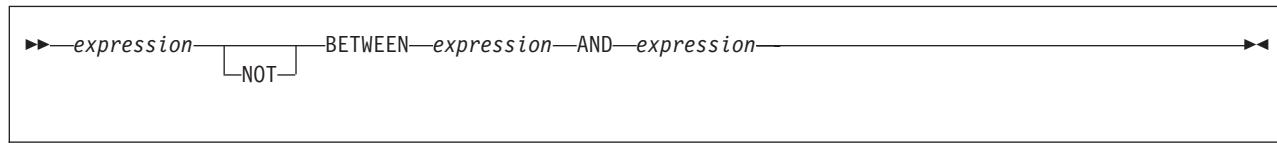
```
COLA < ALL(SELECT COLB FROM TBLB WHERE COLB>3
UNION
SELECT COLB FROM TBLC WHERE COLB>3)
```

Example 6: In the following predicate, the fullselect returns an empty result column. Hence, the predicate is false for all rows of TBLA.

```
COLA < ANY(SELECT COLB FROM TBLB WHERE COLB>3
UNION
SELECT COLB FROM TBLC WHERE COLB>3)
```

If COLA were null in one or more rows of TBLA, the predicate would still be false for all rows of TBLA.

BETWEEN predicate



The BETWEEN predicate determines whether a given value lies between two other given values specified in ascending order. Each of the predicate's two forms has an equivalent search condition, as shown below:

Table 23. BETWEEN predicate and equivalent search conditions

BETWEEN predicate	Equivalent search condition
value1 BETWEEN value2 AND value3	value1 >= value2 AND value1 <= value3
value1 NOT BETWEEN value2 AND value3 or, equivalently: NOT(value1 BETWEEN value2 AND value3)	value1 < value2 OR value1 > value3

Search conditions are discussed in “Search conditions” on page 145.

If the operands include a mixture of datetime values and valid string representations of datetime values, all values are converted to the data type of the datetime operand.

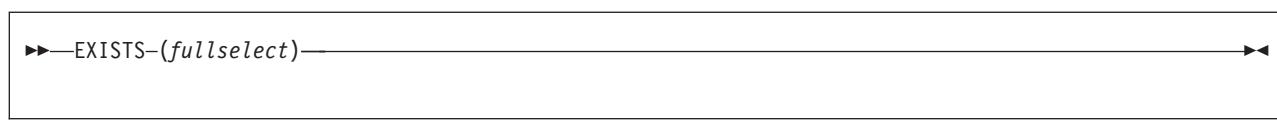
Example: Consider predicate:

A BETWEEN B AND C

The following table shows the value of the predicate for various values of A, B, and C.

Value of A	Value of B	Value of C	Value of predicate
1,2, or 3	1	3	true
0 or 4	1	3	false
0	1	null	false
4	null	3	false
null	any	any	unknown
2	1	null	unknown
3	null	4	unknown

EXISTS predicate



The EXISTS predicate tests for the existence of certain rows.

The result of the predicate is true if the result table returned by the fullselect contains at least one row. Otherwise, the result is false.

The SELECT clause in the fullselect can specify any number of columns because the values returned by the fullselect are ignored. For convenience, use:

```
SELECT *
```

Unlike the NULL, LIKE, and IN predicates, the EXISTS predicate has no form that contains the word NOT. To negate an EXISTS predicate, precede it with the logical operator NOT, as follows:

```
NOT EXISTS (fullselect)
```

The result is then false if the EXISTS predicate is true, and true if the predicate is false. Here, NOT is a logical operator and not a part of the predicate. Logical operators are discussed in “Search conditions” on page 145.

The result cannot be unknown.

Example 1: The following query lists the employee number of everyone represented in DSN8710.EMP who works in a department where at least one employee has a salary less than 20000. Like many EXISTS predicates, the one in this query involves a correlated variable.

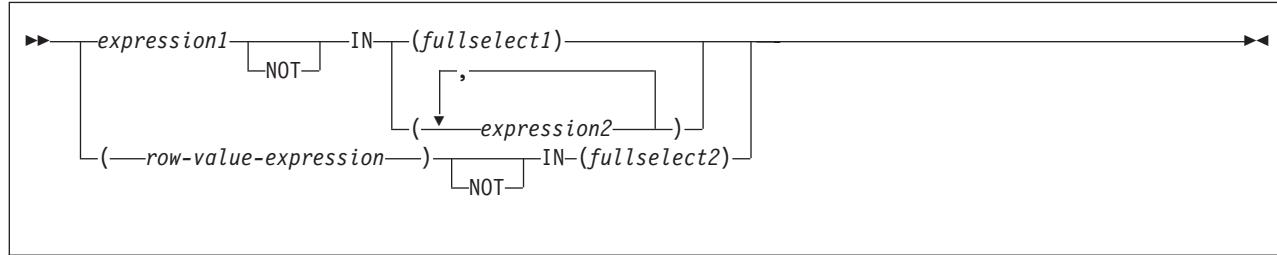
```
SELECT EMPNO
  FROM DSN8710.EMP X
 WHERE EXISTS (SELECT * FROM DSN8710.EMP
                WHERE X.WORKDEPT=WORKDEPT AND SALARY<20000);
```

Example 2: List the subscribers (SNO) in the state of California who made at least one call during the first quarter of 2000. Order the results according to SNO. Each MONTHnn table has columns for SNO, CHARGES, and DATE. The CUST table has columns for SNO and STATE.

```
SELECT C.SNO
  FROM CUST C
 WHERE C.STATE = 'CA'
AND EXISTS (
  SELECT *
    FROM MONTH1
      WHERE DATE BETWEEN '01/01/2000' AND '01/31/2000'
    AND C.SNO = SNO
UNION ALL
  SELECT *
    FROM MONTH2
      WHERE DATE BETWEEN '02/01/2000' AND '02/29/2000'
    AND C.SNO = SNO
UNION ALL
  SELECT *
    FROM MONTH3
      WHERE DATE BETWEEN '03/01/2000' AND '03/31/2000'
    AND C.SNO = SNO
)
ORDER BY C.SNO;
```

Predicates

IN predicate



The IN predicate compares a value with a set of values.

When *expression* and *fullselect1* are used the fullselect must specify a single result column and can return any number of values, whether null or not null. The IN predicate is equivalent to the quantified predicate as follows:

Table 24. IN predicate and equivalent quantified predicates

IN predicate	Equivalent quantified predicate
<i>expression</i> IN (<i>fullselect</i>)	<i>expression</i> = ANY (<i>fullselect</i>)
<i>expression</i> NOT IN (<i>fullselect</i>)	<i>expression</i> <> ALL (<i>fullselect</i>)

When *expression* is used in the non-fullselect form of the IN predicate, the second operand is a set of one or more values specified by any combination of expressions. The values for *expression1* and *expression2* or the column of *fullselect1* in the IN predicate must be compatible. Each expression in *row-value-expression* and its corresponding column of *fullselect2* must be compatible. To determine the attributes of the result type used in the comparison, see “Rules for result data types” on page 77. For information on the types of expressions, see “Expressions” on page 111. If *expression* is a single host variable, the host variable can identify a structure. Any host variable or structure that is specified must be described in the application program according to the rules for declaring host structures and variables. An IN predicate of the form:

expression IN (*value1*, *value2*, ..., *valuen*)

is logically equivalent to:

expression IN (SELECT * FROM R)

when T is a table with a single row and R is a result table formed by the following fullselect:

```
SELECT value1 FROM T
  UNION
SELECT value2 FROM T
  UNION
  .
  .
  .
  UNION
SELECT valuen FROM T
```

When *row-value-expression* is used, *fullselect2* returns a number of result columns that is equal to the number of expressions on the left side of the IN predicate.

If the operands of the IN predicate are strings with different CCSIDs, the rules used to determine which operands are converted are those for operations that combine strings. See “String comparisons” on page 73.

Example 1: The following predicate is true for any row whose employee is in department D11, B01, or C01.

```
WORKDEPT IN ('D11', 'B01', 'C01')
```

Example 2: The following predicate is true for any row whose employee works in department E11.

```
EMPNO IN (SELECT EMPNO FROM DSN8710.EMP  
WHERE WORKDEPT = 'E11')
```

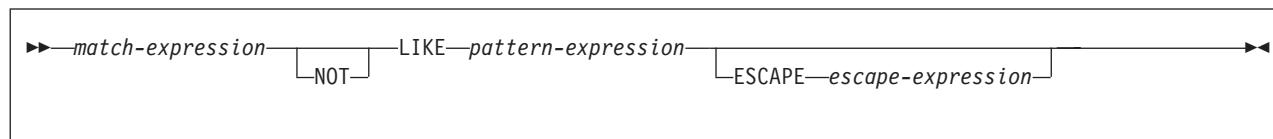
Example 3: The following predicate is true if the date that a project is estimated to start (PRENDATE) is within the next two years.

```
YEAR(PRENDATE) IN (YEAR(CURRENT DATE),  
YEAR(CURRENT DATE + 1 YEAR),  
YEAR(CURRENT DATE + 2 YEARS))
```

Example 4: The following example obtains the phone number of an employee in DSN8710.EMP where the employee number (EMPNO) is a value specified within the COBOL structure defined below.

```
77 PHNUM PIC X(6).  
01 EMPNO-STRUCTURE.  
    05 CHAR-ELEMENT-1 PIC X(6) VALUE '000140'.  
    05 CHAR-ELEMENT-2 PIC X(6) VALUE '000340'.  
    05 CHAR-ELEMENT-3 PIC X(6) VALUE '000220'.  
    .  
    .  
    .  
EXEC SQL DECLARE PHCURS CURSOR FOR  
    SELECT PHONENO FROM DSN8710.EMP  
    WHERE EMPNO IN  
        (:EMPNO-STRUCTURE.CHAR-ELEMENT-1,  
        :EMPNO-STRUCTURE.CHAR-ELEMENT-2,  
        :EMPNO-STRUCTURE.CHAR-ELEMENT-3)  
END-EXEC.  
EXEC SQL OPEN PHCURS  
END-EXEC.  
EXEC SQL FETCH PHCURS INTO :PHNUM  
END-EXEC.
```

LIKE predicate



The LIKE predicate searches for strings that have a certain pattern. The *match-expression* is the string to be tested for conformity to the pattern specified in *pattern-expression*. Underscore and percent sign characters in the pattern have a special meaning instead of their literal meanings unless *escape-expression* is specified, as discussed under the description of *pattern-expression*.

The following rules summarize how a predicate in the form of “*m* LIKE *p*” is evaluated:

Predicates

- If m or p is null, the result of the predicate is unknown.
- If m and p are both empty, the result of the predicate is true.
- If m is empty and p is not, the result of the predicate is unknown unless p consists of one or more percent signs.
- If m is not empty and p is empty, the result of the predicate is false.
- Otherwise, if m matches the pattern in p , the result of the predicate is true. The description of *pattern-expression* provides a detailed explanation on how the pattern is matched to evaluate the predicate to true or false. See the “rigorous description of the pattern” for this information.

The values for *match-expression*, *pattern-expression*, and *escape-expression* must all be character or graphic strings or a mixture of both or they must all be binary strings (BLOBs). None of the expressions can yield a distinct type; however, an expression can be a function that casts a distinct type to its source type.

There are slight differences in what expressions are supported for each argument. The description of each argument lists the supported expressions:

match-expression

An expression that specifies the string to be tested for conformity to a certain pattern of characters.

LIKE *pattern-expression*

An expression that specifies the pattern of characters to be matched.

The expression can be specified by any one of the following:

- A constant
- A special register
- A host variable (including a LOB locator variable)
- A scalar function whose arguments are any of the above (though nested function invocations cannot be used)
- A CAST specification whose arguments are any of the above
- An expression that concatenates (using CONCAT or ||) any of the above

The expression must also meet these restrictions:

- The maximum length of *pattern-expression* must not be larger than 4000 bytes.
- If a host variable is used in *pattern-expression*, the host variable must be defined in accordance with the rules for declaring string host variables and must not be a structure.
- If *escape-expression* is specified, *pattern-expression* must not contain the escape character identified by *escape-expression* except when immediately followed by the escape character, '%', or '_'. For example, if '+' is the escape character, any occurrences of '+' other than '++', '+_ ', or '+%' in the pattern is an error.

When the pattern specified in a LIKE predicate is a parameter marker and a fixed-length character host variable is used to replace the parameter marker, specify a value for the host variable that is the correct length. If you do not specify the correct length, the select does not return the intended results. For example, if the host variable is defined as CHAR(10) and the value WYSE% is assigned to that host variable, the host variable is padded with blanks on assignment. The pattern used is 'WYSE%', which requests DB2 to search for all values that start with WYSE and end with five blank spaces. If you intended to search for only the values that start with 'WYSE,' you should assign the value 'WYSE%%%%%%%%%' to the host variable.

If the pattern is specified in a fixed-length string variable, any trailing blanks are interpreted as part of the pattern. Therefore, it is better to use a varying-length string variable with an actual length that is the same as the length of the pattern. If the host language does not allow varying-length string variables, place the pattern in a fixed-length string variable whose length is the length of the pattern.

For more on the use of host variables with specific programming languages, see Part 2 of *DB2 Application Programming and SQL Guide*.

The pattern is used to specify the conformance criteria for values in the *match-expression* where:

- The underscore character (_) represents any single character.
- The percent sign (%) represents a string of zero or more characters.
- Any other character represents a single occurrence of itself.

If the *pattern-expression* needs to include either the underscore or the percent character, the *escape-expression* is used to specify a character to precede either the underscore or percent character in the pattern. For character strings, the terms *character*, *percent sign*, and *underscore* refer to SBCS characters. For graphic strings, the terms refer to double-byte or UTF-16 characters.

A rigorous description of the pattern

This more rigorous description of the pattern ignores the use of the *escape-expression*.

Let m denote the value of *match-expression* and let p denote the value of *pattern-expression*. The string p is interpreted as a sequence of the minimum number of substring specifiers so each character of p is part of exactly one substring specifier. A substring specifier is an underscore, a percent sign, or any non-empty sequence of characters other than an underscore or a percent sign.

The result of the predicate is unknown if m or p is the null value. Otherwise, the result is either true or false. The result is true if m and p are both empty strings or there exists a partitioning of m into substrings such that:

- A substring of m is a sequence of zero or more contiguous characters and each character of m is part of exactly one substring.
- If the n th substring specifier is an underscore, the n th substring of m is any single character.
- If the n th substring specifier is a percent sign, the n th substring of m is any sequence of zero or more characters.
- If the n th substring specifier is neither an underscore nor a percent sign, the n th substring of m is equal to that substring specifier and has the same length as that substring specifier.
- The number of substrings of m is the same as the number of substring specifiers.

It follows that if p is an empty string and m is not an empty string, the result is false. Similarly, if m is an empty string and p is not an empty string, the result is false.

The predicate m NOT LIKE p is equivalent to the search condition NOT (m LIKE p).

Mixed data patterns: If *match-expression* represents mixed data, the pattern is assumed to be mixed data. For ASCII and EBCDIC, the special characters in the pattern are interpreted as follows:

- An SBCS underscore refers to one SBCS character.
- A DBCS underscore refers to one MBCS character.
- A percent sign (either SBCS or DBCS) refers to a string of zero or more SBCS or MBCS characters.
- Redundant shift bytes in *match-expression* or *pattern-expression* are ignored.

For Unicode, the special characters in the pattern are interpreted as follows:

- An SBCS or DBCS underscore refers to one character (either SBCS or MBCS).
- A percent sign (either SBCS or DBCS) refers to a string of zero or more SBCS or MBCS characters.

When the LIKE predicate is used with Unicode data, the Unicode percent sign and underscore use the code points indicated in the following table:

Character	UTF-8	UTF-16
Half-width %	X'25'	X'0025'
Full-width %	X'EFBC85'	X'FF05'
Half-width _	X'5F'	X'005F'
Full-width _	X'EFBCBF'	X'FF3F'

The full-width or half-width % matches zero or more characters. The full-width or half width _ character matches exactly one character. (For ASCII or EBCDIC data, a full-width _ character matches one DBCS character.)

ESCAPE *escape-expression*

An expression that specifies the escape character to be used to modify the special meaning of the underscore (_) and percent (%) characters in *pattern-expression*. Specifying an expression, which is optional, allows values that contain the actual percent and underscore characters to be matched. The escape character consists of a single SBCS (1 byte) or DBCS (2 bytes) character. An escape clause is allowed for Unicode mixed (UTF-8) data, but is restricted for ASCII and EBCDIC mixed data.

The expression can be specified by any one of:

- A constant
- A host variable (including a LOB locator variable)
- A scalar function whose arguments are any of the above (though nested function invocations cannot be used)
- A CAST specification whose arguments are any of the above

The following rules also apply to the use of the ESCAPE clause and *escape-expression*:

- The result of *escape-expression* must be one SBCS or DBCS character or a binary string that contains exactly 1 byte.
- The ESCAPE clause cannot be used if *match-expression* is mixed data.
- If *escape-expression* is specified by a host variable, the host variable must be defined in accordance with the rules for declaring fixed-length string host variables.¹⁹ If the host variable has a negative indicator variable, the result of the predicate is unknown.
- The pattern must not contain the escape character except when followed by the escape character, '%' or '_'. For example, if '+' is the escape character, any occurrences of '+' other than '++', '+_', or '+%' in the pattern is an error.

The following example shows the effect of successive occurrences of the escape character, which in this case is the plus sign (+).

When the pattern string is... The actual pattern is...

+	A percent sign
++%	A plus sign followed by zero or more arbitrary characters
+++%	A plus sign followed by a percent sign

19. If it is NUL-terminated, a C character string variable of length 2 can be specified.

Predicates

Examples

Example 1: The following predicate is true when the string to be tested in NAME has the value SMITH, NESMITH, SMITHSON, or NESMITHY. It is not true when the string has the value SMYTHe:

```
NAME LIKE '%SMITH%'
```

Example 2: In the predicate below, a host variable named PATTERN holds the string for the pattern:

```
NAME LIKE :PATTERN ESCAPE '+'
```

Assume that the string in PATTERN has the value:

```
AB+_C_%
```

Observe that in this string, the plus sign preceding the first underscore is an escape character. The predicate is true when the string being tested in NAME has the value AB_CD or AB_CDE. It is false when this string has the value AB, AB_, or AB_C.

Example 3: The following two predicates are equivalent; three of the four percent signs in the first predicate are redundant.

```
NAME LIKE 'AB%%%CD'  
NAME LIKE 'AB%CD'
```

Example 4: Assume that a distinct type named ZIP_TYPE with a source data type of CHAR(5) exists and an ADDRZIP column with data type ZIP_TYPE exists in some table TABLEY. The following statement selects the row if the zip code (ADDRZIP) begins with '9555'.

```
SELECT * FROM TABLEY  
WHERE CHAR(ADDRZIP) LIKE '9555%';
```

Example 5: The RESUME column in sample table DSN8710.EMP_PHOTO_RESUME is defined as a CLOB. The following statement selects the RESUME column when the string JONES appears anywhere in the column.

```
SELECT RESUME FROM DSN8710.EMP_PHOTO_RESUME  
WHERE RESUME LIKE '%JONES%';
```

Example 6: In the following table, assume COL1 is a column that contains mixed EBCDIC data. The table shows the results when the predicate in the first column is evaluated using the COL1 value in the second column:

Predicates	COL1 Values	Result
WHERE COL1 LIKE 'aaa A B% C '	'aaa A BDZC'	True
WHERE COL1 LIKE 'aaa A B% C '	'aaa A B dzx C '	True
WHERE COL1 LIKE 'a% C '	'a C ' 'ax C ' 'ab D E fg C '	True True True
WHERE COL1 LIKE 'a_ C '	'a% C ' 'a X C'	True False
WHERE COL1 LIKE 'a_ C '	'a X C' 'ax C '	True False
WHERE COL1 LIKE '_ C '	Empty string	True
WHERE COL1 LIKE 'ab C '	'ab C , d' 'ab C , d'	True True

Example 7: In the following table, assume COL1 is a column that contains mixed ASCII data. The table shows the results when the predicate in the first column is evaluated using the COL1 value in the second column:

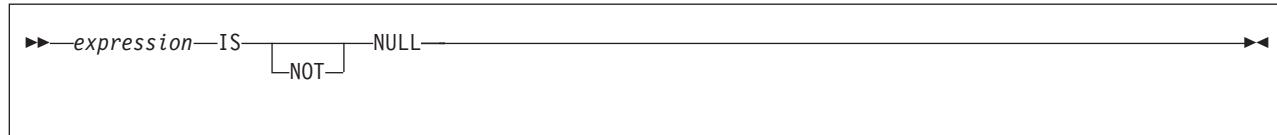
Predicates	COL1 Values	Result
WHERE COL1 LIKE 'aaa A B% C '	'aaa A BDZC'	True
WHERE COL1 LIKE 'aaa A B% C '	'aaa A B dzx C '	True

Example 8: In the following table, assume COL1 is a column that contains Unicode data. The table shows the results when the predicate in the first column is evaluated using the COL1 value in the second column:

Predicates

Predicates	COL1 Values	Result
WHERE COL1 LIKE 'aaa ₀ A B% C '	'aaa ₀ A DZC' 'aaa ₀ A B ₁ dzx ₀ C ' Empty string	True True False
WHERE COL1 LIKE 'aaa ₀ A B ₁ % ₀ C '	'aaa ₀ A DZC' 'aaa ₀ A B ₁ dzx ₀ C ' Empty string	True True False
WHERE COL1 LIKE ' ₀ A DZC'	'aaa ₀ A DZC' 'aaa ₀ A B ₁ dzx ₀ C ' Empty string	False False True
WHERE COL1 LIKE ' ₀ A DZC ₁ '	'aaa ₀ A DZC' 'aaa ₀ A B ₁ dzx ₀ C ' Empty string	True True False
WHERE COL1 LIKE ' ₀ A DZC ₁ ₀ C '	'aaa ₀ A DZC' 'aaa ₀ A B ₁ dzx ₀ C ' Empty string	True True False
WHERE COL1 LIKE ' ₀ A DZC ₁ ₀ C ₁ '	'aaa ₀ A DZC' 'aaa ₀ A B ₁ dzx ₀ C ' Empty string	False False False

NULL predicate



The NULL predicate tests for null values.

The result of a NULL predicate cannot be unknown. If the value of the expression is null, the result is true. If the value is not null, the result is false. If NOT is specified, the result is reversed.

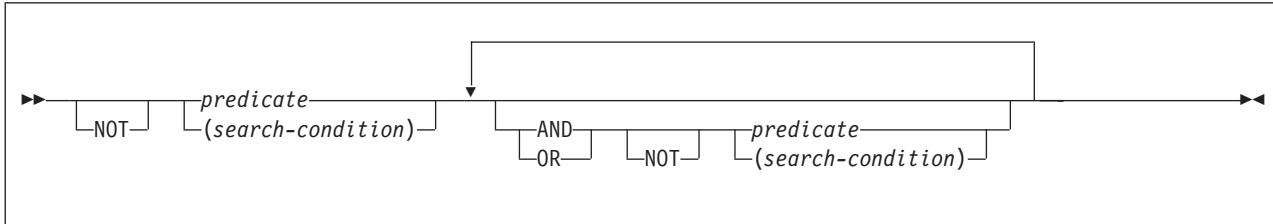
A parameter marker must not be specified for or within the expression.

Example: The following predicate is true whenever PHONENO has the null value, and is false otherwise.

PHONENO IS NULL

Search conditions

A *search condition* specifies a condition that is true, false, or unknown about a given row or group. When the condition is true, the row or group qualifies for the results. When the condition is false or unknown, the row or group does not qualify.



The result of a search condition is derived by application of the specified *logical operators* (AND, OR, NOT) to the result of each specified predicate. If logical operators are not specified, the result of the search condition is the result of the specified predicate.

AND and OR are defined in the following table, in which P and Q are any predicates:

Table 25. Truth table for AND and OR

P	Q	P AND Q	P OR Q
True	True	True	True
True	False	False	True
True	Unknown	Unknown	True
False	True	False	True
False	False	False	False
False	Unknown	False	Unknown
Unknown	True	Unknown	True
Unknown	False	False	Unknown
Unknown	Unknown	Unknown	Unknown

NOT(true) is false and NOT(false) is true, but NOT(unknown) is still unknown. The NOT logical operator has no affect on an unknown condition. The result of NOT(unknown) is still unknown.

Search conditions within parentheses are evaluated first. If the order of evaluation is not specified by parentheses, NOT is applied before AND, and AND is applied before OR. The order in which operators at the same precedence level are evaluated is undefined to allow for optimization of search conditions.

Example 1: In the first of the search conditions below, AND is applied before OR. In the second, OR is applied before AND.

```
SALARY>:SS AND COMM>:CC OR BONUS>:BB
SALARY>:SS AND (COMM>:CC OR BONUS>:BB)
```

Example 2: In the first of the search conditions below, NOT is applied before AND. In the second, AND is applied before NOT.

```
NOT SALARY>:SS AND COMM>:CC
NOT (SALARY>:SS AND COMM>:CC)
```

Search conditions

Example 3: For the following search condition, AND is applied first. After the application of AND, the ORs could be applied in either order without changing the result. DB2 can therefore select the order of applying the ORs.

SALARY>:SS AND COMM>:CC OR BONUS>:BB OR SEX=:GG

Options affecting SQL

Certain DB2 precompiler options, DB2 subsystem parameters (set through the installation panels), bind options, and special registers affect how SQL statements can be composed or determine how SQL statements are processed.

Table 26 summarizes the effect of these options and shows where to find more information. (Some of the items are described in detail following the table, while other items are described elsewhere.)

Table 26. Summary of items affecting composition and processing of SQL statements

Precompiler option	Other ¹	Affects
	DYNAMICRULES bind option	The rules that DB2 applies to dynamic SQL statements. For details about authorization, see “Authorization IDs and dynamic SQL” on page 43. The bind option can also affect decimal point representation, string delimiters, mixed data, and decimal arithmetic. For details about how DB2 applies the precompiler options to dynamic SQL statements when DYNAMICRULES bind, define, or invoke behavior is in effect, see “Precompiler options for dynamic statements” on page 148.
	USE FOR DYNAMICRULES	Use of precompiler options for dynamic statements when DYNAMICRULES bind, define, or, invoke behavior is in effect. For details, see “Precompiler options for dynamic statements” on page 148.
COMMA PERIOD	DECIMAL POINT IS	Representation of decimal points in SQL statements. For details, see page 148.
APOSTSQL QUOTESQL	SQL STRING DELIMITER	Representation of string delimiters in SQL statements. For details, see page 149.
	ASCII CODED CHAR SET	A numeric value that determines the CCSID of ASCII string data. For details, see page 150.
	EBCDIC CODED CHAR SET	A numeric value that determines the CCSID of EBCDIC string data and whether Katakana characters can be used in ordinary identifiers. For details, see page 150.
	UNICODE CCSID	A numeric value that determines the CCSID of Unicode string data. For details, see page 150.
GRAPHIC NOGRAPHIC	MIXED DATA	Use of ASCII or EBCDIC character strings with a mixture of SBCS and DBCS characters. For details, see page 150.

Table 26. Summary of items affecting composition and processing of SQL statements (continued)

Precompiler option	Other ¹	Affects
DATE TIME	DATE FORMAT TIME FORMAT LOCAL DATE LENGTH LOCAL TIME LENGTH	Formatting of datetime strings. For details, see page 151.
STDSQL		Conformance with the SQL standard. For details, see page 151.
NOFOR or STDSQL		Whether the FOR UPDATE clause must be specified (in the SELECT statement of the DECLARE CURSOR statement). For details, see page 153.
CONNECT		Whether the rules for a type 1 or a type 2 CONNECT statement apply. See "CONNECT" on page 501 for a description of the rules.
	CURRENTSERVER bind option	Establishing a server other than the local DB2 subsystem. For details, see "Establishing a different server" on page 503.
	SQLRULES bind option	Whether a type 2 CONNECT statement is processed with DB2 rules or SQL standard rules.
	CURRENT RULES special register	Whether the statements ALTER TABLE, CREATE TABLE, GRANT, and REVOKE are processed with DB2 rules or SQL standard rules. For details, see "CURRENT RULES" on page 90. Whether DB2 automatically creates the LOB table space, auxiliary table, and index on the auxiliary table for a LOB column in a base table. For details, see "Creating a table with LOB columns" on page 675. Whether DB2 automatically creates an index on a ROWID column that is defined with GENERATED BY DEFAULT. For details, see the description of the clause for "CREATE TABLE" on page 653. Whether a stored procedure runs as a main or subprogram. For details, see "CREATE PROCEDURE (external)" on page 617.
	SQLRULES bind option or CURRENT RULES special register	Whether SQLCODE +236 is issued when the SQLDA provided on DESCRIBE or PREPARE INTO is too small and the result columns do not involve LOBs or distinct types. For details, see "DESCRIBE (prepared statement or table)" on page 749 and Appendix C, "SQLCA and SQLDA," on page 979. Whether the SELECT privilege is required in a searched DELETE or UPDATE. For details, see "DELETE" on page 742 or "UPDATE" on page 928.

Options affecting SQL

Table 26. Summary of items affecting composition and processing of SQL statements (continued)

Precompiler option	Other ¹	Affects
DEC	DECIMAL ARITHMETIC or CURRENT PRECISION special register	Whether DEC15 or DEC31 rules are used when both operands in a decimal operation have 15 digits or less. For details, see “Arithmetic with two decimal operands” on page 114.

Note: ¹The entries in this column are fields on installation panels unless otherwise noted.

For further details on precompiler options, see Part 5 of *DB2 Application Programming and SQL Guide*. For more details on bind options, see Chapter 2 of *DB2 Command Reference*.

Precompiler options for dynamic statements

Generally, dynamic statements use the application programming defaults specified on installation panel DSNTIPF. However, if the value of installation panel field USE FOR DYNAMICRULES is NO and DYNAMICRULES bind, define, or invoke behavior is in effect, the following precompiler options are used instead of the application programming defaults:

- COMMA or PERIOD
- APOST or QUOTE
- APOSTSQL or QUOTESQL
- GRAPHIC or NOGRAPHIC
- DEC(15) or DEC(31)

For some languages, the precompiler option defaults to a value and no alternative is allowed. If the value of installation panel field USE FOR DYNAMICRULES is YES, dynamic statements use the application programming defaults regardless of the value of bind option DYNAMICRULES.

For additional information on the effect of precompiler options and application programming defaults on:

- Decimal point representation, see page 148.
- String delimiters, see page 149.
- Mixed data, see page 150.
- Decimal arithmetic, see “Arithmetic with two decimal operands” on page 114.

For a list of the DYNAMICRULES bind option values that specify run, bind, define, or invoke behavior, see Table 2 on page 44.

Decimal point representation

Decimal points in SQL statements are represented with either periods or commas. Two values control the representation:

- The value of field DECIMAL POINT IS on installation panel DSNTIPF, which can be a comma (,) or period (.)
- COMMA or PERIOD, which are mutually exclusive DB2 precompiler options for COBOL

These values apply to SQL statements as follows:

- For a distributed operation, the decimal point is the first of the following values that applies:
 - The decimal point value specified by the requester

- The value of field DECIMAL POINT IS on panel DSNTIPF at the DB2 where the package is bound
- Otherwise:
 - For static SQL statements:
 - In a COBOL program, the DB2 precompiler option COMMA or PERIOD determines the decimal point representation for every static SQL statement. If neither precompiler option is specified, the value of DECIMAL POINT IS at precompilation time determines the representation.
 - In non-COBOL programs, the decimal representation for static SQL statements is always the period.
 - For dynamic SQL statements:
 - If DYNAMICRULES run behavior applies, the decimal point is the value of field DECIMAL POINT IS on installation panel DSNTIPF at the local DB2 when the statement is prepared.
 - For a list of the DYNAMICRULES bind option values that specify run, bind, define, or invoke behavior, see Table 2 on page 44.
 - If DYNAMICRULES bind, define, or invoke behavior applies, and the value of install panel field USE FOR DYNAMICRULES is YES, the decimal point is the value of field DECIMAL POINT IS.
 - If bind, define, or invoke behavior applies, and field USE FOR DYNAMIC RULES is NO, the precompiler option determines the decimal point representation. For COBOL programs, which supports precompiler option COMMA or PERIOD, the decimal point representation is determined as described above for static SQL statements in COBOL programs. For programs written in other host languages, the default precompiler option, which can only be PERIOD, is used.

If the comma is the decimal point, these rules apply:

- In any context, a comma intended as a separator must be followed by a space. Such commas could appear, for example, in a VALUES clause, an IN predicate, or an ORDER BY clause in which numbers are used to identify columns.
- In any context, a comma intended as a decimal point must not be followed by a space.
- If the DECIMAL POINT IS field (and not the precompiler option) determines the comma as the decimal point, DB2 will recognize either a comma or a period as the decimal point in numbers in dynamic SQL.

Apostrophes and quotation marks in string delimiters

The following precompiler options control the representation of string delimiters:

- APOST and QUOTE are mutually exclusive DB2 precompiler options for COBOL. Their meanings are exactly what they are for the COBOL compilers:
 - APOST names the apostrophe (') as the string delimiter in COBOL statements.
 - QUOTE names the quotation mark ("") as the string delimiter.
- Neither option applies to SQL syntax. Do not confuse them with the APOSTSQL and QUOTESQL options.
- APOSTSQL and QUOTESQL are mutually exclusive DB2 precompiler options for COBOL. Their meanings are:
 - APOSTSQL names the apostrophe (') as the string delimiter and the quotation mark ("") as the escape character in SQL statements.
 - QUOTESQL names the quotation mark ("") as the string delimiter and the apostrophe (') as the escape character in SQL statements.

Options affecting SQL

These values apply to SQL statements as follows:

- For a distributed operation, the string delimiter is the first of the following values that applies:
 - The SQL string delimiter value specified by the requester
 - The value of the field SQL STRING DELIMITER on installation panel DSNTIPF at the DB2 where the package is bound
- Otherwise:
 - For static SQL statements:

In a COBOL program, the DB2 precompiler option APOSTSQL or QUOTESQL determines the string delimiter and escape character. If neither precompiler option is specified, the value of field SQL STRING DELIMITER on installation panel DSNTIPF determines the string delimiter and escape character.

In a non-COBOL program, the string delimiter is the apostrophe, and the escape character is the quotation mark.
 - For dynamic SQL statements:
 - If DYNAMICRULES run behavior applies, the string delimiter and escape character is the value of field SQL STRING DELIMITER on installation panel DSNTIPF at the local DB2 when the statement is prepared.

For a list of the DYNAMICRULES bind option values that specify run, bind, define, or invoke behavior, see Table 2 on page 44.
 - If DYNAMICRULES bind, define, or invoke behavior applies and the value of install panel field USE FOR DYNAMICRULES is YES, the string delimiter and escape character is the value of field SQL STRING DELIMITER.

If bind, define, or invoke behavior applies and USE FOR DYNAMICRULES is NO, the precompiler option determines the string delimiter and escape character. For COBOL programs, precompiler option APOSTSQL or QUOTESQL determines the string delimiter and escape character. If neither precompiler option is specified, the value of field SQL STRING DELIMITER determines them. For programs written in other host languages, the default precompiler option, which can only be APOSTSQL, determines the string delimiter and escape character.

Katakana characters for EBCDIC

The field EBCDIC CODED CHAR SET on installation panel DSNTIPF determines the system CCSIDs for EBCDIC-encoded data. Ordinary identifiers with an EBCDIC encoding scheme can contain Katakana characters if the field contains the value 5026 or 930. There are no corresponding precompiler options. EBCDIC CODED CHAR SET applies equally to static and dynamic statements. For dynamically prepared statements, the applicable value is always the one at the local DB2.

Mixed data in character strings

The field MIXED DATA on installation panel DSNTIPF can have the value YES or NO for ASCII or EBCDIC character strings. The value YES indicates that character strings can contain a mixture of SBCS and DBCS characters. The value NO indicates that they cannot. For Unicode, the default is always mixed. A corresponding precompiler option (GRAPHIC or NOGRAPHIC) exists for every host language supported.

For static SQL statements, the value of the precompiler option determines whether ASCII or EBCDIC character strings can contain mixed data. For dynamic SQL statements, either the value of field MIXED DATA or the precompiler option is used, depending on the value of bind option DYNAMICRULES in effect:

- If DYNAMICRULES run behavior applies, field MIXED DATA is used. For a list of the DYNAMICRULES bind option values that specify run, bind, define, or invoke behavior, see Table 2 on page 44.
- If bind, define, or invoke behavior applies and the value of install panel field USE FOR DYNAMICRULES is YES, field MIXED DATA is used. If USE FOR DYNAMICRULES is NO, the precompiler option is used.

The value of MIXED DATA and the precompiler option affects the parsing of SQL character string constants, the execution of the LIKE predicate, and the assignment of character strings to host variables when truncation is needed. It can also affect concatenation, as explained in “With the concatenation operator” on page 112. A value that applies to a statement executed at the local DB2 also applies to any statement executed at another server. An exception is the LIKE predicate, for which the applicable value of MIXED DATA is always the one at the statement’s server.

The value of MIXED DATA also affects the choice of system CCSIDs for the local DB2 and the choice of data subtypes for character columns. When this value is YES, multiple CCSIDs are available for ASCII and EBCDIC data (SBCS, DBCS, and MIXED). The CCSID specified in the ASCII CODED CHAR SET or EBCDIC CODED CHAR SET field is the MIXED CCSID. In this case, DB2 derives the SBCS and MIXED CCSIDs from the DBCS CCSID specified installation panel DSNTIPF. Moreover, a character column can have any one of the allowable data subtypes—BIT, SBCS, or MIXED.

On the other hand, when MIXED DATA is NO, the only ASCII or EBCDIC system CCSIDs are those for SBCS data. Therefore, only BIT and SBCS can be data subtypes for character columns.

Formatting of datetime strings

Fields on installation panel DSNTIPF (DATE FORMAT, TIME FORMAT, LOCAL DATE LENGTH, and LOCAL TIME LENGTH) and DB2 precompiler options affect the formatting of datetime strings.

The formatting of datetime strings is described in “String representations of datetime values” on page 57. Unlike the subsystem parameters and options previously described, a value in effect for a statement executed at the local DB2 is not necessarily in effect for a statement executed at a different server. See “Restrictions on the use of local datetime formats” on page 59 for more information.

SQL standard language

DB2 SQL and the SQL standard are not identical. The STDSQL precompiler option addresses some of the differences:

- STDSQL(NO) indicates that conformance with the SQL standard is not intended. The default is the value of field STD SQL LANGUAGE on installation panel DSNTIP4 (which has a default of NO).
- STDSQL(YES)²⁰ indicates that conformance with the SQL standard is intended.

When a program is precompiled with the STDSQL(YES) option, the following rules apply:

20. STDSQL(86) is a synonym, but STDSQL(YES) should be used.

Options affecting SQL

Declaring host variables: All host variable declarations must lie between pairs of BEGIN DECLARE SECTION and END DECLARE SECTION statements:

```
BEGIN DECLARE SECTION  
  (one or more host variable declarations)  
END DECLARE SECTION
```

Separate pairs of these statements can bracket separate sets of host variable declarations.

Declarations for SQLCODE and SQLSTATE: The programmer must declare host variables for either SQLCODE or SQLSTATE, or both. SQLCODE should be defined as a fullword integer and SQLSTATE should be defined as a 5-byte character string. SQLCODE and SQLSTATE cannot be part of any structure. The variables must be declared in the DECLARE SECTION of a program; however, SQLCODE can be declared outside of the DECLARE SECTION when no host variable is defined for SQLSTATE. For PL/I, an acceptable declaration can look like this:

```
DECLARE SQLCODE BIN FIXED(31);  
DECLARE SQLSTATE CHAR(5);
```

In Fortran programs, the variable SQLCOD should be used for SQLCODE, and either SQLSTATE or SQLSTA can be used for SQLSTATE.

Definitions for the SQLCA: An SQLCA must not be defined in your program, either by coding its definition manually or by using the INCLUDE SQLCA statement. When STDSQL(YES) is specified, the DB2 precompiler automatically generates an SQLCA that includes the variable name SQLCADE instead of SQLCODE and SQLSTAT instead of SQLSTATE. After each SQL statement executes, DB2 assigns status information to SQLCODE and SQLSTATE, whose declarations are described above, as follows:

- SQLCODE: DB2 assigns the value in SQLCADE to SQLCODE. In Fortran, SQLCAD and SQLCOD are used for SQLCADE and SQLCODE, respectively.
- SQLSTATE: DB2 assigns the value in SQLSTAT to SQLSTATE. (In Fortran, SQLSTT and SQLSTA are used for SQLSTAT and SQLSTATE, respectively.)
- No declaration for either SQLSTATE or SQLCODE: DB2 assigns the value in SQLCADE to SQLCODE.

If the precompiler encounters an INCLUDE SQLCA statement, it ignores the statement and issues a warning message. The precompiler also does not recognize hand-coded definitions, and a hand-coded definition creates a compile-time conflict with the precompiler-generated definition. A similar conflict arises if definitions of SQLCADE or SQLSTAT, other than the ones generated by the DB2 precompiler, appear in the program.

Comments in static SQL statements: Static SQL statements can include SQL comments. Two consecutive hyphens (--) indicate that the characters after the hyphens are a comment.

SQL comments are recognized only in a program that has been precompiled with the STDSQL(YES) option. If STDSQL(YES) is not specified, the use of an SQL comment might cause a syntax error. Host language comments can be used instead.

When allowed, SQL comments are subject to the following rules:

- The two hyphens must be on the same line, not separated by a space.
- Comments can be started wherever a space is valid (except within a delimiter token or between EXEC and SQL).
- Comments are terminated by the end of the line.
- Comments are not allowed within statements that are dynamically prepared (using PREPARE or EXECUTE IMMEDIATE).
- Within a statement embedded in a COBOL program, the two hyphens must be preceded by a blank unless they begin a line.

This example shows how to include comments in a statement:

```
EXEC SQL CREATE VIEW PRJ_MAXPER -- projects with most support personnel
      AS SELECT PROJNO, PROJNAME -- number and name of project
            FROM DSN8710.PROJ
              WHERE DEPTNO = 'E21'      -- systems support dept code
                AND PRSTAFF > 1
END-EXEC.
```

Positioned updates of columns

The NOFOR precompiler option affects the use of the FOR UPDATE clause. The NOFOR option is in effect when either of the following are true:

- The NOFOR option is specified.
- The STDSQL(YES) option is in effect.

Otherwise, the NOFOR option is not in effect. The following table summarizes the differences when the option is in effect and when the option is not in effect:

Table 27. The NOFOR precompiler option

When NOFOR is in effect	When NOFOR is not in effect
The use of the FOR UPDATE clause in the SELECT statement of the DECLARE CURSOR statement is optional. This clause restricts updates to the specified columns and causes the acquisition of update locks when the cursor is used to fetch a row. If no columns are specified, positioned updates can be made to any updatable columns in the table or view that is identified in the first FROM clause in the SELECT statement. If the FOR UPDATE OF clause is not specified, positioned updates can be made to any columns that the program has DB2 authority to update.	The FOR UPDATE clause must be specified.
DBRMs must be built entirely in virtual storage, which might possibly increase the virtual storage requirements of the DB2 precompiler. However, creating DBRMs entirely in virtual storage might ease concurrency problems with DBRM libraries.	DBRMs can be built incrementally using the DB2 precompiler.

#

Precompiler options do not affect ODBC behavior.

Chapter 3. Functions

A *function* is an operation denoted by a function name followed by zero or more
operands that are enclosed in parentheses. It represents a relationship between a
set of input values and a set of result values. The input values to a function are
called *arguments*.

The types of functions are column, scalar, and table. A built-in function is classified
as a column function or a scalar function. A user-defined function can be a column,
scalar, or table function. For more information on the types of functions, see
| “Functions” on page 104. In the syntax of SQL, the term *function* is used only in the
| definition of an expression. Thus, a function can be used only where an expression
| is used. However, some restrictions apply to the use of column functions as
specified in “Column functions” on page 161 and in Chapter 4, “Queries,” on page
347. Most of the built-in functions can also be used as the source function for a
user-defined function as described under “CREATE FUNCTION (sourced)” on page
571.

One set of functions that DB2 provides are DB2 MQSeries® functions, which
integrate MQSeries messaging operations within SQL statements. The functions
help you integrate MQSeries messaging with database applications. You can use
the functions to access MQSeries messaging from within SQL statements and to
combine MQSeries messaging with DB2 database access.

To use the MQSeries functions that provide support for XML messages, you must
have IBM DB2 XML Extender installed. DB2 XML Extender introduces the
user-defined data types DB2XML.XMLVARCHAR and DB2XML.XMLCLOB that are
used to support XML data in message processing. XMLVARCHAR is defined as
VARCHAR(3000) and XMLCLOB is defined as CLOB(2G). Refer to *DB2 XML*
Extender for OS/390 and z/OS Administration and Programming for more
information about DB2 XML Extender.

The MQSeries functions are installed into DB2 and provide access to the MQSeries
server using AMI (Application Messaging Interface). AMI supports the use of an
external configuration file (the AMI repository) to store configuration information. AMI
uses two key concepts: service point and policy. A service point is a logical endpoint
from which a message may be sent or received. Policy defines the quality of
service option that should be used for a given messaging operation. The MQSeries
functions can read, receive, or send messages to the queue specified by the
service and policy in the AMI repository.

The functions can be scalar or table functions. For more information on using
MQSeries functions, see the information on enabling MQSeries functions in *DB2*
Installation Guide and on programming techniques in *DB2 Application Programming*
and *SQL Guide*.

Table 28 lists the functions that DB2 supports.

Table 28. Supported functions

Function name	Description	Page
ABS or ABSVAL	Returns the absolute value of its argument	174
ACOS	Returns the arccosine of an argument as an angle, expressed in radians	175

Functions

Table 28. Supported functions (continued)

Function name	Description	Page
ADD_MONTHS	Returns a date that represents the date argument plus the number of months argument	176
ASIN	Returns the arcsine of an argument as an angle, expressed in radians	178
ATAN	Returns the arctangent of an argument as an angle, expressed in radians	179
ATANH	Returns the hyperbolic arctangent of an argument as an angle, expressed in radians	180
ATAN2	Returns the arctangent of x and y coordinates as an angle, expressed in radians	181
AVG	Returns the average of a set of numbers	162
BLOB	Returns a BLOB representation of its argument	182
CCSID_ENCODING	Returns the encoding scheme of a CCSID with a value of ASCII, EBCDIC, UNICODE, or UNKNOWN	183
CEIL or CEILING	Returns the smallest integer greater than or equal to the argument	184
CHAR	Returns a fixed-length character string representation of its argument	185
CLOB	Returns a CLOB representation of its argument	191
COALESCE	Returns the first argument in a set of arguments that is not null	192
CONCAT	Returns the concatenation of two strings	194
COS	Returns the cosine of an argument that is expressed as an angle in radians	195
COSH	Returns the hyperbolic cosine of an argument that is expressed as an angle in radians	196
COUNT	Returns the number of rows or values in a set of rows or values	163
COUNT_BIG	Same as COUNT, except the result can be greater than the maximum value of an integer	164
DATE	Returns a date derived from its argument	197
DAY	Returns the day part of its argument	198
DAYOFMONTH	Similar to DAY	199
DAYOFWEEK	Returns an integer in the range of 1 to 7, where 1 represents Sunday	200
DAYOFWEEK_ISO	Returns an integer in the range of 1 to 7, where 1 represents Monday	201
DAYOFYEAR	Returns an integer in the range of 1 to 366, where 1 represents January 1	202
DAYS	Returns an integer representation of a date	203
DBCLOB	Returns a DBCLOB representation of its argument	204
DECIMAL or DEC	Returns a decimal representation of its argument	205
DEGREES	Returns the number of degrees for an argument that is expressed in radians	207
DIGITS	Returns a character string representation of a number	208

Table 28. Supported functions (continued)

Function name	Description	Page
DOUBLE or DOUBLE-PRECISION	Returns a double precision floating-point representation of its argument	209
EXP	Returns the exponential function of an argument	210
FLOAT	Same as DOUBLE	209
FLOOR	Returns the largest integer that is less than or equal to the argument	212
# GENERATE_UNIQUE	Returns a bit character string that is unique compared to any other execution of the function	213
I GRAPHIC	Returns a GRAPHIC representation of its argument	215
HEX	Returns a hexadecimal representation of its argument	218
HOUR	Returns the hour part of its argument	219
I IDENTITY_VAL_LOCAL	Returns the most recently assigned value for an identity column	220
IFNULL	Returns the first argument in a set of two arguments that is not null	224
INSERT	Returns a string that is composed of an argument inserted into another argument at the same position where some number of bytes have been deleted	225
INTEGER or INT	Returns an integer representation of its argument	228
JULIAN_DAY	Returns an integer that represents the number of days from January 1, 4712 B.C.	229
LAST_DAY	Returns a date that represents the last day of the month of the date argument	230
LCASE or LOWER	Returns a string with the characters converted to lowercase	231
LEFT	Returns a string that consists of the specified number of leftmost bytes of a string	232
LENGTH	Returns the length of its argument	234
I LN	Returns the natural logarithm of an argument	235
LOCATE	Returns the position at which the first occurrence of an argument starts within another argument	236
LOG10	Returns the base 10 logarithm of an argument	238
LTRIM	Returns the characters of a string with the leading blanks removed	239
MAX	Returns the maximum value in a set of column values	166
I MAX (scalar)	Returns the maximum value in a set of values	240
MICROSECOND	Returns the microsecond part of its argument	241
MIDNIGHT_SECONDS	Returns an integer in the range of 0 to 86400 that represents the number of seconds between midnight and the argument	242
MIN	Returns the minimum value in a set of column values	167
I MIN (scalar)	Returns the minimum value in a set of values	243
MINUTE	Returns the minute part of its argument	244
MOD	Returns the remainder of one argument divided by a second argument	245
MONTH	Returns the month part of its argument	247

Functions

Table 28. Supported functions (continued)

Function name	Description	Page
# MQPUBLISH	Publishes a message to the specified MSQSeries publisher	248
# MQPUBLISHXML	Publishes the XML data in a message to the specified MSQSeries publisher	250
# MQREAD	Returns a message from a specified MQSeries location (return value of VARCHAR) without removing the message from the queue	252
# MQREADALL	Returns a table containing the messages and message metadata from a specified MQSeries location with a VARCHAR column and without removing the messages from the queue	335
# MQREADALLCLOB	Returns a table containing the messages and message metadata from a specified MQSeries location with a CLOB column and without removing the messages from the queue	337
# MQREADALLXML	Returns a table containing the messages and message metadata from a specified MQSeries location with a column that contains XML data without removing the messages from the queue	339
# MQREADCLOB	Returns a message from a specified MQSeries location (return value of CLOB) without removing the message from the queue	254
# MQREADXML	Returns a message from a specified MQSeries location (return value of VARCHAR) without removing the message from the queue	256
# MQRECEIVE	Returns a message from a specified MQSeries location (return value of VARCHAR) with removal of message from the queue	257
# MQRECEIVEALL	Returns a table containing the messages and message metadata from a specified MQSeries location with a VARCHAR column and with removal of messages from the queue	341
# MQRECEIVEALLCLOB	Returns a table containing the messages and message metadata from a specified MQSeries location with a CLOB column and with removal of messages from the queue	343
# MQRECEIVEALLXML	Returns a table containing the messages and message metadata from a specified MQSeries location with column that contains XML data and with removal of messages from the queue	345
# MQRECEIVECLOB	Returns a message from a specified MQSeries location (return value of CLOB) with removal of message from the queue	259
# MQRECEIVEXML	Returns a message from a specified MQSeries location (return value of XML data in a user-defined data type that is based on VARCHAR) with removal of message from the queue	261
# MQSEND	Sends data contained in msg-data to a specified MQSeries location	263
# MQSENDXML	Sends the XML data contained in msg-data to a specified MQSeries location	265
# MQSENDXMLFILE	Sends data contained in an XML file that is up to 3K in size to a specified MQSeries location	267

Table 28. Supported functions (continued)

Function name	Description	Page
# MQSENDXMLFILECLOB	Sends data contained in an XML file that is up to 1M in size to a specified MQSeries location	269
# MQSUBSCRIBE	Registers a subscription to MQSeries messages that are published on a specified topic	271
# MQUNSUBSCRIBE	Unregisters an existing subscription to MQSeries messages that are published on a specified topic	273
MULTIPLY_ALT	Returns the product of the two arguments as a decimal value, used when the sum of the argument precisions exceeds 31	275
NEXT_DAY	Returns a timestamp that represents the first weekday, named by the second argument, after the date argument	276
NULLIF	Returns NULL if the arguments are equal; else the first argument	277
POSSTR	Returns the position of the first occurrence of an argument within another argument	278
POWER	Returns the value of one argument raised to the power of a second argument	280
QUARTER	Returns an integer in the range of 1 to 4 that represents the quarter of the year for the date specified in the argument	281
RADIANS	Returns the number of radians for an argument that is expressed in degrees	282
RAISE_ERROR	Raises an error in the SQLCA with the specified SQLSTATE and error description	283
RAND	Returns a double precision floating-point random number	284
REAL	Returns a single precision floating-point representation of its argument	285
REPEAT	Returns a character string composed of an argument repeated a specified number of times	286
REPLACE	Returns a string in which all occurrences of an argument within a second argument are replaced with a third argument	288
RIGHT	Returns a string that consists of the specified number of rightmost bytes of a string	290
ROUND	Returns a number rounded to the specified number of places to the right or left of the decimal place	292
ROUND_TIMESTAMP	Returns a timestamp rounded to the unit specified by the timestamp format string	294
ROWID	Returns a row ID representation of its argument	297
RTRIM	Returns the characters of an argument with the trailing blanks removed	298
SECOND	Returns the second part of its argument	299
SIGN	Returns the sign of an argument	300
SIN	Returns the sine of an argument that is expressed as an angle in radians	301
SINH	Returns the hyperbolic sine of an argument that is expressed as an angle in radians	302
SMALLINT	Returns a small integer representation of its argument	303

Functions

Table 28. Supported functions (continued)

Function name	Description	Page
# SOAPHTTPC or # SOAPHTTPV	Returns a COLB or VARCHAR representation of XML data from a request to a web service	304
SPACE	Returns a string that consists of the number of blanks the argument specifies	305
SQRT	Returns the square root of its argument	306
STDDEV or STDDEV_POP	Returns the standard deviation (/n) of a set of numbers	168
STDDEV_SAMP	Returns the sample standard deviation (/n-1) of a set of numbers	169
STRIP	Returns the characters of a string with the blanks (or specified character) at the beginning, end, or both beginning and end of the string removed	307
SUBSTR	Returns a substring of a string	309
SUM	Returns the sum of a set of numbers	170
TAN	Returns the tangent of an argument that is expressed as an angle in radians	312
TANH	Returns the hyperbolic tangent of an argument that is expressed as an angle in radians	313
TIME	Returns a time derived from its argument	314
TIMESTAMP	Returns a timestamp derived from its arguments	315
TIMESTAMP_FORMAT	Returns a timestamp for a character string expression, using a specified format to interpret the string	316
TRANSLATE	Returns a string with one or more characters translated	317
TRUNCATE or TRUNC	Returns a number truncated to the specified number of places to the right or left of the decimal point	320
TRUNC_TIMESTAMP	Returns a timestamp truncated to the unit specified by the timestamp format string	321
UCASE or UPPER	Returns a string with the characters converted to uppercase	322
VARCHAR	Returns the varying-length character string representation of its argument	323
VARCHAR_FORMAT	Returns a character string representation of a timestamp, with the string in a specified format	327
VARGRAPHIC	Returns a graphic string representation of its argument	328
VARIANCE, VAR, or VAR_POP	Returns the variance of a set of numbers	171
VARIANCE_SAMP or VAR_SAMP	Returns the sample variance of a set of numbers	172
WEEK	Returns an integer that represents the week of the year with Sunday as the first day of the week	331
WEEK_ISO	Returns an integer that represents the week of the year with Monday as first day of a week	332
YEAR	Returns the year part of its argument	333

Column functions

The following information applies to all column functions, except for the COUNT(*) and COUNT_BIG(*) variations of the COUNT and COUNT_BIG functions.

The argument of a column function is a set of values derived from an expression. The expression must include a column name and must not include another column function. The scope of the set is a group or an intermediate result table as explained in Chapter 4, “Queries,” on page 347.

If a GROUP BY clause is specified in a query and the intermediate result from the FROM, WHERE, GROUP BY, and HAVING clauses is the empty set, then the column functions are not applied and the result of the query is the empty set.

If the GROUP BY clause is not specified in a query and the intermediate result set of the FROM, WHERE, and HAVING clauses is the empty set, then the column functions are applied to the empty set.

For example, the result of the following SELECT statement is the number of distinct values of JOB for employees in department D11:

```
SELECT COUNT(DISTINCT JOB)
      FROM DSN8710.EMP
        WHERE WORKDEPT = 'D11';
```

The keyword DISTINCT is not considered an argument of the function but rather a specification of an operation that is performed before the function is applied. If DISTINCT is specified, duplicate values are eliminated. If ALL is implicitly or explicitly specified, duplicate values are not eliminated.

A column function can be used in a WHERE clause only if that clause is part of a subquery of a HAVING clause and the column name specified in the expression is a correlated reference to a group. If the expression includes more than one column name, each column name must be a correlated reference to the same group.

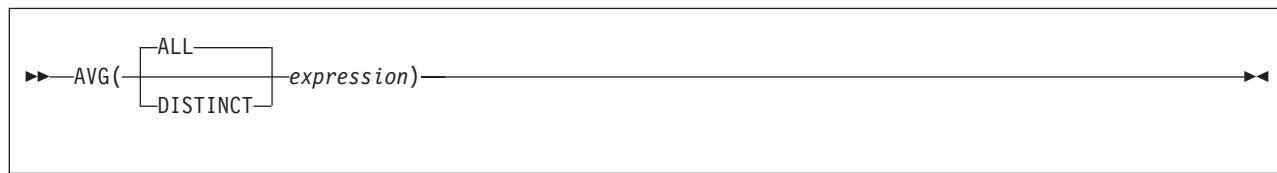
The result of the COUNT and COUNT_BIG functions cannot be the null value. As specified in the description of AVG, MAX, MIN, STDDEV, SUM, and VARIANCE, the result is the null value when the function is applied to an empty set. However, the result is also the null value when the function is specified in an outer select list, the argument is given by an arithmetic expression, and any evaluation of the expression causes an arithmetic exception (such as division by zero).

If the argument values of a column function are strings from a column with a field procedure, the function is applied to the encoded form of the values and the result of the function inherits the field procedure.

Following in alphabetic order is a definition of each of the built-in column functions.

AVG

AVG



The schema is SYSIBM.

The AVG function returns the average of a set of numbers.

The argument values must be of any built-in numeric data type, and their sum must be within the range of the data type of the result.

The data type of the result is the same as the data type of the argument values, except that the result is a large integer if the argument values are small integers, and the result is double precision floating-point if the argument values are single precision floating-point. The result can be null.

If the data type of the argument values is decimal with precision p and scale s , the precision (P) and scale (S) of the result depend on p and the decimal precision option:

- If p is greater than 15 or the DEC31 option is in effect, P is 31 and S is $\max(0, 28-p+s)$.
- Otherwise, P is 15 and S is $15-p+s$.

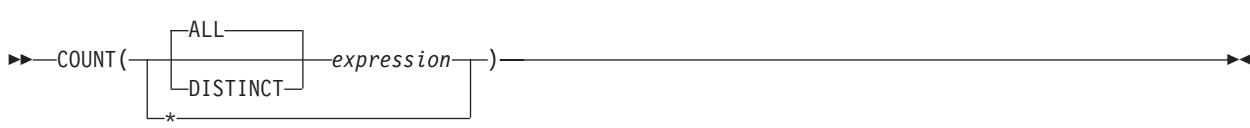
The function is applied to the set of values derived from the argument values by the elimination of null values. If DISTINCT is specified, duplicate values are also eliminated.

If the function is applied to an empty set, the result is the null value. Otherwise, the result is the average value of the set. If the type of the result is an integer, the fractional part of the average is lost. The order in which the summation part of the operation is performed is undefined but every intermediate result must be within the range of the result data type.

Example: Assuming DEC15, set the DECIMAL(15,2) variable AVERAGE to the average salary in department D11 of the employees in the sample table DSN8710.EMP.

```
EXEC SQL SELECT AVG(SALARY)
  INTO :AVERAGE
  FROM DSN8710.EMP
 WHERE WORKDEPT = 'D11';
```

COUNT



The schema is SYSIBM.

The COUNT function returns the number of rows or values in a set of rows or values.

The argument values can be of any built-in data type other than a BLOB, CLOB, or DBCLOB. If DISTINCT is used, the resulting expression cannot have a maximum length greater than 255 for a character column and 127 for a graphic column.

The result is a large integer. The result cannot be null.

The argument of COUNT(*expression*) or COUNT(ALL *expression*) is a set of values. The function is applied to the set of values derived from the argument values by the elimination of null values. The result is the number of non-null values in the set, including duplicates.

The argument of COUNT(DISTINCT *expression*) is a set of values. The function is applied to the set of values derived from the argument values by the elimination of null values and duplicate values. The result is the number of values in the set.

The argument of COUNT(*) is a set of rows. The result is the number of rows in the set. Any row that includes only null values is included in the count.

Example 1: Set the integer host variable FEMALE to the number of females represented in the sample table DSN8710.EMP.

```

EXEC SQL SELECT COUNT(*)
  INTO :FEMALE
  FROM DSN8710.EMP
  WHERE SEX = 'F';
  
```

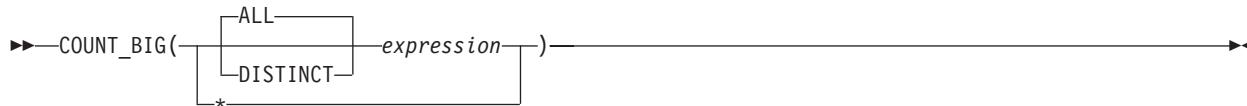
Example 2: Set the integer host variable FEMALE_IN_DEPT to the number of departments that have at least one female as a member.

```

EXEC SQL SELECT COUNT(DISTINCT WORKDEPT)
  INTO :FEMALE_IN_DEPT
  FROM DSN8710.EMP
  WHERE SEX = 'F';
  
```

COUNT_BIG

COUNT_BIG



The schema is SYSIBM.

The COUNT_BIG function returns the number of rows or values in a set of rows or values. It is similar to COUNT except that the result can be greater than the maximum value of an integer.

The argument values can be of any built-in data type other than a BLOB, CLOB, or DBCLOB. If DISTINCT is used, the resulting expression must not have a maximum length greater than 255 for a character column or 127 for a graphic column.

The result of the function is a decimal number with precision 31 and scale 0. The result cannot be null.

The argument of COUNT_BIG(*expression*) or COUNT_BIG(ALL *expression*) is a set of values. The function is applied to the set of values derived from the argument values by the elimination of null values. The result is the number of non-null values in the set, including duplicates.

The argument of COUNT_BIG(DISTINCT *expression*) is a set of values. The function is applied to the set of values derived from the argument values by the elimination of null and duplicate values. The result is the number of different nonnull values in the set.

The argument of COUNT_BIG(*) is a set of rows. The result is the number of rows in the set. A row that includes only null values is included in the count.

Example 1: The examples for COUNT are also applicable for COUNT_BIG. Refer to the COUNT examples and substitute COUNT_BIG for the occurrences of COUNT. The results are the same except for the data type of the result.

Example 2: To create a sourced function that is similar to the built-in COUNT_BIG function, the definition of the sourced function must include the type of the column that can be specified when the new function is invoked. In this example, the CREATE FUNCTION statement creates a sourced function that takes a CHAR column as input and uses COUNT_BIG to perform the counting. The result is returned as a double precision floating-point number. The query shown counts the number of unique departments in the sample employee table.

```
CREATE FUNCTION RICK.COUNT(CHAR()) RETURNS DOUBLE  
    SOURCE SYSIBM.COUNT_BIG(CHAR());
```

```
SET PATH RICK, SYSTEM PATH;
```

```
SELECT COUNT(DISTINCT WORKDEPT) FROM DSN8710.EMP;
```

The empty parenthesis in the parameter list for the new function (RICK.COUNT) means that the input parameter for the new function is the same type as the input parameter for the function named in the SOURCE clause. The empty parenthesis in

| the parameter list in the SOURCE clause (SYSIBM.COUNT_BIG) means that when
| DB2 uses function resolution to identify the source function, the attribute (in this
| example the length) is ignored for determining whether the data types match.

MAX

MAX

The diagram shows the SQL MAX function syntax. It consists of the word 'MAX' followed by a left parenthesis. Inside the parenthesis, there is a horizontal line with two vertical branches. The top branch is labeled 'ALL' and the bottom branch is labeled 'DISTINCT'. To the right of this line is the word 'expression)'.

The schema is SYSIBM.

The MAX function returns the maximum value in a set of values.

The argument values can be of any built-in data type other than a BLOB, CLOB, DBCLOB, or row ID. Character string arguments cannot have a maximum length greater than 255, and graphic string arguments cannot have a maximum length greater than 127.

The data type of the result and its other attributes (for example, the length and CCSID of a string) are the same as the data type and attributes of the argument values. The result can be null.

The function is applied to the set of values derived from the argument values by the elimination of null values.

If the function is applied to an empty set, the result is the null value. Otherwise, the result is the maximum value in the set.

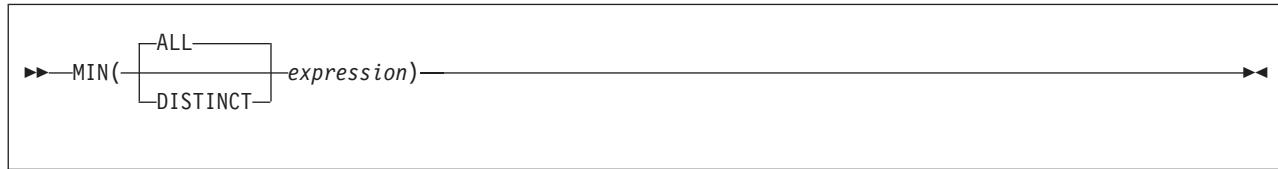
The specification of DISTINCT has no effect on the result and is not advised.

Example 1: Set the DECIMAL(8,2) variable MAX_SALARY to the maximum monthly salary of the employees represented in the sample table DSN8710.EMP.

```
EXEC SQL SELECT MAX(SALARY) / 12
  INTO :MAX_SALARY
  FROM DSN8710.EMP;
```

Example 2: Find the surname that comes last in the collating sequence for the employees represented in the sample table DSN8710.EMP. Set the VARCHAR(15) variable LAST_NAME to that surname.

```
EXEC SQL SELECT MAX(LASTNAME)
  INTO :LAST_NAME
  FROM DSN8710.EMP;
```

MIN

The schema is SYSIBM.

The MIN function returns the minimum value in a set of values.

The argument values can be of any built-in data type other than a BLOB, CLOB, DBCLOB, or row ID. Character string arguments cannot have a maximum length greater than 255, and graphic string arguments cannot have a maximum length greater than 127.

The data type of the result and its other attributes (for example, the length and CCSID of a string) are the same as the data type and attributes of the argument values. The result can be null.

The function is applied to the set of values derived from the argument values by the elimination of null values.

If the function is applied to an empty set, the result is the null value. Otherwise, the result is the minimum value in the set.

The specification of DISTINCT has no effect on the result and is not advised.

Example 1: Set the DECIMAL(15,2) variable MIN_SALARY to the minimum monthly salary of the employees represented in the sample table DSN8710.EMP.

```

EXEC SQL SELECT MIN(SALARY) / 12
  INTO :MIN_SALARY
  FROM DSN8710.EMP;
  
```

Example 2: Find the surname that comes first in the collating sequence for the employees represented in the sample table DSN8710.EMP. Set the VARCHAR(15) variable FIRST_NAME to that surname.

```

EXEC SQL SELECT MIN(LASTNAME)
  INTO :FIRST_NAME
  FROM DSN8710.EMP;
  
```

STDDEV

STDDEV or STDDEV_POP

```
►►STDDEV([ALL|DISTINCT] expression) (1)►►
```

Notes:

- 1 STDDEV_POP can be specified as an alternative to STDDEV.

The schema is SYSIBM.

The STDDEV function returns the standard deviation ($/n$) of a set of numbers. The formula that is used to calculate STDDEV is:

$$\text{STDDEV} = \text{SQRT}(\text{VAR})$$

where SQRT(VAR) is the square root of the variance.

The argument values must each be the value of any built-in numeric data type, and their sum must be within the range of the data type of the result.

The result of the function is double precision floating-point number. The result can be null.

Before the function is applied to the set of values derived from the argument values, null values are eliminated. If DISTINCT is specified, duplicate values are also eliminated.

If the function is applied to an empty set, the result is the null value. Otherwise, the result is the standard deviation of the values in the set.

The order in which the values are aggregated is undefined, but every intermediate result must be within the range of the result data type.

Example: Using sample table DSN8710.EMP, set the host variable DEV, which is defined as double precision floating-point, to the standard deviation of the salaries for the employees in department 'A00' (WORKDEPT='A00').

```
SELECT STDDEV(SALARY)
  INTO :DEV
  FROM DSN8710.EMP
 WHERE WORKDEPT = 'A00';
```

For this example, host variable DEV is set to a double precision float-pointing number with an approximate value of 9742.43.

STDDEV_SAMP

```
►► STDDEV_SAMP( [ALL | DISTINCT] expression )
```

The schema is SYSIBM.

The STDDEV_SAMP function returns the standard deviation ($\sqrt{\frac{1}{n-1} \sum (x - \bar{x})^2}$) of a set of numbers. The formula that is used to calculate STDDEV_SAMP is:

$$\text{STDDEV_SAMP} = \text{SQRT}(\text{VAR_SAMP})$$

where SQRT(VAR_SAMP) is the square root of the sample variance.

The argument values must each be the value of any built-in numeric data type, and their sum must be within the range of the data type of the result.

The result of the function is a double precision floating-point number. The result can be null.

Before the function is applied to the set of values derived from the argument values, null values are eliminated. If DISTINCT is specified, duplicate values are also eliminated.

If the function is applied to an empty set, or a set with only one row, the result is the null value. Otherwise, the result is the sample standard deviation of the values in the set.

The order in which the values are aggregated is undefined, but every intermediate result must be within the range of the result data type.

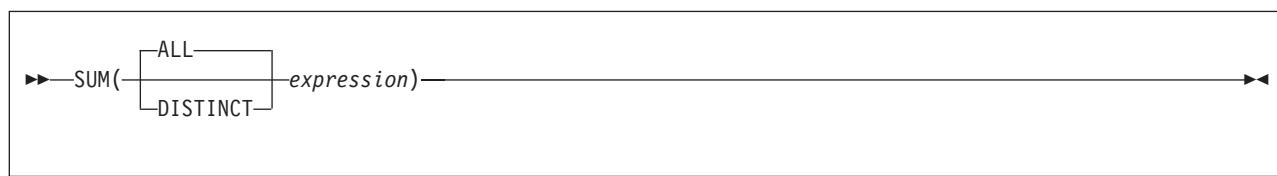
Example: Using sample table DSN8710.EMP, set the host variable DEV, which is defined as double precision floating-point, to the sample standard deviation of the salaries for the employees in department 'A00' (WORKDEPT='A00').

```
SELECT STDDEV_SAMP(SALARY)
      INTO :DEV
      FROM DSN8710.EMP
      WHERE WORKDEPT = 'A00';
```

For this example, host variable DEV is set to a double precision float-pointing number with an approximate value of +1.08923711835394E+004.

SUM

SUM



The schema is SYSIBM.

The SUM function returns the sum of a set of numbers.

The argument values must be of any built-in numeric data type, and their sum must be within the range of the data type of the result.

The data type of the result is the same as the data type of the argument values, except that the result is a large integer if the argument values are small integers, and the result is double precision floating-point if the argument values are single precision floating-point. The result can be null.

If the data type of the argument values is decimal, the scale of the result is the same as the scale of the argument values, and the precision of the result depends on the precision of the argument values and the decimal precision option:

- If the precision of the argument values is greater than 15 or the DEC31 option is in effect, the precision of the result is min(31,P+10), where P is the precision of the argument values.
- Otherwise, the precision of the result is 15.

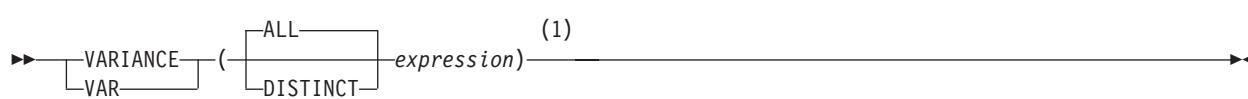
The function is applied to the set of values derived from the argument values by the elimination of null values. If DISTINCT is specified, duplicate values are also eliminated.

If the function is applied to an empty set, the result is the null value. Otherwise, the result is the sum of the values in the set. The order in which the summation is performed is undefined but every intermediate result must be within the range of the result data type.

Example: Set the large integer host variable INCOME to the total income from all sources (salaries, commissions, and bonuses) of the employees represented in the sample table DSN8710.EMP. If DEC31 is not in effect, the resultant sum is DECIMAL(15,2) because all three columns are DECIMAL(9,2).

```
EXEC SQL SELECT SUM(SALARY+COMM+BONUS)
  INTO :INCOME
  FROM DSN8710.EMP;
```

I VARIANCE, VAR, or VAR_POP



Notes:

- 1 VAR_POP can be specified as an alternative to VARIANCE or VAR.

The schema is SYSIBM.

The VARIANCE or VAR function returns the variance of a set of numbers. The result is the biased variance ($/n$) of the set of numbers. The formula used to calculate VARIANCE is:

$$\text{VARIANCE} = \text{SUM}(X^{**2})/\text{COUNT}(X) - (\text{SUM}(X)/\text{COUNT}(X))^{**2}$$

The argument values must be of any built-in numeric type, and their sum must be within the range of the data type of the result.

The result of the function is a double precision floating-point number. The result can be null.

Before the function is applied to the set of values derived from the argument values, null values are limited. If DISTINCT is specified, duplicate values are also eliminated.

If the function is applied to an empty set, the result is the null value. Otherwise, the result is the variance of the values in the set.

The order in which the values are added is undefined, but every intermediate result must be within the range of the result data type.

Example: Using sample table DSN8710.EMP, set host variable VARNCE, which is defined as double precision floating-point, to the variance of the salaries (SALARY) for those employees in department (WORKDEPT) 'A00'.

```
SELECT VARIANCE(SALARY)
  INTO :VARNCE
  FROM DSN8710.EMP
 WHERE WORKDEPT = 'A00';
```

The result in VARNCE is set to a double precision-floating point number with an approximate value of 94915000.00.

VARIANCE_SAMP or VAR_SAMP

VARIANCE_SAMP or VAR_SAMP

```
►-- VARIANCE_SAMP --> ( [ALL] expression )  
|-- VAR_SAMP -->  
|-- DISTINCT -->
```

The schema is SYSIBM.

The VARIANCE_SAMP or VAR_SAMP function returns the sample variance of a set of numbers. The result is the sample variance ($/n-1$) of the set of numbers. The formula used to calculate VARIANCE is:

$$\text{VAR_SAMP} = (\text{SUM}(X^{**2}) - ((\text{SUM}(X)^{**2}) / (\text{COUNT}(*)))) / (\text{COUNT}(*))$$

The argument values must be of any built-in numeric type, and their sum must be within the range of the data type of the result.

The result of the function is a double precision floating-point number. The result can be null.

Before the function is applied to the set of values derived from the argument values, null values are limited. If DISTINCT is specified, duplicate values are also eliminated.

If the function is applied to an empty set, or a set with only one row, the result is the null value. Otherwise, the result is the variance of the values in the set.

The order in which the values are added is undefined, but every intermediate result must be within the range of the result data type.

Example: Using sample table DSN8710.EMP, set host variable VARNCE, which is defined as double precision floating-point, to the sample variance of the salaries (SALARY) for those employees in department (WORKDEPT) 'A00'.

```
SELECT VARIANCE_SAMP(SALARY)  
  INTO :VARNCE  
  FROM DSN8710.EMP  
 WHERE WORKDEPT = 'A00';
```

The result in VARNCE is set to a double precision-floating point number with an approximate value of 118643750.

Scalar functions

A scalar function can be used wherever an expression can be used. However, the restrictions that apply to the use of expressions and column functions also apply when an expression or column function is used within a scalar function. For example, the argument of a scalar function can be a column function only if a column function is allowed in the context in which the scalar function is used.

If the argument of a scalar function is a string from a column with a field procedure, the function applies to the decoded form of the value and the result of the function does not inherit the field procedure.

Example: The following SELECT statement calls for the employee number, last name, and age of each employee in department D11 in the sample table DSN8710.EMP. To obtain the ages, the scalar function YEAR is applied to the expression:

```
CURRENT DATE - BIRTHDATE
```

in each row of DSN8710.EMP for which the employee represented is in department D11:

```
SELECT EMPNO, LASTNAME, YEAR(CURRENT DATE - BIRTHDATE)
  FROM DSN8710.EMP
 WHERE WORKDEPT = 'D11';
```

Following in alphabetic order is the definition of each of the scalar functions.

ABS

ABS or ABSVAL

```
►—ABS—(—expression—)►
```

|
| ABSVAL
|

The schema is SYSIBM.

The ABS function returns the absolute value of the argument.

The argument is an expression that returns a value of any built-in numeric data type.

The result of the function has the same data type and length attribute as the argument. The result can be null. If the argument is null, the result is the null value.

Example: Assume that host variable PROFIT is a large integer with a value of -50000. The following statement returns a large integer with a value of 50000.

```
SELECT ABS(:PROFIT)
      FROM SYSIBM.SYSDUMMY1;
```

ACOS

```
►►—ACOS(expression)-----►►
```

The schema is SYSIBM.

The ACOS function returns the arccosine of the argument as an angle expressed in radians. The ACOS and COS functions are inverse operations.

The argument is an expression whose value is a number in the range of -1 to 1, and has any built-in numeric data type. If the argument is not a double precision floating-point number, it is converted to one for processing by the function.

The result of the function is a double precision floating-point number. The result can be null; if the argument is null, the result is the null value.

Example: Assume that host variable ACOSINE is DECIMAL(10,9) with a value of 0.070737202. The following statement:

```
SELECT ACOS(:ACOSINE)
      FROM SYSIBM.SYSDUMMY1;
```

returns a double precision floating-point number with an approximate value of 1.49.

ADD_MONTHS

ADD_MONTHS

```
►►—ADD_MONTHS(date-expression,expression)————►►
```

The schema is SYSIBM.

The ADD_MONTHS function returns a date that represents *date-expression* plus *expression* months.

date-expression must be a date, a timestamp, or a valid string representation of a date or timestamp. A string representation must not be a BLOB, CLOB, or DBCLOB, and must have an actual length that is not greater than 255 bytes. For the valid formats of string representations of dates and timestamps, see “String representations of datetime values” on page 57.

The result of the function is a DATE. The result can be null; if any argument is null, the result is the null value.

If *date-expression* is the last day of the month or if the resulting month has fewer days than the day component of *date-expression*, then the result is the last day of the resulting month. Otherwise, the result has the same day component as *date-expression*.

expression can be any zero-scale numeric value.

Example 1: Assume today is January 31, 2000. Set the host variable ADD_MONTH with the last day of January plus 1 month.

```
SET :ADD_MONTH = ADD_MONTHS(LAST_DAY(CURRENT_DATE), 1);
```

The host variable ADD_MONTH is set with the value representing the end of February, 2000-02-29.

Example 2: Assume DATE is a host variable with the value July 27, 1965. Set the host variable ADD_MONTH with the value of that day plus 3 months.

```
SET :ADD_MONTH = ADD_MONTHS(:DATE,3);
```

The host variable ADD_MONTH is set with the value representing the day plus 3 months, 1965-10-27.

Example 3: It is possible to achieve similar results with the ADD_MONTHS function and date arithmetic. The following examples demonstrate the similarities and contrasts.

```
SET :DATEHV = DATE('2000-2-28') + 4 MONTHS;  
SET :DATEHV = ADD_MONTHS('2000-2-28', 4);
```

In both cases, the host variable DATEHV is set with the value '2000-06-28'.

Now consider the same examples but with the date '2000-2-29' as the argument.

```
SET :DATEHV = DATE('2000-2-29') + 4 MONTHS;
```

The host variable DATEHV is set with the value '2000-06-29'.

```
SET :DATEHV = ADD_MONTHS('2000-2-29', 4);
```

- | The host variable DATEHV is set with the value '2000-06-30'.
- | In this case, the ADD_MONTHS function returns the last day of the month, which is June 30, 2000, instead of June 29, 2000. The reason is that February 29 is the last day of the month. So, the ADD_MONTHS function returns the last day of June.

ASIN

```
►►ASIN(expression)--►►
```

The schema is SYSIBM.

The ASIN function returns the arcsine of the argument as an angle expressed in radians. The ASIN and SIN functions are inverse operations.

The argument is an expression whose value is a number in the range of -1 to 1, and has any built-in numeric data type. If the argument is not a double precision floating-point number, it is converted to one for processing by the function.

The result of the function is a double precision floating-point number. The result can be null; if the argument is null, the result is the null value.

Example: Assume that host variable ASINE is DECIMAL(10,9) with a value of 0.997494987. The following statement:

```
SELECT ASIN(:ASINE)
   FROM SYSIBM.SYSDUMMY1;
```

returns a double precision floating-point number with an approximate value of 1.50.

ATAN

```
►►—ATAN(expression)-----►►
```

The schema is SYSIBM.

The ATAN function returns the arctangent of the argument as an angle expressed in radians. The ATAN and TAN functions are inverse operations.

The argument is an expression that returns the value of any built-in numeric data type. If the argument is not a double precision floating-point number, it is converted to one for processing by the function.

The result of the function is a double precision floating-point number. The result can be null; if the argument is null, the result is the null value.

Example: Assume that host variable ATANGENT is DECIMAL(10,8) with a value of 14.10141995. The following statement:

```
SELECT ATAN(:ATANGENT)
      FROM SYSIBM.SYSDUMMY1;
```

returns a double precision floating-point number with an approximate value of 1.50.

ATANH

ATANH

```
►►—ATANH(expression)—►►
```

The schema is SYSIBM.

The ATANH function returns the hyperbolic arc tangent of the argument as an angle expressed in radians. The ATANH and TANH functions are inverse operations.

The argument is an expression whose value is a number in the range of -1 to 1, and has any built-in numeric data type. If the argument is not a double precision floating-point number, it is converted to one for processing by the function.

The result of the function is a double precision floating-point number. The result can be null; if the argument is null, the result is the null value.

Example: Assume that host variable HATAN is DECIMAL(10,9) with a value of 0.905148254. The following statement:

```
SELECT ATANH(:HATAN)
   FROM SYSIBM.SYSDUMMY1;
```

returns a double precision floating-point number with an approximate value of 1.50.

ATAN2

```
►►—ATAN2(expression1,expression2)—►►
```

The schema is SYSIBM.

The ATAN2 function returns the arctangent of *x* and *y* coordinates as an angle expressed in radians. The first and second arguments specify the *x* and *y* coordinates, respectively.

Each argument is an expression that returns the value of any built-in numeric data type. Both arguments must not be 0. Any argument that is not a double precision floating-point number is converted to one for processing by the function.

The result of the function is a double precision floating-point number. The result can be null; if any argument is null, the result is the null value.

Example: Assume that host variables HATAN2A and HATAN2B are DOUBLE host variables with values of 1 and 2, respectively. The following statement:

```
SELECT ATAN2(:HATAN2A,:HATAN2B)
   FROM SYSIBM.SYSDUMMY1;
```

returns a double precision floating-point number with an approximate value of 1.1071487.

BLOB

BLOB

```
►►BLOB(expression [ , —integer ])►►
```

The schema is SYSIBM.

The BLOB function returns a BLOB representation of a string of any type or a row ID type.

expression

An expression whose value is a character string, graphic string, binary string, or a row ID type.

integer

An integer value specifying the length attribute of the resulting BLOB data type. The value must be an integer between 0 and the maximum length of a BLOB.

Do not specify *integer* if *expression* is a row ID type.

If you do not specify *integer*, the length attribute of the result is the same as the length attribute of *expression*, except when the input is graphic data. In this case, the length attribute of the result is twice the length of *expression*.

The result of the function is a BLOB. If the first argument can be null, the result can be null; if the first argument is null, the result is the null value.

The actual length of the result is the minimum of the length attribute of the result and the actual length of *expression* (or twice the length of *expression* when the input is graphic data). If the length of *expression* is greater than the length specified, truncation is performed. A warning is returned unless the first input argument is a character string and all the truncated characters are blanks, or the first input argument is a graphic string and all the truncated characters are double-byte blanks.

Example 1: The following function returns a BLOB for the string 'This is a BLOB'.

```
SELECT BLOB('This is a BLOB')  
      FROM SYSIBM.SYSDUMMY1;
```

Example 2: The following function returns a BLOB for the large object that is identified by locator myclob_locator.

```
SELECT BLOB(:myclob_locator)  
      FROM SYSIBM.SYSDUMMY1;
```

Example 3: Assume that a table has a BLOB column named TOPOGRAPHIC_MAP and a VARCHAR column named MAP_NAME. Locate any maps that contain the string 'Engles Island' and return a single binary string with the map name concatenated in front of the actual map.

```
SELECT BLOB(MAP_NAME || ':' || TOPOGRAPHIC_MAP  
      FROM ONTARIO_SERIES_4  
     WHERE TOPOGRAPHIC_MAP LIKE BLOB('%Engles Island%')
```

CCSID_ENCODING

```
►►CCSID_ENCODING(expression)►►
```

The schema is SYSIBM.

The CCSID_ENCODING function returns the encoding scheme of a CCSID in the form of a character string with a value of one of the following: ASCII, EBCDIC, UNICODE, or UNKNOWN.

The argument can be of any built-in data type other than a character string with a maximum length greater than 255 or a graphic string with a length greater than 127. It cannot be a BLOB, CLOB, or DBCLOB.

The result of the function is a fixed-length character string of length 8, which is padded on the right if necessary. If the argument can be null, the result can be null. If the argument is null, the result is the null value.

If another string data from a table or view is selected by a query, the encoding scheme of that string determines the resulting encoding scheme. The resulting CCSID for string data is the appropriate CCSID for the encoding scheme of the statement. If there is no other string data from a table or a view in the query, the default encoding scheme is used.

Example 1: The following function returns a CCSID with a value for EBCDIC data.

```
SELECT CCSID_ENCODING(37) AS CCSID  
      FROM SYSIBM.SYSDUMMY1;
```

Example 2: The following function returns a CCSID with a value for ASCII data.

```
SELECT CCSID_ENCODING(850) AS CCSID  
      FROM SYSIBM.SYSDUMMY1;
```

Example 3: The following function returns a CCSID with a value for Unicode data.

```
SELECT CCSID_ENCODING(1208) AS CCSID  
      FROM SYSIBM.SYSDUMMY1;
```

Example 4: The following function returns a CCSID with a value of UNKNOWN.

```
SELECT CCSID_ENCODING(1) AS CCSID  
      FROM SYSIBM.SYSDUMMY1;
```

CEIL or CEILING

CEIL or CEILING

```
►-- CEIL --► (—expression—) --►  
|   CEILING |
```

The schema is SYSIBM.

The CEIL or CEILING function returns the smallest integer value that is greater than or equal to the argument.

The argument is an expression that returns a value of any built-in numeric data type.

The result of the function has the same data type and length attribute as the argument except that the scale is 0 if the argument is DECIMAL. For example, an argument with a data type of DECIMAL(5,5) results in DECIMAL(05,0). The result can be null. If the argument is null, the result is the null value.

Example 1: The following statement shows the use of CEILING on positive and negative values:

```
SELECT CEILING(3.5), CEILING(3.1), CEILING(-3.1), CEILING(-3.5)  
      FROM SYSIBM.SYSDUMMY1;
```

This example returns: 04., 04., -03., -03.

Example 2: Using sample table DSN8710.EMP, find the highest monthly salary for all the employees. Round the result up to the next integer. The SALARY column has a decimal data type.

```
SELECT CEIL(MAX(SALARY)/12)  
      FROM DSN8710.EMP;
```

This example returns 04396. because the highest paid employee is Christine Haas who earns \$52750.00 per year. Her average monthly salary before applying the CEIL function is 4395.83.

CHAR

Datetime to Character:

```
►►—CHAR(datetime-expression) ——————
      , ——————
           | ISO
           | USA
           | EUR
           | JIS
           | LOCAL
```

String to Character:

```
►►—CHAR(string-expression) ——————
      , ——————
           | integer
```

Integer to Character:

```
►►—CHAR(integer-expression) ——————
```

Decimal to Character:

```
►►—CHAR(decimal-expression) ——————
      , ——————
           | decimal-character
```

Floating-Point to Character:

```
►►—CHAR(floating-point-expression) ——————
```

Row ID to Character:

```
►►—CHAR(row-ID-expression) ——————
```

The schema is SYSIBM.

The CHAR function returns a fixed-length character string representation of one of the following values:

- Datetime value if the first argument is a date, time, or timestamp
- String value if the first argument is any type of string
- Integer number if the first argument is a small or large integer
- Decimal number if the first argument is a decimal number
- Floating-point number if the first argument is a single or double precision floating-point number
- Row ID value if the first argument is a row ID

The result of the function is a fixed-length character string (CHAR).

CHAR

If the first argument can be null, the result can be null. If the first argument is null, the result is the null value.

Datetime to Character

datetime-expression

An expression whose value has one of the following three data types:

date The result is the character string representation of the date in the format that is specified by the second argument. If the second argument is omitted, the DATE precompiler option, if one is provided, or else field DATE FORMAT on installation panel DSNTIP4 specifies the format. If the format is to be LOCAL, field LOCAL DATE LENGTH on installation panel DSNTIP4 specifies the length of the result. Otherwise, the length of the result is 10.

LOCAL denotes the local format at the DB2 that executes the SQL statement. If LOCAL is used for the format, a date exit routine must be installed at that DB2.

An error occurs if the second argument is specified and is not a valid value.

time The result is the character string representation of the time in the format specified by the second argument. If the second argument is omitted, the TIME precompiler option, if one is provided, or else field TIME FORMAT on installation panel DSNTIP4 specifies the format. If the format is to be LOCAL, the field LOCAL TIME LENGTH on installation panel DSNTIP4 specifies the length of the result. Otherwise, the length of the result is 8.

LOCAL denotes the local format at the DB2 that executes the SQL statement. If LOCAL is used for the format, a time exit routine must be installed at that DB2.

An error occurs if the second argument is specified and is not a valid value.

timestamp

The result is the character string representation of the timestamp. The length of the result is 26. The second argument must not be specified.

The CCSID of the result is the SBCS CCSID of the appropriate encoding scheme.

String to Character

string-expression

An expression whose value is any type of string, except BLOB.

integer

The length attribute for the resulting fixed-length character string. The value must be between 1 and 255.

If the length is not specified, the length of the result is the same as the length of *string-expression*, which must be 255 bytes or less. If the string-expression is a graphic string, the length of the result is $(3 * \text{length}(\text{string-expression}))$. If *string-expression* is an empty string constant, an error occurs.

#

The actual length is the same as the length attribute of the result. If the length of *string-expression* is less than the length attribute of the result, the result is padded with blanks to the length of the result.

If *string-expression* is bit data, the result is bit data. Otherwise, the CCSID of the result is the same as the CCSID of *string-expression*.

Integer to Character

integer-expression

An expression whose value is an integer data type (SMALLINT or INTEGER).

The result is the fixed-length character string representation of the argument in the form of an SQL integer constant. The result consists of *n* characters that are the significant digits that represent the value of the argument with a preceding minus sign if the argument is negative. The result is left justified, and its length depends on whether the argument is a small or large integer:

- For a small integer, the length of the result is 6. If the number of characters in the result is less than 6, the result is padded on the right with blanks to a length of 6.
- For a large integer, the length of the result is 11; if the number of characters in the result is less than 11, the result is padded on right with blanks to a length of 11.

A positive value always includes one trailing blank.

The CCSID of the result is the SBCS CCSID of the appropriate encoding scheme.

Decimal to Character

decimal-expression

An expression whose value is a decimal data type. To specify a different precision and scale, you can use the DECIMAL scalar function first to make the change.

decimal-character

Specifies the single-byte character constant (CHAR or VARCHAR only) that is used to delimit the decimal digits in the resulting character string. Do not specify a digit, plus ('+'), minus ('-') or blank. The default is a period ('.') or comma (,). For information on what factors govern the choice, see “Options affecting SQL” on page 146.

The result is the fixed-length character string representation of the argument in the form of an SQL decimal constant. The result includes a decimal-character and *p* digits, where *p* is the precision of the *decimal-expression*. If the argument is negative, the first character of the result is a minus sign. Otherwise, the first character is a blank, which means that a positive value always has one leading blank.

The length of the result is $2+p$, where *p* is the precision of the *decimal-expression*.

The CCSID of the result is the SBCS CCSID of the appropriate encoding scheme.

Floating-Point to Character

floating-point-expression

An expression whose value is a floating-point data type (DOUBLE or REAL).

CHAR

The result is the fixed-length character string representation of the argument in the form of a floating-point constant. The length of the result is 24 bytes.

If the argument is negative, the first character of the result is a minus sign. Otherwise, the first character is a digit. If the value of the argument is zero, the result is 0E0. Otherwise, the result includes the smallest number of characters that can represent the value of the argument such that the mantissa consists of a single digit, other than zero, followed by a period and a sequence of digits.

If the number of characters in the result is less than 24, the result is padded on the right with blanks to length of 24.

The CCSID of the result is the SBCS CCSID of the appropriate encoding scheme.

Row ID to Character

row-ID-expression

An expression whose value is a row ID data type.

The result is the fixed-length character string representation of the argument. It is bit data and does not have an associated CCSID.

The length of the result is 40. If the length of *row-ID-expression* is less than 40, the result is padded on the right with hexadecimal zeroes to length of 40.

Example 1: HIREDATE is a DATE column in sample table DSN8710.EMP. When it represents 15 December 1976 (as it does for employee 140):

```
EXEC SQL SELECT CHAR(HIREDATE, USA)
  INTO :DATESTRING
  FROM DSN8710.EMP
 WHERE EMPNO = '000140';
```

returns the string value '12/15/1976' in character-string variable DATESTRING.

Example 2: Host variable HOUR has a data type of DECIMAL(6,0) and contains a value of 50000. Interpreted as a time duration, this value is 5 hours. Assume that STARTING is a TIME column in some table. Then, when STARTING represents 17 hours, 30 minutes, and 12 seconds after midnight:

```
CHAR(STARTING+:HOURS, USA)
```

returns the value '10:30 PM'.

Example 3: Assume that RECEIVED is defined as a TIMESTAMP column in table TABLEY. When the value of the date portion of RECEIVED represents 10 March 1997 and the time portion represents 6 hours and 15 seconds after midnight, this example:

```
SELECT CHAR(RECEIVED)
  FROM TABLEY
 WHERE INTCOL = 1234;
```

returns the string value '1997-03-10-06.00.15.000000'.

Example 4: For sample table DSN8710.EMP, the following SQL statement sets the host variable AVERAGE, which is defined as CHAR(33), to the character string representation of the average employee salary.

```
EXEC SQL SELECT CHAR(AVG(SALARY))
  INTO :AVERAGE
  FROM DSN8710.EMP;
```

With DEC31, the result of AVG applied to a decimal number is a decimal number with a precision of 31 digits. The only host languages in which such a large decimal variable can be defined are Assembler and C. For host languages that do not support such large decimal numbers, use the method shown in this example.

Example 5: For the rows in sample table DSN8710.EMP, return the values in column LASTNAME, which is defined as VARCHAR(15), as a fixed-length character string and limit the length of the results to 10 characters.

```
SELECT CHAR(LASTNAME,10)
  FROM DSN8710.EMP;
```

For rows that have a LASTNAME with a length greater than 10 characters (excluding trailing blanks), a warning that the value is truncated is returned.

Example 6: For the rows in sample table DSN8710.EMP, return the values in column EDLEVEL, which is defined as SMALLINT, as a fixed-length character string.

```
SELECT CHAR(EDLEVEL)
  FROM DSN8710.EMP;
```

An EDLEVEL of 18 is returned as CHAR(6) value '18 ' (18 followed by four blanks).

Example 7: In sample table DSN8710.EMP, the SALARY column is defined as DECIMAL(9,2). For those employees who have a salary of 52750.00, return the hire date and the salary, using a comma as the decimal character in the salary (52750,00).

```
#          SELECT HIREDATE, CHAR(SALARY, ',')
#          FROM DSN8710.EMP
#          WHERE SALARY = 52750.00;
```

The salary is returned as the string value ' 0052750,00'.

Example 8: Repeat the scenario in Example 7 except subtract the SALARY column from 60000.00 and return the salary with the default decimal character.

```
#          SELECT HIREDATE, CHAR (60000.00 - SALARY)
#          FROM DSN8710.EMP
#          WHERE SALARY = 52750.00;
```

The salary is returned as the string value ' 0007250.00'.

Example 9: Assume that host variable SEASONS_TICKETS is defined as INTEGER and has a value of 10000. Use the DECIMAL and CHAR functions to change the value into the character string ' 10000.00'.

```
SELECT CHAR(DECIMAL(:SEASONS_TICKETS,7,2))
  FROM SYSIBM.SYSDUMMY1;
```

Example 10: Assume that columns COL1 and COL2 in table T1 are both defined as REAL and that T1 contains a single row with the values 7.1E+1 and 7.2E+2 for the two columns. Add the two columns and represent the result as a character string.

```
SELECT CHAR(COL1 + COL2)
  FROM T1;
```

CHAR

The result is the character value '1.43E2'

'.

CLOB

```
►►—CLOB(string-expression [ ,integer ])—►►
```

The schema is SYSIBM.

The CLOB function returns a CLOB representation of a string.

string-expression

An expression whose value is a string. If *string-expression* is bit data, an error occurs.

integer

An integer value specifying the length attribute of the resulting CLOB data type. The value must be between 0 and the maximum length of a CLOB.

If you do not specify *integer*, the length attribute of the result is the same as the length attribute of *string-expression*. If the string-expression is a graphic string, the length of the result is $(3 * \text{length}(\text{string-expression}))$.

The result of the function is a CLOB. If the first argument can be null, the result can be null; if the first argument is null, the result is the null value.

The actual length of the result is the minimum of the length attribute of the result and the actual length of *string-expression*. If the length of *string-expression* is greater than the length specified, the result is truncated. Unless all of the truncated characters are blanks, a warning is returned.

The subtype and CCSID of the result are determined as follows:

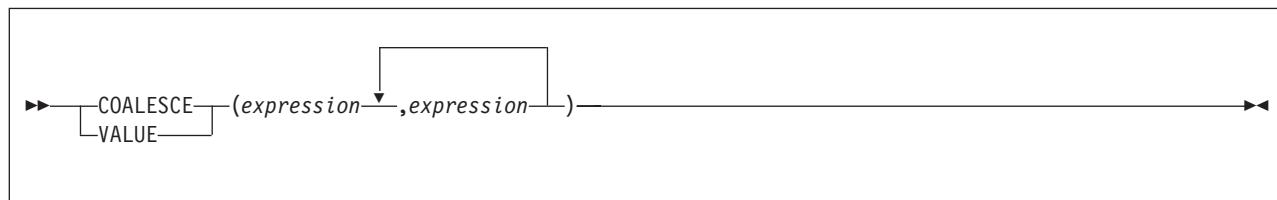
- If *string-expression* is character SBCS data, the result is SBCS data and the CCSID is CCSID for the encoding scheme of the SQL statement.
- If the first argument is mixed data, the result is mixed data and the CCSID is the CCSID for the encoding scheme of the SQL statement.

Example: The following function returns a CLOB for the string 'This is a CLOB'.

```
SELECT CLOB('This is a CLOB')
  FROM SYSIBM.SYSDUMMY1;
```

COALESCE

COALESCE



The schema is SYSIBM.

The COALESCE function returns the first argument that is not null. VALUE can be used as a synonym for COALESCE. Use COALESCE to conform to the SQL standard.

The arguments must be compatible. For more information on compatibility, refer to the compatibility matrix in Table 9 on page 65. The arguments can be of either a built-in or user-defined data type.²¹

The arguments are evaluated in the order in which they are specified, and the result of the function is the first argument that is not null. The result can be null only if all arguments can be null. The result is null only if all arguments are null.

The selected argument is converted, if necessary, to the attributes of the result. The attributes of the result are determined using the “Rules for result data types” on page 77. If the COALESCE function has more than two arguments, the rules are applied to the first two arguments to determine a candidate result type. The rules are then applied to that candidate result type and the third argument to determine another candidate result type. This process continues until all arguments are analyzed and the final result type is determined.

The COALESCE function can also handle a subset of the functions provided by CASE expressions. The result of using COALESCE(e1,e2) is the same as using the expression:

```
CASE WHEN e1 IS NOT NULL THEN e1 ELSE e2 END
```

Example 1: Assume that SCORE1 and SCORE2 are SMALLINT columns in table GRADES, and that nulls are allowed in SCORE1 but not in SCORE2. Select all the rows in GRADES for which SCORE1 + SCORE2 > 100, assuming a value of 0 for SCORE1 when SCORE1 is null.

```
SELECT * FROM GRADES  
WHERE COALESCE(SCORE1,0) + SCORE2 > 100;
```

Example 2: Assume that a table named DSN8710.EMP contains a DATE column named HIREDATE, and that nulls are allowed for this column. The following query selects all rows in DSN8710.EMP for which the date in HIREDATE is either unknown (null) or earlier than 1 January 1960.

```
SELECT * FROM DSN8710.EMP  
WHERE COALESCE(HIREDATE,DATE('1959-12-31')) < '1960-01-01';
```

21. This function cannot be used as a source function when creating a user-defined function. Because it accepts any compatible data types as arguments, it is not necessary to create additional signatures to support user-defined distinct types.

The predicate could also be coded as `VALUE(HIREDATE, '1959-12-31')` because for comparison purposes, a string representation of a date can be compared to a date.

Example 3: Assume that for the years 1993 and 1994 there is a table that records the sales results of each department. Each table, S1993 and S1994, consists of a DEPTNO column and a SALES column, neither of which can be null. The following query provides the sales information for both years.

```
SELECT COALESCE(S1993.DEPTNO,S1994.DEPTNO) AS DEPT, S1993.SALES, S1994.SALES  
FROM S1993 FULL JOIN S1994 ON S1993.DEPTNO = S1994.DEPTNO  
ORDER BY DEPT;
```

The full outer join ensures that the results include all departments, regardless of whether they had sales or existed in both years. The COALESCE function allows the two join columns to be combined into a single column, which enables the results to be ordered.

CONCAT

CONCAT

```
►-- CONCAT --►(expression1,expression2)---  
| | (1)  
"||"
```

Notes:

- 1 The vertical bars (||) must be surrounded by the SQL escape character in effect (" or ').

The schema is SYSIBM.

You can use either the keyword CONCAT keyword or vertical bars (||) to invoke the concatenation function. Vertical bars (or the characters that must be used in place of vertical bars in some countries) can cause parsing errors in statements passed from one DBMS to another. The problem occurs if the statement undergoes character conversion with certain combinations of source and target CCSIDs. Thus, CONCAT is the preferable syntax.

The CONCAT function, which is identical to the CONCAT operator, links two string arguments to form a string expression. The two arguments must be compatible strings. For more information on compatibility, refer to the compatibility matrix in Table 9 on page 65.

The result of the function is a string that consists of the first argument string followed by the second. If either argument can be null, the result can be null; if either is null, the result is the null value.

See “With the concatenation operator” on page 112 for more information.

Example: Using sample table DSN8710.EMP, concatenate column FIRSTNAME with column LASTNAME. Both columns are defined as varying-length character strings.

```
SELECT CONCAT(FIRSTNAME, LASTNAME)  
FROM DSN8710.EMP;
```

COS

```
►►COS(expression)►►
```

The schema is SYSIBM.

The COS function returns the cosine of the argument, where the argument is an angle expressed in radians. The COS and ACOS functions are inverse operations.

The argument is an expression that returns the value of any built-in numeric data type. If the argument is not a double precision floating-point number, it is converted to one for processing by the function.

The result of the function is a double precision floating-point number. The result can be null; if the argument is null, the result is the null value.

Example: Assume that host variable COSINE is DECIMAL(2,1) with a value of 1.5. The following statement:

```
SELECT COS(:COSINE)
      FROM SYSIBM.SYSDUMMY1;
```

returns a double precision floating-point number with an approximate value of 0.07.

COSH

COSH

```
►►COSH(expression)►►
```

The schema is SYSIBM.

The COSH function returns the hyperbolic cosine of the argument, where the argument is an angle expressed in radians.

The argument is an expression that returns the value of any built-in numeric data type. If the argument is not a double precision floating-point number, it is converted to one for processing by the function.

The result of the function is a double precision floating-point number. The result can be null; if the argument is null, the result is the null value.

Example: Assume that host variable HCOS is DECIMAL(2,1) with a value of 1.5. The following statement:

```
SELECT COSH(:HCOS)
   FROM SYSIBM.SYSDUMMY1;
```

returns a double precision floating-point number with an approximate value of 2.35.

DATE

►►—DATE(*expression*)—►►

The schema is SYSIBM.

The DATE function returns a date derived from its argument.

The argument must be a date, a timestamp, a valid string representation of a date or timestamp, a positive number with a built-in data type that is less than or equal to 3652059, or a string of length 7. An argument with a string data type must not be a BLOB, CLOB, or DBCLOB and must have an actual length that is not greater than 255 bytes. For the valid formats of string representations of dates and timestamps, see “String representations of datetime values” on page 57.

If the argument is a string, the result is the date represented by the string. If the CCSID of the string is not the same as the corresponding default CCSID at the server, the string is first converted to that CCSID.

The result of the function is a date. If the argument can be null, the result can be null; if the argument is null, the result is the null value.

The other rules depend on the data type of the argument:

If the argument is a timestamp, the result is the date part of the timestamp.

If the argument is a date, the result is that date.

If the argument is a number, the result is the date that is *n*-1 days after January 1, 0001, where *n* is the integral part of the number.

If the argument is a string, the result is the date represented by the string. If the CCSID of the string is not the same as the corresponding default CCSID at the server, the string is first converted to that CCSID.

Example 1: Assume that RECEIVED is a TIMESTAMP column in some table, and that one of its values is equivalent to the timestamp '1988-12-25-17.12.30.000000'. Then, for this value:

DATE(RECEIVED)

returns the internal representation of 25 December 1988.

Example 2: Assume that DATCOL is a CHAR(7) column in some table, and that one of its values is the character string '1989061'. Then, for this value:

DATE(DATCOL)

returns the internal representation of 2 March 1989.

Example 3: DB2 recognizes '1989-03-02' as the ISO representation of 2 March 1989. Therefore:

DATE('1989-03-02')

returns the internal representation of 2 March 1989.

DAY

DAY

```
►►DAY(expression)►►
```

The schema is SYSIBM.

The DAY function returns the day part of its argument.

The argument must be a date, timestamp, date duration, timestamp duration, or valid string representation of a date or timestamp. A string representation must not be a BLOB, CLOB, or DBCLOB and must have an actual length that is not greater than 255 bytes. For the valid formats of string representations of dates and timestamps, see “String representations of datetime values” on page 57.

The result of the function is a large integer. If the argument can be null, the result can be null; if the argument is null, the result is the null value.

The other rules for the function depend on the data type of the argument:

If the argument is a date, timestamp, or string representation of either, the result is the day part of the value, which is an integer between 1 and 31.

If the argument is a date duration or timestamp duration, the result is the day part of the value, which is an integer between -99 and 99. A nonzero result has the same sign as the argument.

Example 1: Set the INTEGER host variable DAYVAR to the day of the month on which employee 140 in the sample table DSN8710.EMP was hired.

```
EXEC SQL SELECT DAY(HIREDATE)
  INTO :DAYVAR
  FROM DSN8710.EMP
 WHERE EMPNO = '000140';
```

Example 2: Assume that DATE1 and DATE2 are DATE columns in the same table. Assume also that for a given row in this table, DATE1 and DATE2 represent the dates 15 January 2000 and 31 December 1999, respectively. Then, for the given row:

```
DAY(DATE1 - DATE2)
```

returns the value 15.

DAYOFMONTH

```
►►DAYOFMONTH(expression)►►
```

The schema is SYSIBM.

The DAYOFMONTH function returns the day part of its argument. The function is similar to the DAY function, except DAYOFMONTH does not support a date or timestamp duration as an argument.

The argument must be a date, a timestamp, or a valid string representation of a date or timestamp. A string representation must not be a BLOB, CLOB, or DBCLOB and must have an actual length that is not greater than 255 bytes. For the valid formats of string representations of dates and timestamps, see “String representations of datetime values” on page 57.

The result of the function is a large integer between 1 and 31, which represents the day part of the value. If the argument can be null, the result can be null; if the argument is null, the result is the null value.

Example: Set the INTEGER variable DAYVAR to the day of the month on which employee 140 in sample table DSN8710.EMP was hired.

```
SELECT DAYOFMONTH(HIREDATE)
  INTO :DAYVAR
  FROM DSN8710.EMP
 WHERE EMPNO = '000140';
```

DAYOFWEEK

DAYOFWEEK

```
►►DAYOFWEEK(expression)►►
```

The schema is SYSIBM.

The DAYOFWEEK function returns an integer in the range of 1 to 7 that represents the day of the week where 1 is Sunday and 7 is Saturday.

The argument must be a date, a timestamp, or a valid string representation of a date or timestamp. A string representation must not be a BLOB, CLOB, or DBCLOB and must have an actual length that is not greater than 255 bytes. For the valid formats of string representations of dates and timestamps, see “String representations of datetime values” on page 57.

The result of the function is a large integer. If the argument can be null, the result can be null; if the argument is null, the result is the null value.

Example 1: Using sample table DSN8710.EMP, set the integer host variable DAY_OF_WEEK to the day of the week that Christine Haas (EMPNO = '000010') was hired (HIREDATE).

```
SELECT DAYOFWEEK(HIREDATE)
      INTO :DAY_OF_WEEK
     FROM DSN8710.EMP
    WHERE EMPNO = '000010';
```

The result is that DAY_OF_WEEK is set to 6, which represents Friday.

Example 2: The following query returns four values: 1, 2, 1, and 2.

```
SELECT DAYOFWEEK(CAST('10/11/1998' AS DATE)),
       DAYOFWEEK(TIMESTAMP('10/12/1998', '01.02')),
       DAYOFWEEK(CAST(CAST('10/11/1998' AS DATE) AS CHAR(20))),
       DAYOFWEEK(CAST(TIMESTAMP('10/12/1998', '01.02') AS CHAR(20)))
  FROM SYSIBM.SYSDUMMY1;
```

DAYOFWEEK_ISO

►►—DAYOFWEEK_ISO(*expression*)—►►

The schema is SYSIBM.

The DAYOFWEEK_ISO function returns an integer in the range of 1 to 7 that represents the day of the week, where 1 is Monday and 7 is Sunday.

The argument must be a date, a timestamp, or a valid string representation of a date or timestamp. A string representation must not be a BLOB, CLOB, or DBCLOB and must have an actual length that is not greater than 255 bytes. For the valid formats of string representations of dates and timestamps, see “String representations of datetime values” on page 57.

The result of the function is a large integer. If the argument can be null, the result can be null; if the argument is null, the result is the null value.

Example 1: Using sample table DSN8710.EMP, set the integer host variable DAY_OF_WEEK to the day of the week that Christine Haas (EMPNO = '000010') was hired (HIREDATE).

```
SELECT DAYOFWEEK_ISO(HIREDATE)
  INTO :DAY_OF_WEEK
  FROM DSN8710.EMP
 WHERE EMPNO = '000010';
```

The result is that DAY_OF_WEEK is set to 5, which represents Friday.

Example 2: The following query returns four values: 7, 1, 7, and 1.

```
SELECT DAYOFWEEK_ISO(CAST('10/11/1998' AS DATE)),
       DAYOFWEEK_ISO(TIMESTAMP('10/12/1998', '01.02')),
       DAYOFWEEK_ISO(CHAR(20) CAST(CAST('10/11/1998' AS DATE) AS CHAR(20))),
       DAYOFWEEK_ISO(CHAR(20) CAST(TIMESTAMP('10/12/1998', '01.02') AS CHAR(20)))
  FROM SYSIBM.SYSDUMMY1;
```

Example 3: The following list shows what is returned by the DAYOFWEEK_ISO function for various dates.

DATE	DAYOFWEEK_ISO
1997-12-28	'7'
1997-12-31	'3'
1998-01-01	'4'
1999-01-01	'5'
1999-01-04	'1'
1999-12-31	'5'
2000-01-01	'6'
2000-01-03	'1'

DAYOFYEAR

DAYOFYEAR

```
►►DAYOFYEAR(expression)
```

The schema is SYSIBM.

The DAYOFYEAR function returns an integer in the range of 1 to 366 that represents the day of the year where 1 is January 1.

The argument must be a date, a timestamp, or a valid string representation of a date or timestamp. A string representation must not be a BLOB, CLOB, or DBCLOB and must have an actual length that is not greater than 255 bytes. For the valid formats of string representations of dates and timestamps, see “String representations of datetime values” on page 57.

The result of the function is a large integer. If the argument can be null, the result can be null; if the argument is null, the result is the null value.

Example: Using sample table DSN8710.EMP, set the integer host variable AVG_DAY_OF_YEAR to the average of the day of the year on which employees were hired (HIREDATE):

```
SELECT AVG(DAYOFYEAR(HIREDATE))
  INTO :AVG_DAY_OF_YEAR
    FROM DSN8710.EMP;
```

The result is that AVG_DAY_OF_YEAR is set to 202.

DAYS

```
►►—DAYS(expression)—►►
```

The schema is SYSIBM.

The DAYS function returns an integer representation of a date.

The argument must be a date, a timestamp, or a valid string representation of a date or timestamp. A string representation must not be a BLOB, CLOB, or DBCLOB and must have an actual length that is not greater than 255 bytes. For the valid formats of string representations of dates and timestamps, see “String representations of datetime values” on page 57.

The result of the function is a large integer. If the argument can be null, the result can be null. If the argument is null, the result is the null value.

The result is 1 more than the number of days from January 1, 0001 to D , where D is the date that would occur if the DATE function were applied to the argument.

Example: Set the INTEGER host variable DAYSVAR to the number of days that employee 140 had been with the company on the last day of 1997.

```
EXEC SQL SELECT DAYS('1997-12-31') - DAYS(HIREDATE) + 1  
      INTO :DAYSVAR  
     FROM DSN8710.EMP  
    WHERE EMPNO = '000140';
```

DBCLOB

DBCLOB

```
►►DBCLOB(string-expression [ ,-integer ])►►
```

The schema is SYSIBM.

| The DBCLOB function returns a DBCLOB representation of a string type.

| The length of the result is measured in double-byte characters.

| *string-expression*

| An expression whose value is a string. The expression cannot be character
FOR BIT DATA. The value must not be an empty string constant.

| *integer*

| An integer value specifying the length attribute of the resulting DBCLOB. The
value must be between 0 and the maximum length of a DBCLOB.

| If you do not specify *integer*, the length attribute of the result is the same as the
length attribute of *string-expression*.

The result of the function is a DBCLOB. If the first argument can be null, the result
can be null; if the first argument is null, the result is the null value.

The actual length of the result is the minimum of the length attribute of the result
and the actual length of *string-expression*. If the length of *string-expression* is
greater than the length specified, the result is truncated. Unless all of the truncated
characters are double-byte blanks, a warning is returned.

| The CCSID of the result is the same as the CCSID of *string-expression*.

Example: Assume that the application encoding scheme is Unicode. The following
statement returns a graphic (UTF-16) host variable.

```
VALUES DBCLOB('123')  
      INTO :GHV1;
```

DECIMAL or DEC

Numeric to Decimal:

```
►--DECIMAL--(numeric-expression-->,<precision-integer-->,<scale-integer-->)-->
```

Character to Decimal:

```
►--DECIMAL--(string-expression-->,<precision-integer-->,<scale-integer-->,<decimal-character-->)-->
```

The schema is SYSIBM.

The DECIMAL or DEC function returns a decimal representation of a number or character string in the form of a numeric constant.

numeric-expression

An expression that returns a value of any built-in numeric data type.

The result of the function is a decimal number. The result is the same number that would occur if the argument were assigned to a decimal column or variable with precision p and scale s , where p and s are specified by the second and third arguments.

string-expression

An expression that returns any type of string (except a BLOB, CLOB, or DBCLOB) with a maximum length that is not greater than 255 bytes. Leading and trailing blanks are removed from the string, and the resulting substring must conform to the rules for forming a string representation of an SQL integer or decimal constant.

The result of the function is a decimal number. The result is the same number that would occur if the corresponding integer or decimal constant were assigned to a decimal column or variable with precision p and scale s , where p and s are specified by the second and third arguments.

precision-integer

An integer constant with a value in the range of 1 to 31. The value of this second argument specifies the precision of the result.

The default value depends on the data type of the first argument as follows:

- 5 if the first argument is a small integer
- 11 if the first argument is a large integer
- 15 in all other cases

scale-integer

An integer constant with a value in the range of 1 to p , where p is the value of the second argument. The value of this third argument specifies the scale of the result. The default value is 0.

decimal-character

A single-byte character constant used to delimit the decimal digits in

DECIMAL or DEC

| *string-expression* from the whole part of the number. The character cannot be a
| digit, plus (+), minus (-), or blank. The default value is period (.) or comma (,);
| the default value cannot be used in *string-expression* if a different value for
| *decimal-character* is specified.

The data type of the result is DECIMAL(*p,s*), where *p* and *s* are the second and third arguments. If the first argument can be null, the result can be null; if the first argument is null, the result is null.

An error occurs if the number of significant digits required to represent the whole part of the number is greater than *p-s*.

Example 1: Represent the average salary of the employees in DSN8710.EMP as an 8-digit decimal number with two of these digits to the right of the decimal point.

```
SELECT DECIMAL(AVG(SALARY),8,2)  
      FROM DSN8710.EMP;
```

Example 2: Assume that updates to the SALARY column are input as a character string that uses comma as the decimal character. For example, the user inputs 21400,50. The input value is assigned to the host variable NEWSALARY that is defined as CHAR(10), and the host variable is used in the following UPDATE statement:

```
UPDATE DSN8710.EMP  
      SET SALARY = DECIMAL (:NEWSALARY,9,2,',')  
      WHERE EMPNO = :EMPID;
```

DEGREES

```
►►—DEGREES(expression)—►►
```

The schema is SYSIBM.

The DEGREES function returns the number of degrees converted from the argument expressed in radians.

The argument is an expression that returns the value of any built-in numeric data type. If the argument is not a double precision floating-point number, it is converted to one for processing by the function.

The result of the function is a double precision floating-point number. The result can be null; if the argument is null, the result is the null value.

Example: Assume that host variable HRAD is a DOUBLE with a value of 3.1415926536. The following statement:

```
SELECT DEGREES(:HRAD)
      FROM SYSIBM.SYSDUMMY1;
```

returns a double precision floating-point number with an approximate value of 180.0.

DIGITS

DIGITS

```
►►DIGITS(expression)—►►
```

The schema is SYSIBM.

The DIGITS function returns a character string representation of its argument.

The argument must be a built-in exact numeric data type of SMALLINT, INTEGER, or DECIMAL.

If the argument can be null, the result can be null; if the argument is null, the result is the null value.

The result of the function is a fixed-length character string representing the absolute value of the argument without regard to its scale. The result does not include a sign or a decimal point. Instead, it consists exclusively of digits, including, if necessary, leading zeros to fill out the string. The length of the string is:

- 5 if the argument is a small integer
- 10 if the argument is a large integer
- p if the argument is a decimal number with a precision of p

Example 1: Assume that an INTEGER column called INTCOL containing a 10-digit number is in a table called TABLEX. INTCOL has the data type INTEGER instead of CHAR(10) to save space. List all combinations of the first four digits in column INTCOL.

```
SELECT DISTINCT SUBSTR(DIGITS(INTCOL),1,4)  
  FROM TABLEX;
```

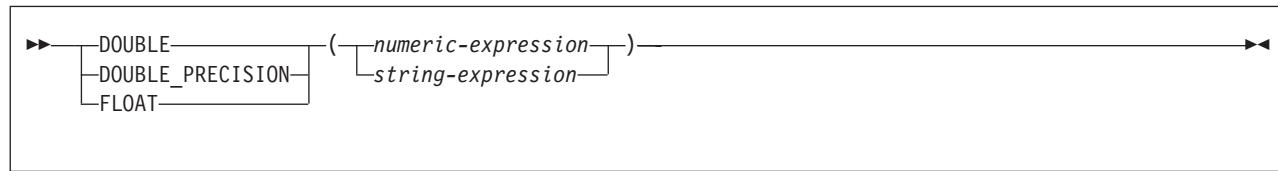
Example 2: Assume that COLUMNX has the data type DECIMAL(6,2), and that one of its values is -6.28. Then, for this value:

```
DIGITS(COLUMNX)
```

the value '000628' is returned.

The result is a string of length six (the precision of the column) with leading zeros padding the string out to this length. Neither sign nor decimal point appear in the result.

DOUBLE or DOUBLE PRECISION



The schema is SYSIBM.

The DOUBLE or DOUBLE_PRECISION function returns a double precision floating-point representation of a number or character string in the form of a numeric constant. FLOAT is a synonym for DOUBLE or DOUBLE_PRECISION.

numeric-expression

An expression that returns a value of any built-in numeric data type.

The result of the function is a double precision floating-point number. The result is the same number that would occur if the expression were assigned to a double precision floating-point column or variable.

string-expression

An expression that returns any type of string (except a BLOB, CLOB, or DBCLOB) with an actual length that is not greater than 255 bytes. Leading and trailing blanks are removed from the string, and the resulting substring must conform to the rules for forming a string representation of an SQL floating-point constant.

The result of the function is a double precision floating-point number. The result is the same number that would occur if the corresponding numeric constant were assigned to a double precision floating-point column or variable.

If the argument can be null, the result can be null; if the argument is null, the result is the null value.

Example: Using sample table DSN8710.EMP, find the ratio of salary to commission for employees whose commission is not zero. The columns involved in the calculation, SALARY and COMM, have decimal data types. To eliminate the possibility of out-of-range results, apply the DOUBLE function to SALARY so that the division is carried out in floating-point.

```
SELECT EMPNO, DOUBLE(SALARY)/COMM  
      FROM DSN8710.EMP  
 WHERE COMM > 0;
```

EXP

EXP

```
►►EXP(expression)—►►
```

The schema is SYSIBM.

The EXP function returns the exponential function of the argument (a value that is the base of the natural logarithm (e) raised to a power specified by the argument). The EXP and LN functions are inverse operations.

The argument is an expression that returns the value of any built-in numeric data type. If the argument is not a double precision floating-point number, it is converted to one for processing by the function.

The result of the function is a double precision floating-point number. The result can be null; if the argument is null, the result is the null value.

Example: Assume that host variable E is DECIMAL(10,9) with a value of 3.453789832. The following statement:

```
SELECT EXP(:E)
      FROM SYSIBM.SYSDUMMY1;
```

returns a double precision floating-point number with an approximate value of 31.62.

FLOAT

```
►►FLOAT(expression)►►
```

The schema is SYSIBM.

The FLOAT function returns a floating-point representation of its argument.

FLOAT is a synonym for the DOUBLE function. See “DOUBLE or DOUBLE_PRECISION” on page 209 for details.

FLOOR

FLOOR

```
►►FLOOR(expression)─
```

The schema is SYSIBM.

The FLOOR function returns the largest integer value that is less than or equal to the argument.

The argument is an expression that returns a value of any built-in numeric data type.

The result of the function has the same data type and length attribute as the argument. The result can be null. If the argument is null, the result is the null value. When the argument is DECIMAL, the scale of the result is 0 and not the scale of the input argument.

Example: Using sample table DSN8710.EMP, find the highest monthly salary, rounding the result down to the next integer. The SALARY column has a decimal data type.

```
SELECT FLOOR(MAX(SALARY)/12)
      FROM DSN8710.EMP;
```

This example returns 04395 because the highest paid employee is Christine Haas who earns \$52750.00 per year. Her average monthly salary before applying the FLOOR function is 4395.83.

GENERATE_UNIQUE

#

►►—GENERATE_UNIQUE()————►►

#

#

#

The schema is SYSIBM.

The GENERATE_UNIQUE function returns a bit data character string 13 bytes long (CHAR(13) FOR BIT DATA) that is unique compared to any other execution of the same function. The function is defined as not-deterministic. Although the function has no arguments, the empty parentheses must be specified.

The result of the function is a unique value that includes the internal form of the Universal Time, Coordinated (UTC) and, if in a sysplex environment, the sysplex member where the function was processed. The result cannot be null.

The result of this function can be used to provide unique values in a table. Each successive value will be greater than the previous value, providing a sequence that can be used within a table. If applicable, the value also includes information that identifies the sysplex member where the function was executed. The sequence is based on the time when the function was executed.

This function differs from using the special register CURRENT TIMESTAMP in that a unique value is generated for each row of a multiple row insert statement or an insert statement with a fullselect.

The timestamp value that is part of the result of this function can be determined using the TIMESTAMP scalar function with the result of GENERATE_UNIQUE as an argument.

Example: Create a table that includes a column that is unique for each row. Populate this column using the GENERATE_UNIQUE function. Notice that the UNIQUE_ID column is defined as FOR BIT DATA to identify the column as a bit data character string.

```
# CREATE TABLE EMP_UPDATE
#   (UNIQUE_ID VARCHAR(13)FOR BIT DATA,
#    EMPNO CHAR(6),
#    TEXT VARCHAR(1000));
#
# INSERT INTO EMP_UPDATE
#   VALUES (GENERATE_UNIQUE(),'000020','Update entry 1...');

# INSERT INTO EMP_UPDATE
#   VALUES (GENERATE_UNIQUE(),'000050','Update entry 2...');
```

This table will have a unique identifier for each row if GENERATE_UNIQUE is always used to set the value of the UNIQUE_ID column. You can create a trigger on the table to ensure that GENERATE_UNIQUE is used to set the value:

```
# CREATE TRIGGER EMP_UPDATE_UNIQUE
#   NO CASCADE BEFORE INSERT ON EMP_UPDATE
#   REFERENCING NEW AS NEW_UPD
#   FOR EACH ROW MODE DB2SQL
#   SET NEW_UPD.UNIQUE_ID = GENERATE_UNIQUE();
```

GENERATE_UNIQUE

```
# With this trigger, the previous INSERT statements that were used to populate the
# table could be issued without specifying a value for the UNIQUE_ID column:
#
#     INSERT INTO EMP_UPDATE (EMPNO,TEXT)
#             VALUES ('000020','Update entry 1...');

#     INSERT INTO EMP_UPDATE (EMPNO,TEXT)
#             VALUES ('000050','Update entry 2...');
```

The timestamp (in UTC) for when a row was added to EMP_UPDATE can be
returned using:

```
#     SELECT TIMESTAMP(UNIQUE_ID),EMPNO,TEXT FROM EMP_UPDATE;
```

Therefore, the table does not need a timestamp column to record when a row is
inserted.

GRAPHIC

Character to Graphic:

```
►►—GRAPHIC(character-expression)  
          └,—integer┘————→
```

Graphic to Graphic:

```
►►—GRAPHIC(graphic-expression)  
          └,—integer┘————→
```

The schema is SYSIBM.

The GRAPHIC function returns a graphic representation of a character string value, with the single-byte characters converted to double-byte characters, or a graphic string value if the first argument is a graphic string.

The result of the function is a fixed-length graphic string (GRAPHIC).

If the argument can be null, the result can be null; if the argument is null, the result is the null value.

The length attribute of the result is measured in double-byte characters because it is a graphic string.

Character to Graphic

character-expression

An expression whose value must be an EBCDIC-encoded or Unicode-encoded character string. The GRAPHIC function is not allowed for ASCII data. The argument does not need to be mixed data, but any occurrences of X'OE' and X'OF' in the string must conform to the rules for EBCDIC mixed data. (See “Character strings” on page 49 for these rules.)

The value of the expression must not be an empty string if *integer* is not specified or have the value X'0EOF' if the string is an EBCDIC string.

integer

The length of the resulting fixed-length graphic string. The value must be an integer between 1 and 127. If the length of *character-expression* is less than the length specified, the result is padded with double-byte blanks to the length of the result.

| If *integer* is not specified, the length of the result for an EBCDIC string is the minimum of 127 and the length attribute of *character-expression*, excluding shift characters. For a Unicode (UTF-8) string, the length is data dependent, but does not exceed 127.

| The CCSID of the result is the system CCSID for EBCDIC or Unicode GRAPHIC data. If the input is EBCDIC and there is no system CCSID for EBCDIC GRAPHIC data, the CCSID of the result is X'FFFE'.

GRAPHIC

| For EBCDIC data:

Each character of *character-expression* determines a character of the result. The argument might need to be converted to the native form of mixed data before the result is derived. Let M be the system CCSID for mixed data. The argument is not converted if any of the following conditions is true:

- The argument is mixed data and its CCSID is M.
- The argument is SBCS data and its CCSID is the same as the system CCSID for SBCS data. In this case, the operation proceeds as if the CCSID of the argument is M.
- The argument cannot be BIT data.

Otherwise, the argument is a new string S derived by converting the characters to the coded character set identified by M. If there is no system CCSID for EBCDIC mixed data, conversion is to the coded character set that the system EBCDIC CCSID for SBCS data identifies.

The result is derived from S using the following steps:

- Each shift character (X'0E' or X'0F') is removed.
- Each double-byte character remains as is.
- Each single-byte character is replaced by a double-byte character.

The replacement for an SBCS character is the equivalent DBCS character if an equivalent exists. Otherwise, the replacement is X'FEFE'. The existence of an equivalent character depends on M. If there is no system CCSID for mixed data, the DBCS equivalent of X'xx' for EBCDIC is X'42xx', except for X'40', whose DBCS equivalent is X'4040'.

| For Unicode data:

| Each character of *character-expression* determines a character of the result. The argument might need to be converted to the native form of mixed data before the result is derived. Let M be the system CCSID for mixed data. The argument is not converted if any of the following conditions is true:

- The argument is mixed data, and its CCSID is M.
- The argument is SBCS data, and its CCSID is the same as the system CCSID for SBCS data. In this case, the operation proceeds as if the CCSID of the argument is M.

| Otherwise, the argument is a new string S derived by converting the characters to the coded character set identified by M.

| The result is derived from S by using the following steps:

- Each non-surrogate character is replaced by a Unicode double-byte character (a UTF-16 code point). A non-surrogate character in UTF-8 is between 1 and 3 bytes.
- Each surrogate character is replaced by a pair of Unicode double-byte characters (a pair of UTF-16 code points).

| The replacement for a single-byte character is the Unicode equivalent character if an equivalent exists. Otherwise, the replacement is X'FEFE'.

Graphic to Graphic

graphic-expression

An expression whose value is a graphic string. The graphic string must not be an empty string if *integer* is not specified.

integer

The length of the resulting fixed-length graphic string. The value must be an integer between 1 and 127. If the length of *graphic-expression* is less than the length specified, the result is padded with double-byte blanks to the length of the result.

If *integer* is not specified, the length of the result is the minimum of 127 and the length attribute of *graphic-expression*. The graphic string must not be an empty string if *integer* is not specified.

If the length of the *graphic-expression* is greater than the specified length of the result, the result is truncated. Unless all the truncated characters are blanks, a warning is returned.

The CCSID of the result is the same as the CCSID of *graphic-expression*.

Example: Assume that MYCOL is a VARCHAR column in TABLEY. The following function returns the string in MYCOL as a fixed-length graphic string.

```
SELECT GRAPHIC(MYCOL)
      FROM TABLEY;
```

HEX

HEX

```
►►HEX(expression)►►
```

The schema is SYSIBM.

The HEX function returns a hexadecimal representation of its argument.

The argument can be of any built-in data type other than a character or binary string with a maximum length greater than 255 or a graphic string with a maximum length greater than 127.

The result of the function is a character string. If the argument can be null, the result can be null; if the argument is null, the result is the null value.

The result is a string of hexadecimal digits. The first two represent the first byte of the argument, the next two represent the second byte of the argument, and so forth. If the argument is a datetime value, the result is the hexadecimal representation of the internal form of the argument.

If the argument is a graphic string, the length of the result is four times the maximum length of the argument. Otherwise, the length of the result is twice the (maximum) length of the argument.

If the argument is not a varying-length string, and the length of the result is less than 255, the result is a fixed-length string. Otherwise, the result is a varying-length string whose maximum length depends on the following considerations:

If the argument is not a varying-length string, the maximum length of the result string is the same as the length of the result.

If the argument is a varying-length character or binary string, the maximum length of the result string is twice the maximum length of the argument.

If the argument is a varying-length graphic string, the maximum length of the result string is four times the maximum length of the argument.

If the maximum length of the result is greater than 254 bytes, the result is subject to the restrictions that apply to long strings.

Example: Return the hexadecimal representation of START_RBA in the SYSIBM.SYSCOPY catalog table.

```
SELECT HEX(START_RBA) FROM SYSIBM.SYSCOPY;
```

HOUR

```
►► HOUR(expression) ──────────────────────────────────►►
```

The schema is SYSIBM.

The HOUR function returns the hour part of its argument.

The argument must be a time, timestamp, time duration, timestamp duration, or valid string representation of a time or timestamp. A string representation must not be a BLOB, CLOB, or DBCLOB and must have an actual length that is not greater than 255 bytes. For the valid formats of string representations of times and timestamps, see “String representations of datetime values” on page 57.

The result of the function is a large integer. If the argument can be null, the result can be null; if the argument is null, the result is the null value.

The other rules depend on the data type of the argument:

If the argument is a time, timestamp, or string representation of either, the result is the hour part of the value, which is an integer between 0 and 24.

If the argument is a time duration or timestamp duration, the result is the hour part of the value, which is an integer between -99 and +99. A nonzero result has the same sign as the argument.

Example: Assume that a table named CLASSES contains a row for each scheduled class. Also assume that the class starting times are in a TIME column named STARTTM. Select those rows in CLASSES that represent classes that start after the noon hour.

```
SELECT *  
  FROM CLASSES  
 WHERE HOUR(STARTTM) > 12;
```

IDENTITY_VAL_LOCAL

IDENTITY_VAL_LOCAL

```
►►►IDENTITY_VAL_LOCAL()►►►
```

The schema is SYSIBM.

The **IDENTITY_VAL_LOCAL** function is a nondeterministic function ²² that returns the most recently assigned value for an identity column. The function has no input parameters.

The result is DECIMAL(31,0), regardless of the actual data type of the identity column that the result value corresponds to.

The value returned is the value that was assigned to the identity column of the table identified in the most recent row **INSERT** statement with a **VALUES** clause for a table with an identity column. The **INSERT** statement has to be issued at the same level; that is, the value has to be available *locally* within the level at which it was assigned until replaced by the next assigned value. A new level is initiated when a trigger, function, or stored procedure is invoked. A trigger condition is at the same level as the associated triggered action.

The assigned value can be a value supplied by the user (if the identity column is defined as **GENERATED BY DEFAULT**) or an identity value that was generated by DB2.

The function returns the null value in the following situations:

- When a single row **INSERT** statement with a **VALUES** clause has not been issued for a table containing an identity column at the current processing level
- When a **COMMIT** or **ROLLBACK** of a unit of work occurred since the most recent **INSERT** statement that assigned a value

The result of the function is not affected by the following statements:

- An **INSERT** statement with a **VALUES** clause for a table that does not contain an identity column
- An **INSERT** statement with a subselect
- A **ROLLBACK TO SAVEPOINT** statement

Notes

The following notes explain the behavior of the function when it is invoked in various situations:

Invoking the function within the VALUES clause of an INSERT statement

Expressions in the **VALUES** clause of an **INSERT** statement are evaluated before values are assigned to the target columns of the **INSERT** statement. Thus, when you invoke **IDENTITY_VAL_LOCAL** in a **VALUES** clause of an **INSERT** statement, the value that is used is the most recently assigned value for an identity column from a previous **INSERT** statement. The function returns the null value if no such **INSERT** statement had been

22. Being nondeterministic affects what optimization (such as view processing and parallel processing) can be done when this function is used and in what contexts the function can be invoked. For example, the **RAND** function is another built-in scalar that is not deterministic. Using nondeterministic functions within a predicate can cause unpredictable results.

executed within the same level as the invocation of the IDENTITY_VAL_LOCAL function. Each INSERT statement involving an IDENTITY column causes the identity value to be copied into connection-specific storage in DB2. Thus, the most recent identity value is used for a connection, regardless of what is happening with other concurrent user connections.

Invoking the function following a failed INSERT statement

The function returns an unpredictable result when it is invoked after the unsuccessful execution of a single row INSERT with a VALUES clause for a table with an identity column. The value might be the value that would have been returned from the function had it been invoked before the failed INSERT or the value that would have been assigned had the INSERT succeeded. The actual value returned depends on the point of failure and is therefore unpredictable.

Invoking the function within the SELECT statement of a cursor

Because the results of the IDENTITY_VAL_LOCAL function are not deterministic, the result of an invocation of the IDENTITY_VAL_LOCAL function from within the SELECT statement of a cursor can vary for each FETCH statement.

Invoking the function within the trigger condition of an insert trigger

The result of invoking the IDENTITY_VAL_LOCAL function from within the condition of an insert trigger is the null value.

Invoking the function within a triggered action of an insert trigger

Multiple before or after insert triggers can exist for a table. In such cases, each trigger is processed separately, and identity values generated by SQL statements issued within a triggered action are not available to other triggered actions using the IDENTITY_VAL_LOCAL function. This is the case even though the multiple triggered actions are conceptually defined at the same level.

Do not use the IDENTITY_VAL_LOCAL function in the triggered action of a before insert trigger. The result of invoking the IDENTITY_VAL_LOCAL function from within the triggered action of a before insert trigger is the null value. The value for the identity column of the table for which the trigger is defined cannot be obtained by invoking the IDENTITY_VAL_LOCAL function within the triggered action of a before insert trigger. However, the value for the identity column can be obtained in the triggered action by referencing the trigger transition variable for the identity column.

The result of invoking the IDENTITY_VAL_LOCAL function in the triggered action of an after insert trigger is the value assigned to an identity column of the table identified in the most recent single row INSERT statement. That statement is the one invoked in the same triggered action that had a VALUES clause for a table containing an identity column. If a single row INSERT statement with a VALUES clause for a table containing an identity column was not executed within the same triggered action before invoking the IDENTITY_VAL_LOCAL function, then the function returns a null value.

Invoking the function following an INSERT with triggered actions

The result of invoking the function after an INSERT that activates triggers is the value actually assigned to the identity column (that is, the value that would be returned on a subsequent SELECT statement). This value is not necessarily the value provided in the VALUES clause of the INSERT statement or a value generated by DB2. The assigned value could be a

IDENTITY_VAL_LOCAL

value that was specified in a SET transition variable statement within the triggered action of a before insert trigger for a trigger transition variable associated with the identity column.

Examples

Example 1: Set the variable IVAR to the value assigned to the identity column in the EMPLOYEE table. The value returned from the function in the VALUES statement should be 1.

```
CREATE TABLE EMPLOYEE  
(EMPNO      INTEGER GENERATED ALWAYS AS IDENTITY,  
 NAME       CHAR(30),  
 SALARY     DECIMAL(5,2),  
 DEPTNO    SMALLINT);  
  
INSERT INTO EMPLOYEE  
(NAME, SALARY, DEPTNO)  
VALUES ('Rupert', 989.99, 50);  
  
VALUES IDENTITY_VAL_LOCAL() INTO :IVAR;
```

Example 2: Assume two tables, T1 and T2, have an identity column named C1. DB2 generates values 1, 2, 3, . . . for the C1 column in table T1, and values 10, 11, 12, . . . for the C1 column in table T2.

```
CREATE TABLE T1 (C1 SMALLINT GENERATED ALWAYS AS IDENTITY,  
                 C2 SMALLINT );  
  
CREATE TABLE T2 (C1 DECIMAL(15,0) GENERATED BY DEFAULT AS IDENTITY  
                  (START WITH 10),  
                 C2 SMALLINT );  
  
INSERT INTO T1 (C2) VALUES (5);  
  
INSERT INTO T1 (C2) VALUES (5);  
  
SELECT * FROM T1;  
  
C1          C2  
-----  -----  
1           5  
2           5  
  
VALUES IDENTITY_VAL_LOCAL() INTO :IVAR;
```

At this point, the IDENTITY_VAL_LOCAL function would return a value of 2 in IVAR. The following INSERT statement inserts a single row into T2 where column C2 gets a value of 2 from the IDENTITY_VAL_LOCAL function

```
INSERT INTO T2 (C2) VALUES (IDENTITY_VAL_LOCAL());  
  
SELECT * FROM T2  
WHERE C1 = DECIMAL(IDENTITY_VAL_LOCAL(),15,0);
```

C1	C2
10	2

Invoking the IDENTITY_VAL_LOCAL function after this insert would result in a value of 10, which is the value generated by DB2 for column C1 of T2. Assume another single row is inserted into T2. For the following INSERT statement, DB2 assigns a value of 13 to identity column C1 and gives C2 a value of 10 from IDENTITY_VAL_LOCAL. Thus, C2 is given the last identity value that was inserted into T2.

```
|           INSERT INTO T2 (C2, C1) VALUES (IDENTITY_VAL_LOCAL(), 13);
```

Example 3: The IDENTITY_VAL_LOCAL function can also be invoked in an INSERT statement that both invokes the IDENTITY_VAL_LOCAL function and causes a new value for an identity column to be assigned. The next value to be returned is thus established when the IDENTITY_VAL_LOCAL function is invoked after the INSERT statement completes. For example, consider the following table definition:

```
|           CREATE TABLE T1 (C1 SMALLINT GENERATED BY DEFAULT AS IDENTITY,  
|                           C2 SMALLINT);
```

For the following INSERT statement, specify a value of 25 for the C2 column, and DB2 generates a value of 1 for C1, the identity column. This establishes 1 as the value that will be returned on the next invocation of the IDENTITY_VAL_LOCAL function.

```
|           INSERT INTO T1 (C2) VALUES (25);
```

In the following INSERT statement, the IDENTITY_VAL_LOCAL function is invoked to provide a value for the C2 column. A value of 1 (the identity value assigned to the C1 column of the first row) is assigned to the C2 column, and DB2 generates a value of 2 for C1, the identity column. This establishes 2 as the value that will be returned on the next invocation of the IDENTITY_VAL_LOCAL function.

```
|           INSERT INTO T1 (C2) VALUES (IDENTITY_VAL_LOCAL());
```

In the following INSERT statement, the IDENTITY_VAL_LOCAL function is again invoked to provide a value for the C2 column, and the user provides a value of 11 for C1, the identity column. A value of 2 (the identity value assigned to the C1 column of the second row) is assigned to the C2 column. The assignment of 11 to C1 establishes 11 as the value that will be returned on the next invocation of the IDENTITY_VAL_LOCAL function.

```
|           INSERT INTO T1 (C2, C1) VALUES (IDENTITY_VAL_LOCAL(), 11);
```

After the 3 INSERT statements have been processed, table T1 contains the following:

```
|           SELECT * FROM T1;
```

C1	C2
1	25
2	1
11	2

The contents of T1 illustrate that the expressions in the VALUES clause are evaluated before the assignments for the columns of the INSERT statement. Thus, an invocation of an IDENTITY_VAL_LOCAL function invoked from a VALUES clause of an INSERT statement uses the most recently assigned value for an identity column in a previous INSERT statement.

Example 4: Use an application program to find the value automatically allocated for an IDENTITY column just inserted with the SQL INSERT command.

```
|           SQL SET :<host variable> = IDENTITY_VAL_LOCAL()
```

IFNULL

IFNULL

```
►►—IFNULL(expression,expression)—►►
```

The schema is SYSIBM.

The IFNULL function returns the first argument that is not null.

IFNULL is identical to the COALESCE and VALUE scalar functions except that IFNULL is limited to two arguments instead of multiple arguments. For a description, see “COALESCE” on page 192.

Example: For all the rows in sample table DSN8710.EMP, select the employee number and salary. If the salary is missing (is null), have the value 0 returned.

```
SELECT EMPNO, IFNULL(SALARY,0)  
      FROM DSN8710.EMP;
```

INSERT

```
►►—INSERT(expression1,expression2,expression3,expression4)—►►
```

The schema is SYSIBM.

The INSERT function returns a string where, beginning at *expression2* in *expression1*, *expression3* bytes have been deleted and *expression4* has been inserted.

expression1

An expression that specifies the source string. The source string can be any type of character string except a CLOB or any type of graphic string except a DBCLOB. The actual length of the string must be greater than zero.

expression2

An expression that returns an integer. The integer specifies the starting point within the source string where the deletion of bytes and the insertion of another string is to begin. The value of the integer must be in the range of 1 to the length of *expression1* plus one.

expression3

An expression that returns an integer. The integer specifies the number of bytes that are to be deleted from the source string, starting at the position identified by *expression2*. The value of the integer must be in the range of 0 to the length of *expression1*.

expression4

An expression that specifies the string to be inserted into the source string, starting at the position identified by *expression2*. The string to be inserted can be any type of character string except a CLOB or any type of graphic string except a DBCLOB.

expression1 and *expression4* must have compatible string types. For more information on compatibility, see “Conversion rules for string comparison” on page 73. Neither *expression1* nor *expression4* can be a CLOB or DBCLOB.

The result of the function depends on the data type of the first and fourth arguments:

- VARCHAR if *expression1* and *expression4* are character strings
- VARGRAPHIC if *expression1* and *expression4* are graphic strings

The length attribute of the result depends on the arguments:

- If *expression2* and *expression3* are constants, the length attribute of the result is:

$$L1 - \text{MIN}((L1 - V2 + 1), V3) + L4$$

where:

L1 is the length attribute of *expression1*

V2 is the value of *expression2*

V3 is the value of *expression3*

L4 is the length attribute of *expression4*

- Otherwise, the length attribute of the result is the length attribute of *expression1* plus the length attribute of *expression4*. In this case, the length attribute of

INSERT

expression1 plus the length attribute of *expression4* must not exceed 4000 for a VARCHAR result or 2000 for a VARGRAPHIC result.

The actual length of the result is:

$$A1 - \text{MIN}((A1 - V2 + 1), V3) + A4$$

where:

A1 is the actual length of *expression1*

V2 is the value of *expression2*

V3 is the value of *expression3*

A4 is the actual length of *expression4*

If the actual length of the result string exceeds the maximum for the return data type, an error occurs.

If any argument can be null, the result can be null; if any argument is null, the result is the null value.

The subtype and CCSID of the result are determined as follows:

- If either *expression1* and *expression4* is character bit data, the result is bit data and does not have an associated CCSID.
- If *expression1* and *expression4* are both character SBCS data, the result is SBCS data and the CCSID is the CCSID for ASCII, EBCDIC, or Unicode SBCS data, depending on the encoding scheme of the SQL statement.
- If *expression1* and *expression4* are both graphic data, the result is graphic data and the CCSID is the CCSID for ASCII, EBCDIC, Unicode graphic data, depending on the encoding scheme of the SQL statement.
- Otherwise, the result is mixed data. The CCSID is the CCSID for ASCII, EBCDIC or Unicode mixed data, depending on the encoding scheme of the SQL statement.

Example 1: The following example shows how the string 'INSERTING' can be changed into other strings. The use of the CHAR function limits the length of the resulting string to 10 bytes.

```
SELECT CHAR(INSERT('INSERTING',4,2,'IS'),10),
       CHAR(INSERT('INSERTING',4,0,'IS'),10),
       CHAR(INSERT('INSERTING',4,2,''),10)
  FROM SYSIBM.SYSDUMMY1;
```

This example returns 'INSISTING ', 'INSERTIN', and 'INSTING '

Example 2: The previous example demonstrated how to insert text into the middle of some text. This example shows how to insert text before some text by using 1 as the starting point (*expression2*).

```
SELECT CHAR(INSERT('INSERTING',1,0,'XX'),10),
       CHAR(INSERT('INSERTING',1,1,'XX'),10),
       CHAR(INSERT('INSERTING',1,2,'XX'),10),
       CHAR(INSERT('INSERTING',1,3,'XX'),10)
  FROM SYSIBM.SYSDUMMY1;
```

This example returns 'XXINSERTIN', 'XXNINSERTING', 'XXSERTING ', and 'XXERTING '

Example 3: The following example shows how to insert text after some text. Add 'XX' at the end of string 'ABCABC'. Because the source string is 6 characters long, set the starting position to 7 (one plus the length of the source string).

INSERT

```
SELECT CHAR(INSERT('ABCABC',7,0,'XX'),10)
   FROM SYSIBM.SYSDUMMY1;
```

This example returns 'ABCABCXX XX'.

INTEGER or INT

INTEGER or INT

```
►-- INTEGER --> ( -- numeric-expression -- )
|   INT      |   string-expression
```

The schema is SYSIBM.

The INTEGER or INT function returns an integer representation of a number or character string in the form of a numeric constant.

numeric-expression

An expression that returns a value of any built-in numeric data type.

The result of the function is a large integer. The result is the same number that would occur if the argument were assigned to a large integer column or variable. If the whole part of the argument is not within the range of integers, an error occurs. If present, the decimal part of the argument is truncated.

string-expression

An expression that returns any type of string (except a BLOB, CLOB, or DBCLOB) with an actual length that is not greater than 255 bytes. Leading and trailing blanks are removed from the string, and the resulting substring must conform to the rules for forming a string representation of an SQL integer constant.

The result of the function is a large integer. The result is the same number that would occur if the corresponding numeric constant were assigned to a large integer column or variable.

If the argument can be null, the result can be null; if the argument is null, the result is the null value.

Example 1: Using sample table DSN8710.EMP, find the average salary of the employees in department A00, rounding the result to the nearest dollar.

```
SELECT INTEGER(AVG(SALARY)+.5)
   FROM DSN8710.EMP
 WHERE WORKDEPT = 'A00';
```

Example 2: Using sample table DSN8710.EMP, select the EMPNO column, which is defined as CHAR(6), in integer form.

```
SELECT INTEGER(EMPNO)
   FROM DSN8710.EMP;
```

JULIAN_DAY

```
►►—JULIAN_DAY(expression)—►►
```

The schema is SYSIBM.

The JULIAN_DAY function returns an integer value representing a number of days from January 1, 4713 B.C. (the start of the Julian date calendar) to the date specified in the argument.

The argument must be a date, a timestamp, or a valid string representation of a date or timestamp. A string representation must not be a BLOB, CLOB, or DBCLOB and must have an actual length that is not greater than 255 bytes. (For the valid formats of string representations of dates and timestamps, see “String representations of datetime values” on page 57.)

The result of the function is a large integer. If the argument can be null, the result can be null; if the argument is null, the result is the null value.

Example 1: Using sample table DSN8710.EMP, set the integer host variable JDAY to the Julian day of the day that Christine Haas (EMPNO = '000010') was employed (HIREDATE = '1965-01-01').

```
SELECT JULIAN_DAY(HIREDATE)
  INTO :JDAY
  FROM DSN8710.EMP
 WHERE EMPNO = '000010';
```

The result is that JDAY is set to 2438762.

Example 2: Set integer host variable JDAY to the Julian day for January 1, 1998.

```
SELECT JULIAN_DAY('1998-01-01')
  INTO :JDAY
  FROM SYSIBM.SYSDUMMY1;
```

The result is that JDAY is set to 2450815.

LAST_DAY

LAST_DAY

```
►►—LAST_DAY(date-expression)—►►
```

The schema is SYSIBM.

The LAST_DAY scalar function returns a date that represents the last day of the month indicated by *date-expression*.

date-expression must be a date, a timestamp, or a valid string representation of a date or timestamp. A string representation must not be a BLOB, CLOB, or DBCLOB, and must have an actual length that is not greater than 255 bytes. For the valid formats of string representations of dates and timestamps, see “String representations of datetime values” on page 57.

The result of the function is a DATE. The result can be null; if the value of *date-expression* is null, the result is the null value.

Example 1: Set the host variable END_OF_MONTH with the last day of the current month.

```
SET :END_OF_MONTH = LAST_DAY(CURRENT_DATE);
```

The host variable END_OF_MONTH is set with the value representing the end of the current month. If the current day is 2000-02-10, then END_OF_MONTH is set to 2000-02-29.

Example 2: Set the host variable END_OF_MONTH with the last day of the month in EUR format for the given date.

```
SET :END_OF_MONTH = CHAR(LAST_DAY('1965-07-07'), EUR);
```

The host variable END_OF_MONTH is set with the value '31.07.1965'.

LCASE or LOWER

```

    |--> LCASE (string-expression)
    |     |
    |     |--> LOWER
  
```

The schema is SYSIBM.

The LCASE or LOWER function returns a string in which all the characters have been converted to lowercase characters.

string-expression

An expression that specifies the string to be converted. The string must be a character or graphic string. A character string argument must not be a CLOB and must have an actual length that is not greater than 255. A graphic string argument must not be a DBCLOB and must have an actual length that is not greater than 127.

The alphabetic characters of the argument are translated to lowercase characters based on the value of the LC_CTYPE locale in effect for the statement. For example, characters A-Z are translated to a-z, and characters with diacritical marks are translated to their lowercase equivalent, if any. The locale is determined by special register CURRENT LOCALE LC_CTYPE. For information about the special register, see “CURRENT LOCALE LC_CTYPE” on page 87.

If the LC_CTYPE locale is blank when the function is executed, the result of the function depends on the data type of *string-expression*.

- For ASCII and EBCDIC, if *string-expression* specifies a graphic string expression, then an error occurs. For a character string expression, characters A-Z are translated to a-z and characters with diacritical marks are not translated. If the string contains mixed or DBCS characters, fullwidth Latin capital letters A-Z are converted to fullwidth Latin small letters a-z.
- For Unicode, *string-expression* can be either a character string expression or a graphic string expression. The characters A-Z are translated to a-z and all other characters, including characters with diacritic marks, are left unchanged. If LOCALE LC_CTYPE is not blank, an error occurs. Fullwidth Latin capital letters A-Z are converted to fullwidth Latin small letters a-z.

The length attribute, data type, subtype, and CCSID of the result are the same as the argument. If the argument can be null, the result can be null; if the argument is null, the result is the null value.

Example: Return the characters in the value of host variable NAME in lowercase. NAME has a data type of VARCHAR(30) and a value of 'Christine Smith'. Assume that the locale in effect is blank.

```

SELECT LCASE(:NAME)
  FROM SYSIBM.SYSDUMMY1;
  
```

The result is the value 'christine smith'.

LEFT

LEFT

```
►►—LEFT(string-expression, integer)—►►
```

The schema is SYSIBM.

The LEFT function returns a string consisting of the specified number of leftmost *integer* characters of *string-expression*. If *string-expression* is a character or binary string, a character is a byte. If *string-expression* is a graphic string, a character is a DBCS character.

The CCSID of the result is the same as that of the *string-expression*.

string-expression

An expression that specifies the string from which the result is derived. The string must be a character, graphic, or binary string. A substring of *string-expression* is zero or more contiguous bytes of *string-expression*.

The string can contain mixed data. However, because the function operates on a strict byte-count basis, the result is not necessarily a properly formed mixed data character string.

integer

An expression that specifies the length of the result. The value must be an integer between 0 and *n*, where *n* is the length attribute of *string-expression*.

The *string-expression* is effectively padded on the right with the necessary number of characters so that the specified substring of *string-expression* always exists. The encoding scheme of the data determines the padding character:

- For ASCII SBCS data or ASCII mixed data, the padding character is X'20'.
- For ASCII DBCS data, the padding character depends on the CCSID; for example, for Japan (CCSID 301) the padding character is X'8140', while for simplified Chinese it is X'A1A1'.
- For EBCDIC SBCS data or EBCDIC mixed data, the padding character is X'40'.
- For EBCDIC DBCS data, the padding character is X'4040'.
- For Unicode SBCS data or UTF-8 (Unicode mixed data), the padding character is X'20'.
- For UTF-16 (Unicode DBCS) data, the padding character is X'0020'.
- For binary data, the padding character is X'00'.

The result of the function is a varying-length string with a length attribute that is the same as the length attribute of *string-expression* and a data type that depends on the data type of *string*:

- VARCHAR if *string-expression* is CHAR or VARCHAR
- CLOB if *string-expression* is CLOB
- VARGRAPHIC if *string-expression* is GRAPHIC or VARGRAPHIC
- DBCLOB if *string-expression* is DBCLOB
- BLOB if *string-expression* is BLOB

If any argument of the function can be null, the result can be null; if any argument is null, the result is the null value.

Example 1: Assume that host variable ALPHA has a value of 'ABCDEF'. The following statement:

```
SELECT LEFT(:ALPHA,3)
  FROM SYSIBM.SYSDUMMY1;
```

returns 'ABC', which are the three leftmost characters in ALPHA.

Example 2: Assume that host variable NAME, which is defined as VARCHAR(50), has a value of 'KATIE AUSTIN' and the integer host variable FIRSTNAME_LEN has a value of 5. The following statement:

```
SELECT LEFT(:NAME, :FIRSTNAME_LEN)
  FROM SYSIBM.SYSDUMMY1;
```

returns the value 'KATIE'.

Example 3: The following statement returns a zero length string.

```
SELECT LEFT('ABCABC',0)
  FROM SYSIBM.SYSDUMMY1;
```

Example 4: The FIRSTNME column in sample EMP table is defined as VARCHAR(12). Find the first name for an employee whose last name is 'BROWN' and return the first name in a 10-byte string.

```
SELECT LEFT(FIRSTNME,10)
  FROM DSN8710.EMP
 WHERE LASTNAME='BROWN';
```

This function returns a VARCHAR(10) string that has the value of 'DAVID' followed by 5 blank characters.

LENGTH

LENGTH

```
►►—LENGTH(expression)—►►
```

The schema is SYSIBM.

The LENGTH function returns the length of its argument.

The argument is an expression that returns a value of any built-in data type.

The result of the function is a large integer. If the argument can be null, the result can be null; if the argument is null, the result is the null value.

The result is the length of the argument. The length does not include the null indicator byte of column arguments that allow null values. The length of strings includes blanks but does not include the length control field of varying-length strings. The length of a varying-length string is the actual length, not the maximum length.

| The length of a graphic string is the number of double-byte characters. Unicode
| UTF-16 data is treated as graphic data; a UTF-surrogate character takes two DBCS
| characters to represent and as such is counted as two DBCS characters.

The length of all other values is the number of bytes used to represent the value:

- 2 for small integer
- 4 for large integer
- 4 for single precision floating-point
- 8 for double precision floating-point
- $\text{INTEGER}(p/2)+1$ for decimal numbers with precision p
- 4 for date
- 3 for time
- 10 for timestamp
- The length of the string for character strings
- The length of the row ID

Example 1: Assume that FIRSTNME is a VARCHAR(12) column that contains 'ETHEL' for employee 280. The following query:

```
SELECT LENGTH(FIRSTNME)
      FROM DSN8710.EMP
     WHERE EMPNO = '000280';
```

returns the value 5.

Example 2: Assume that HIREDATE is a column of data type DATE. Then, regardless of value:

```
LENGTH(HIREDATE)
```

returns the value 4, and

```
LENGTH(CHAR(HIREDATE, EUR))
```

returns the value 10.

LN

```
►►—LN(expression)—►►
```

The schema is SYSIBM.

The LN function returns the natural logarithm of the argument. The LN and EXP functions are inverse operations. LOG is a synonym for LN.

The argument is an expression that returns the value of any built-in numeric data type. If the argument is not a double precision floating-point number, it is converted to one for processing by the function.

The result of the function is a double precision floating-point number. The result can be null; if the argument is null, the result is the null value.

Example: Assume that host variable NATLOG is DECIMAL(4,2) with a value of 31.62. The following statement:

```
SELECT LN(:NATLOG)
      FROM SYSIBM.SYSDUMMY1;
```

returns a double precision floating-point number with an approximate value of 3.45.

LOCATE

LOCATE

```
►►—LOCATE(search-string,source-string [ ,start ])—►►
```

The schema is SYSIBM.

The LOCATE function returns the starting position of the first occurrence of one string (the *search-string*) within another string (the *source-string*). Numbers for the starting position begin at 1 and not 0. If *search-string* is not found in *source-string*, the function returns 0.

search-string

An expression that specifies the string that is to be searched for. The search string can be a character, graphic, or binary string with an actual length that is no greater than 4000 bytes. The expression can be specified by any of the following:

- A constant
- A special register
- A host variable (including a LOB locator variable)
- A scalar function whose arguments are any of the above
- A CAST specification whose arguments are any of the above
- An expression that concatenates (using CONCAT or II) any of the above
- A column name

These rules are similar to those that are described for *pattern-expression* for the LIKE predicate.

source-string

An expression that specifies the source string that is to be searched. The source string can be a character, graphic, or binary string. The expression can be specified by any of the following:

- A constant
- A special register
- A host variable (including a LOB locator variable)
- A scalar function whose arguments are any of the above
- A CAST specification whose arguments are any of the above
- A column name
- An expression that concatenates (using CONCAT or II) any of the above

start

An expression whose value is a positive integer. The integer specifies the position in the source string at which the search begins. If *start* is specified, the LOCATE function is equivalent to:

```
POSSTR(SUBSTR(source-string, integer), search-string) + integer - 1
```

If *start* is not specified, the search begins at the first character of the source string and the LOCATE function is equivalent to:

```
POSSTR(source-string, search-string)
```

The first and second arguments must have compatible string types. For more information on compatibility, see “Conversion rules for string comparison” on page 73.

The result of the function is a large integer. If any argument can be null, the result can be null; if any argument is null, the result is the null value.

For more information about LOCATE, see the description of “POSSTR” on page 278.

Example 1: Find the location of the first occurrence of the character 'N' in the string 'DINING'.

```
SELECT LOCATE('N', 'DINING')
      FROM SYSIBM.SYSDUMMY1;
```

The result is the value 3.

Example 2: For all the rows in the table named IN_TRAY, select the RECEIVED column, the SUBJECT column, and the starting position of the string 'GOOD' within the NOTE_TEXT column.

```
SELECT RECEIVED, SUBJECT, LOCATE('GOOD', NOTE_TEXT)
      FROM IN_TRAY
     WHERE LOCATE('GOOD', NOTE_TEXT) <> 0;
```

LOG10

LOG10

```
►►—LOG10(expression)—►►
```

The schema is SYSIBM.

The LOG10 function returns the base 10 logarithm of the argument.

The argument is an expression that returns the value of any built-in numeric data type. If the argument is not a double precision floating-point number, it is converted to one for processing by the function.

The result of the function is a double precision floating-point number. The result can be null; if the argument is null, the result is the null value.

Example: Assume that host variable HLOG is an INTEGER with a value of 100. The following statement:

```
SELECT LOG10(:HLOG)
      FROM SYSIBM.SYSDUMMY1;
```

returns a double precision floating-point number with an approximate value of 2.

LTRIM

►►—LTRIM(*string-expression*)—►►

The schema is SYSIBM.

The LTRIM function removes blanks from the beginning of a string expression. The LTRIM function returns the same results as the STRIP function with LEADING specified:

STRIP(*string-expression*,LEADING)

string-expression must be any character string expression other than a CLOB or any graphic string expression other than a DBCLOB. The characters that are interpreted as leading blanks depend on the encoding scheme of the data and the data type:

- If the argument is a graphic string, the leading DBCS blanks are removed. For data that is encoded in ASCII, the ASCII CCSID determines the hex value that represents a double-byte blank. For example, for Japan (CCSID 301), X'8140' represents a double-byte blank, while it is X'A1A1' for Simplified Chinese. For EBCDIC-encoded data, X'4040' represents a double-byte blank. For Unicode-encoded data, X'0020' represents a UTF-16 blank.
- Otherwise, leading SBCS blanks are removed. For data that is encoded in ASCII, X'20' represents a blank. For EBCDIC-encoded data, X'40' represents a blank. For Unicode-encoded data, X'20' represents a single-byte blank.

The result of the function depends on the data type of its argument:

- VARCHAR if the argument is a character string
- VARGRAPHIC if the argument is a graphic string

The length attribute of the result is the same as the length attribute of *string-expression*. The actual length of the result is the length of the expression minus the number of characters removed. If all of the characters are removed, the result is an empty string.

If the argument can be null, the result can be null; if the argument is null, the result is the null value. The CCSID of the result is the same as that of *string-expression*.

Example: Assume that host variable HELLO is defined as CHAR(9) and has a value of 'Hello'.

```
SELECT LTRIM(:HELLO)
  FROM SYSIBM.SYSDUMMY1;
```

The result is 'Hello'.

MAX

MAX

The diagram shows the syntax for the MAX function. It consists of the word "MAX" followed by a left parenthesis, then two arguments separated by a comma, and finally a right parenthesis. A small bracket is placed over the first argument "expression1". A vertical arrow points from the word "Notes:" to this bracketed area.

Notes:

- 1 GREATEST can be specified as an alternative to MAX.

The schema is SYSIBM.

The MAX scalar function returns the maximum value in a set of values. The arguments must be compatible. For more information on compatibility, refer to the compatibility matrix in Table 9 on page 65. All but the first argument can be parameter markers. There must be two or more arguments.

The argument values can be of any built-in data type other than a CLOB, DBCLOB, BLOB, or row ID. Character string arguments cannot have an actual length greater than 255, and graphic string arguments cannot have an actual length greater than 127.

The selected argument is converted, if necessary, to the attributes of the result. The attributes of the result are determined using the “Rules for result data types” on page 77. If the MAX function has more than two arguments, the rules are applied to the first two arguments to determine a candidate result type. The rules are then applied to that candidate result type and the third argument to determine another candidate result type. This process continues until all arguments are analyzed and the final result type is determined.

The result can be null if at least one argument can be null; the result is the null value if one of the arguments is null.

Example 1: Assume the host variable M1 is a DECIMAL(2,1) host variable with a value of 5.5, host variable M2 is a DECIMAL(3,1) host variable with a value of 4.5, and host variable M3 is a DECIMAL(3,2) host variable with a value of 6.25. The function

```
MAX(:M1,:M2,:M3)
```

returns the value 6.25.

Example 2: Assume the host variable M1 is a CHAR(2) host variable with a value of 'AA', host variable M2 is a CHAR(3) host variable with a value of 'AA ', and host variable M3 is a CHAR(4) host variable with a value of 'AA A'. The function

```
MAX(:M1,:M2,:M3)
```

returns the value 'AA A'.

MICROSECOND

```
►►MICROSECOND(expression)►►
```

The schema is SYSIBM.

The MICROSECOND function returns the microsecond part of its argument.

The argument must be a timestamp, a timestamp duration, or a valid string representation of a timestamp. A string representation must not be a BLOB, CLOB, or DBCLOB and must have an actual length that is not greater than 255 bytes. (For the valid formats of string representations of times and timestamps, see “String representations of datetime values” on page 57.)

The result of the function is a large integer. If the argument can be null, the result can be null; if the argument is null, the result is the null value.

The other rules depend on the data type of the argument:

If the argument is a timestamp or string representation of a timestamp, the result is the microsecond part of the value, which is an integer between 0 and 999999.

If the argument is a duration, the result is the microsecond part of the value, which is an integer between -999999 and 999999. A nonzero result has the same sign as the argument.

Example: Assume that table TABLEX contains a TIMESTAMP column named TSTMPCOL and a SMALLINT column named INTCOL. Select the microseconds part of the TSTMPCOL column of the rows where the INTCOL value is 1234:

```
SELECT MICROSECOND(TSTMPCOL) FROM TABLEX  
WHERE INTCOL = 1234;
```

MIDNIGHT_SECONDS

MIDNIGHT_SECONDS

```
►►MIDNIGHT_SECONDS(expression)--►►
```

The schema is SYSIBM.

The MIDNIGHT_SECONDS function returns an integer value in the range of 0 to 86400 that represents the number of seconds between midnight and the time specified by the argument.

The argument must be a time, a timestamp, or a valid string representation of a time or timestamp. A string representation must not be a BLOB, CLOB, or DBCLOB, and must have an actual length that is not greater than 255 bytes. (For the valid formats of string representations of dates and timestamps, see “String representations of datetime values” on page 57.)

The result of the function is a large integer. If the argument can be null, the result can be null; if the argument is null, the result is the null value.

Example 1: Find the number of seconds between midnight and 00:01:00, and midnight and 13:10:10. Assume that host variable XTIME1 has a value of '00:01:00', and that XTIME2 has a value of '13:10:10'.

```
SELECT MIDNIGHT_SECONDS(:XTIME1), MIDNIGHT_SECONDS(:XTIME2)
   FROM SYSIBM.SYSDUMMY1;
```

This example returns 60 and 47410. Because there are 60 seconds in a minute and 3600 seconds in an hour, 00:01:00 is 60 seconds after midnight ((60 * 1) + 0), and 13:10:10 is 47410 seconds ((3600 * 13) + (60 * 10) + 10).

Example 2: Find the number of seconds between midnight and 24:00:00, and midnight and 00:00:00.

```
SELECT MIDNIGHT_SECONDS('24:00:00'), MIDNIGHT_SECONDS('00:00:00')
   FROM SYSIBM.SYSDUMMY1;
```

This example returns 86400 and 0. Although these two values represent the same point in time, different values are returned.

MIN

►►MIN(expression₁,expression₂)₍₁₎

Notes:

- 1 LEAST can be specified as an alternative to MIN.

The schema is SYSIBM.

The MIN scalar function returns the minimum value in a set of values. The arguments must be compatible. For more information on compatibility, refer to the compatibility matrix in Table 9 on page 65. All but the first argument can be parameter markers. There must be two or more arguments.

The argument values can be of any built-in data type other than a CLOB, DBCLOB, BLOB, or row ID. Character string arguments cannot have an actual length greater than 255, and graphic string arguments cannot have an actual length greater than 127.

The selected argument is converted, if necessary, to the attributes of the result. The attributes of the result are determined using the “Rules for result data types” on page 77. If the MIN function has more than two arguments, the rules are applied to the first two arguments to determine a candidate result type. The rules are then applied to that candidate result type and the third argument to determine another candidate result type. This process continues until all arguments are analyzed and the final result type is determined.

The result can be null if at least one argument can be null; the result is the null value if one of the arguments is null.

Example 1: Assume the host variable M1 is a DECIMAL(2,1) host variable with a value of 5.5, host variable M2 is a DECIMAL(3,1) host variable with a value of 4.5, and host variable M3 is a DECIMAL(3,2) host variable with a value of 6.25. The function

`MIN(:M1,:M2,:M3)`

returns the value 4.5.

Example 2: Assume the host variable M1 is a CHAR(2) host variable with a value of 'AA', host variable M2 is a CHAR(3) host variable with a value of 'AAA', and host variable M3 is a CHAR(4) host variable with a value of 'AAAA'. The function

`MIN(:M1,:M2,:M3)`

returns the value 'AA' .

MINUTE

MINUTE

```
►►MINUTE(expression)—►►
```

The schema is SYSIBM.

The MINUTE function returns the minute part of its argument.

The argument must be a time, a timestamp, time duration, timestamp duration, or a valid string representation of a time or timestamp. A string representation must not be a BLOB, CLOB, or DBCLOB and must have an actual length that is not greater than 255 bytes. (For the valid formats of string representations of times and timestamps, see “String representations of datetime values” on page 57.)

The result of the function is a large integer. If the argument can be null, the result can be null; if the argument is null, the result is the null value.

The other rules depend on the data type of the argument:

If the argument is a time, timestamp, or string representation of either, the result is the minute part of the value, which is an integer between 0 and 59.

If the argument is a time duration or timestamp duration, the result is the minute part of the value, which is an integer between -99 and 99. A nonzero result has the same sign as the argument.

Example: Assume that a table named CLASSES contains one row for each scheduled class. Assume also that the class starting times are in the TIME column named STARTTM. Using these assumptions, select those rows in CLASSES that represent classes that start on the hour.

```
SELECT * FROM CLASSES  
WHERE MINUTE(STARTTM) = 0;
```

MOD

►►MOD(*expression, expression*)►►

The schema is SYSIBM.

The MOD function divides the first argument by the second argument and returns the remainder.

The formula used to calculate the remainder is:

$$\text{MOD}(x,y) = x - (x/y) * y$$

where x/y is the truncated integer result of the division.

The arguments must each be an expression that returns a value of any built-in numeric data type. The second argument cannot be zero.

If an argument can be null, the result can be null; if an argument is null, the result is the null value.

The attributes of the result are based on the arguments as follows:

- If both arguments are integers, the data type of the result is a large integer.
- If one argument is an integer and the other is a decimal, the data type of the result is decimal with the same precision and scale as the decimal argument.
- If both arguments are decimal, the data type of the result is decimal. The precision of the result is $\min(p-s,p'-s') + \max(s,s')$, and the scale of the result is $\max(s,s')$, where the symbols p and s denote the precision and scale of the first operand, and the symbols p' and s' denote the precision and scale of the second operand.
- If either argument is a floating-point number, the data type of the result is double precision floating-point.

The operation is performed in floating-point. If necessary, the operands are first converted to double precision floating-point numbers. For example, an operation that involves a floating-point number and either an integer or a decimal number is performed with a temporary copy of the integer or decimal number that has been converted to double precision floating-point. The result of a floating-point operation must be within the range of floating-point numbers.

Example: Assume that M1 and M2 are two host variables. Find the remainder of dividing M1 by M2.

```
SELECT MOD(:M1,:M2)
      FROM SYSIBM.SYSDUMMY1;
```

The following table shows the result for this function for various values of M1 and M2.

M1 data type	M1 value	M2 data type	M2 value	Result of MOD(:M1,:M2)
INTEGER	5	INTEGER	2	1
INTEGER	5	DECIMAL(3,1)	2.2	0.6

MOD

M1 data type	M1 value	M2 data type	M2 value	Result of MOD(:M1,:M2)
INTEGER	5	DECIMAL(3,2)	2.20	0.60
DECIMAL(4,2)	5.50	DECIMAL(4,1)	2.0	1.50

MONTH

```
►►MONTH(expression)►►
```

The schema is SYSIBM.

The MONTH function returns the month part of its argument.

The argument must be a date, a timestamp, date duration, timestamp duration, or a valid string representation of a date or timestamp. A string representation must not be a BLOB, CLOB, or DBCLOB and must have an actual length that is not greater than 255 bytes. (For the valid formats of string representations of dates and timestamps, see “String representations of datetime values” on page 57.)

The result of the function is a large integer. If the argument can be null, the result can be null; if the argument is null, the result is the null value.

The other rules depend on the data type of the argument:

If the argument is a date, timestamp, or string representation of either, the result is the month part of the value, which is an integer between 1 and 12.

If the argument is a date duration or timestamp duration, the result is the month part of the value, which is an integer between -99 and 99. A nonzero result has the same sign as the argument.

Example: Select all rows in the sample table DSN8710.EMP for employees who were born in May:

```
SELECT * FROM DSN8710.EMP  
WHERE MONTH(BIRTHDATE) = 5;
```

MQPUBLISH

MQPUBLISH

```
#  
#  
#>>>MQPUBLISH( [publisher-service], [service-policy], [msg-data], [topic-list], [correl-id] )  
#  
#
```

Notes:

- 1 *correl-id* cannot be specified unless a publisher service and a service policy are also specified.

The schema is DB2MQ1C or DB2MQ2C

The MQPUBLISH function publishes the data that is contained in *msg-data* to the MQSeries publisher that is specified in *publisher-service*, using the quality-of-service policy that is defined by *service-policy*. A list of topics for the message must be specified, and an optional user-defined message correlation identifier can also be specified. The MQPUBLISH function requires the installation and configuration of an MQSeries-based publish and subscribe system, such as Websphere MQSeries Integrator. See www.ibm.com/software/MQSeries for more details. The function returns a value of '1' if successful or '0' if unsuccessful.

publisher-service

An expression that returns a value that is a built-in character string data type that is not a CLOB. The value of the expression must not be the null value, an empty string, or a string with trailing blanks. The expression must have an actual length that is no greater than 48 bytes. The value of the expression refers to service point that is the logical MQSeries destination to which the message is sent. A service point is defined in the DSNAMT repository file, and it specifies a logical end point from which a message is sent or received. A service point definition includes the name of the MQSeries queue manager and the name of the queue. See *MQSeries Application Messaging Interface* for more details.

If *publisher-service* is not specified, the DB2.DEFAULT.PUBLISHER is used.

service-policy

An expression that returns a value that is a built-in character-string data type that is not a CLOB. The value of the expression must not be the null value, an empty string, or a string with trailing blanks. The expression must have an actual length that is no greater than 48 bytes. The value of the expression refers to the MQSeries AMI service policy that is used in handling this message. A service policy is defined in the DSNAMT repository file, and it specifies a set of quality-of-service options that are to be applied to this messaging operation. These options include message priority and message persistence. See *MQSeries Application Messaging Interface* for more details.

If *service-policy* is not specified, the DB2.DEFAULT.POLICY is used.

msg-data

An expression that returns a value that is a built-in character-string data type. If the expression is a CLOB, the value must not be longer than 1MB. Otherwise, the value must not be longer than 4000 bytes. The value of the expression specifies the data to be sent via MQSeries. A null value, an empty string, and a fixed length string with trailing blanks are all considered valid values.

topic-list

An expression that returns a value that is a built-in character-string data type that is not a CLOB. The value of the expression must not be the null value, an

```
# empty string, or a string with trailing blanks. The expression must have an
# actual length that is no greater than 40 bytes. The value of the expression
# specifies the topics for the message publication. One or more topics can be
# specified, where the topics are separated with a colon. For example, 't1:t2:the
# third topic' indicates that the message is associated with all three topics: t1, t2,
# and 'the third topic'.
```

correl-id

An expression that returns a value that is a built-in character-string data type that is not a CLOB. The value of the expression must not be the null value or an empty string, and it must have an actual length that is no greater than 24 bytes. The value of the expression specifies the correlation identifier to be associated with this message. The *correl-id* is often specified in request-and-reply scenarios to associate requests with replies.

```
# A null value, an empty string, and a fixed length string with trailing blanks are all
# considered valid values. However, when the correl-id is specified on another
# request such as MQPUBLISH, the correl-id must be specified the same to be
# recognized as a match. For example, specifying a value of 'test' for correl-id on
# MQRECEIVE will not match a correl-id value of 'test   ' (with trailing blanks)
# specified earlier on an MQPUBLISH request.
```

```
# If correl-id is not specified, a correlation identifier is not used, and a correlation
# identifier is not added to the message.
```

```
# Example 1: Publish the string 'Testing 123' to the default publisher service
# (DB2.DEFAULT.PUBLISHER), using the default service policy
# (DB2.DEFAULT.POLICY). Do not specify a topic or correlation identifier for the
# message.
```

```
# SELECT DB2MQ2C.MQPUBLISH('Testing 123')
#   FROM SYSIBM.SYSDUMMY1;
```

```
# Example 2: Publish the string 'Testing 345' to the publisher service
# 'MYPUBLISHER' under the topic 'TESTS', using the default service policy
# (DB2.DEFAULT.POLICY). Do not specify a correlation identifier for the message.
```

```
# SELECT DB2MQ2C.MQPUBLISH('MYPUBLISHER','Testing 345', 'TESTS')
#   FROM SYSIBM.SYSDUMMY1;
```

```
# Example 3: Publishes the string 'Testing 678' to the publisher service
# 'MYPUBLISHER' under the topic 'TESTS', using the service policy 'MYPOLICY'.
# Specify a correlation identifier of 'TEST1' for the message.
```

```
# SELECT DB2MQ2C.MQPUBLISH('MYPUBLISHER','MYPOLICY','Testing 678', 'TESTS', 'TEST1')
#   FROM SYSIBM.SYSDUMMY1;
```

```
# All of the examples return a value of '1' if they are successful.
```

MQPUBLISHXML

MQPUBLISHXML

```
#  
#  
#>>>MQPUBLISHXML(  
#    publisher-service,  
#    service-policy,  
#    msg-data,  
#    topic-list  
#    [,  
#    correl-id(1)  
#])  
#  
#  
# Notes:  
# 1   correl-id cannot be specified unless a publisher service and a service policy are also specified.  
#
```

The schema is DMQXML1C or DMQXML2C.

The MQPUBLISHXML function publishes the XML data that is contained in
msg-data to the MQSeries publisher that is specified in *publisher-service*, using the
quality-of-service policy that is defined by *service-policy*. A list of topics for the
message must be specified, and an optional user-defined message correlation
identifier can also be specified. The MQPUBLISHXML function requires the
installation and configuration of an MQSeries-based publish and subscribe system,
such as Websphere MQSeries Integrator. See www.ibm.com/software/MQSeries for
more details. The function returns a value of '1' if it is successful or '0' if it is
unsuccessful.

publisher-service

An expression that returns a value that is a built-in character string data type
that is not a CLOB. The value of the expression must not be the null value, an
empty string, or a string with trailing blanks. The expression must have an
actual length that is no greater than 48 bytes. The value of the expression
refers to service point that is the logical MQSeries destination to which the
message is sent. A service point is defined in the DSNAME repository file, and it
specifies a logical end point from which a message is sent or received. A
service point definition includes the name of the MQSeries queue manager and
the name of the queue. See *MQSeries Application Messaging Interface* for
more details.

If *publisher-service* is not specified, DB2.DEFAULT.PUBLISHER is used.

service-policy

An expression that returns a value that is a built-in character-string data type
that is not a CLOB. The value of the expression must not be the null value, an
empty string, or a string with trailing blanks. The expression must have an
actual length that is no greater than 48 bytes. The value of the expression
refers to the MQSeries AMI service policy that is used in handling this message.
A service policy is defined in the DSNAME repository file, and it specifies a set
of quality-of-service options that are to be applied to this messaging operation.
These options include message priority and message persistence. See
MQSeries Application Messaging Interface for more details.

If *service-policy* is not specified, DB2.DEFAULT.POLICY is used.

msg-data

An expression that returns a value with a user-defined data type of
DB2XML.XMLVARCHAR (for messages up to 3K in size) or
DB2XML.XMLCLOB (for messages up to 1M in size). The value of the
expression specifies the message data to be sent via MQSeries.

topic-list

An expression that returns a value that is a built-in character-string data type

```
# that is not a CLOB. The value of the expression must not be the null value, an
# empty string, or a string with trailing blanks. The expression must have an
# actual length that is no greater than 40 bytes. The value of the expression
# specifies the topics for the message publication. One or more topics can be
# specified, where the topics are separated with a colon. For example, 't1:t2:the
# third topic' indicates that the message is associated with all three topics: t1, t2,
# and 'the third topic'.
```

correl-id

An expression that returns a value that is a built-in character-string data type that is not a CLOB. The expression must have an actual length that is no greater than 24 bytes. The value of the expression specifies the correlation identifier to be associated with this message. The *correl-id* is often specified in request-and-reply scenarios to associate requests with replies.

```
# A null value, an empty string, and a fixed-length string with trailing blanks are all
# considered valid values. However, when the correl-id is specified on another
# request, such as MQPUBLISHXML, the correl-id must be specified the same to
# be recognized as a match. For example, specifying a value of 'test' for correl-id
# on MQRECEIVEXML will not match a correl-id value of 'test' (with trailing
# blanks) specified earlier on an MQPUBLISHXML request.
```

```
# If correl-id is not specified, a correlation identifier is not used, and a correlation
# identifier is not added to the message.
```

```
# The result of the function is a varying-length string with a length attribute of 1. The
# result is nullable, even though a null value is never returned.
```

```
# The CCSID of the result is the system CCSID that was in effect at the time that the
# MQSeries function was installed.
```

Example 1: The ORDER column in table ORDER_TABLE contains XML documents. Publish these XML documents to the default publisher service (DB2.DEFAULT.PUBLISHER), using the default service policy (DB2.DEFAULT.POLICY). Do not specify a topic or correlation identifier for the message.

```
# SELECT MQPUBLISHXML(ORDER, 't1')
#   FROM ORDER_TABLE;
```

Example 2: The CUSTOMER column in table CUSTOMER_TABLE contains XML documents. Publish the XML documents for the Midwestern customers to the publisher service 'MYPUBLISHER' under the topic '/MIDWEST/CUSTOMERS', using the default service policy (DB2.DEFAULT.POLICY). Do not specify a correlation identifier for the message.

```
# SELECT MQPUBLISHXML('MYPUBLISHER', CUSTOMER, '/MIDWEST/CUSTOMERS')
#   FROM CUSTOMER_TABLE
# WHERE TERRITORY = 'MIDWEST';
```

```
# All of the examples return a value of '1' if they are successful.
```

MQREAD

MQREAD

```
#  
#      ►►MQREAD( [receive-service] [,—service-policy] )  
#
```

The schema is DB2MQ1C or DB2MQ2C

The MQREAD function returns a message from the MQSeries location that is
specified by *receive-service*, using the quality-of-service policy that is defined in
service-policy. Performing this operation does not remove the message from the
queue that is associated with *receive-service*, but instead returns the message at
the beginning of the queue.

receive-service

An expression that returns a value that is a built-in character-string data type
that is not a CLOB. The value of the expression must not be the null value, an
empty string, or a string with trailing blanks. The expression must have an
actual length that is no greater than 48 bytes. The value of the expression
refers to a service point that is the logical MQSeries destination from which the
message is read. A service point is defined in the DSNAMT repository file, and
it represents a logical end-point from which a message is sent or received. A
service point definition includes the name of the MQSeries queue manager and
the name of the queue. See *MQSeries Application Messaging Interface* for
more details.

If *receive-service* is not specified, DB2.DEFAULT.SERVICE is used.

service-policy

An expression that returns a value that is a built-in character-string data type
that is not a CLOB. The value of the expression must not be the null value, an
empty string, or a string with trailing blanks. The expression must have an
actual length that is no greater than 48 bytes. The value of the expression
refers to an MQSeries AMI service policy that is used in handling this message.
A service policy is defined in the DSNAME repository file, and it specifies a set
of quality-of-service options that are to be applied to this messaging operation.
These options include message priority and message persistence. See
MQSeries Application Messaging Interface for more details.

If *service-policy* is not specified, DB2.DEFAULT.POLICY is used.

The result of the function is a varying-length string with a length attribute of 4000.
The result can be null. If no messages are available to be returned, the result is the
null value.

Example 1: Read the message from the queue specified by the default service
(DB2.DEFAULT.SERVICE), using the default policy (DB2.DEFAULT.POLICY).

```
#          SELECT MQREAD()  
#          FROM  table;
```

The message at the head of the queue specified by the default service and using
the default policy is returned as a VARCHAR(4000).

Example 2: Read the message from the queue specified by the service
MYSERVICE, using the policy MYPOLICY.

```
#          SELECT MQREAD('MYSERVICE','MYPOLICY')
#                         FROM  table;
```

The message at the head of the queue specified by MYSERVICE and using policy
MYPOLICY is returned as a CLOB.

MQREADCLOB

MQREADCLOB

```
#  
#  
# ►►MQREADCLOB( receive-service , service-policy )  
#  
#  
#
```

The schema is DB2MQ1C or DB2MQ2C

The MQREADCLOB function returns a message from the MQSeries location that is specified by *receive-service*, using the quality-of-service policy that is defined in *service-policy*. Performing this operation does not remove the message from the queue that is associated with *receive-service*, but instead returns the message at the beginning of the queue.

receive-service

An expression that returns a value that is a built-in character-string data type that is not a CLOB. The value of the expression must not be the null value, an empty string, or a string with trailing blanks. The expression must have an actual length that is no greater than 48 bytes. The value of the expression refers to a service point that is the logical MQSeries destination from which the message is read. A service point is defined in the DSNAMT repository file, and it represents a logical end-point from which a message is sent or received. A service point definition includes the name of the MQSeries queue manager and the name of the queue. See *MQSeries Application Messaging Interface* for more details.

If *receive-service* is not specified, DB2.DEFAULT.SERVICE is used.

service-policy

An expression that returns a value that is a built-in character-string data type that is not a CLOB. The value of the expression must not be the null value, an empty string, or a string with trailing blanks. The expression must have an actual length that is no greater than 48 bytes. The value of the expression refers to an MQSeries AMI service policy that is used in handling this message. A service policy is defined in the DSNAMT repository file, and it specifies a set of quality-of-service options that are to be applied to this messaging operation. These options include message priority and message persistence. See *MQSeries Application Messaging Interface* for more details.

If *service-policy* is not specified, DB2.DEFAULT.POLICY is used.

The result of the function is a CLOB with a length attribute of 1MB. The result can be null. If no messages are available to be returned, the result is the null value.

Example 1: Read the message from the queue specified by the default service (DB2.DEFAULT.SERVICE), using the default policy (DB2.DEFAULT.POLICY).

```
SELECT MQREADCLOB()  
      FROM table;
```

The message at the head of the queue specified by the default service and using the default policy is returned as a CLOB.

Example 2: Read the message from the queue specified by the service MYSERVICE, using the default policy (DB2.DEFAULT.POLICY).

```
#          SELECT MQREADCLOB('MYSERVICE')
#                FROM  table;
```

The message at the head of the queue specified by MYSERVICE and using the
default policy is returned as a CLOB.

MQREADXML

MQREADXML

```
#  
#    ►►►MQREADXML( [receive-service] [,service-policy] )  
#
```

The schema is DMQXML1C or DMQXML2C

The MQREADXML function returns an XML message from the MQSeries location that is specified by *receive-service*, using the quality-of-service policy that is defined in *service-policy*. Performing this operation does not remove the message from the queue that is associated with *receive-service*, but instead returns the message at the beginning of the queue.

receive-service

An expression that returns a value that is a built-in character-string data type
that is not a CLOB. The value of the expression must not be the null value, an
empty string, or a string with trailing blanks. The expression must have an
actual length that is no greater than 48 bytes. The value of the expression
refers to a service point that is the logical MQSeries destination from which the
message is read. A service point is defined in the DSNAMT repository file, and
it represents a logical end-point from which a message is sent or received. A
service point definition includes the name of the MQSeries queue manager and
the name of the queue. See *MQSeries Application Messaging Interface* for
more details.

If *receive-service* is not specified, DB2.DEFAULT.SERVICE is used.

service-policy

An expression that returns a value that is a built-in character-string data type
that is not a CLOB. The value of the expression must not be the null value, an
empty string, or a string with trailing blanks. The expression must have an
actual length that is no greater than 48 bytes. The value of the expression
refers to an MQSeries AMI service policy that is used in handling this message.
A service policy is defined in the DSNAME repository file, and it specifies a set
of quality-of-service options that are to be applied to this messaging operation.
These options include message priority and message persistence. See
MQSeries Application Messaging Interface for more details.

If *service-policy* is not specified, DB2.DEFAULT.POLICY is used.

The result of the function is a value with the user-defined data type
DB2XML.XMLVARCHAR that contains the XML messages. The result can be null. If

The CCSID of the result is the system CCSID that was in effect at the time that the

Example: Read the message from the queue specified by the default service (`DEFAUTL_QUEUE`) using the default service (`DEFAUTL_SERVICE`)

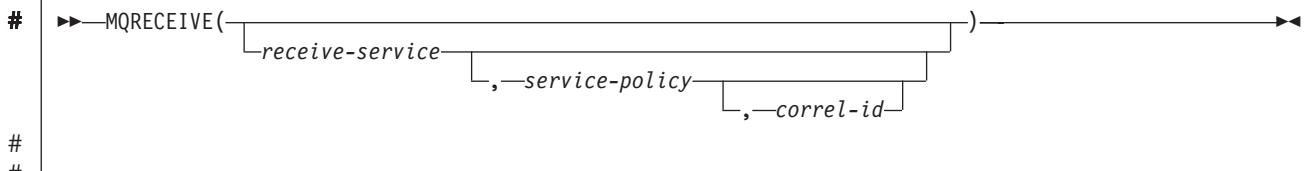
```
#          (DB2.DEFAULT.SERVICE)
#          SELECT MQREADXML()
#          FROM SYSTEM.SQSRVMM
```

The message at the beginning of the queue specified by the default service and
using the default policy is returned as a XML/VARCHAR.

MQRECEIVE

#

#



#

#

#

#

The schema is DB2MQ1C or DB2MQ2C

#

#

#

#

The MQRECEIVE function returns a message from the MQSeries location that is specified by *receive-service*, using the quality-of-service policy that is defined in *service-policy*. Performing this operation removes the message from the queue that is associated with *receive-service*.

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

receive-service

An expression that returns a value that is a built-in character-string data type that is not a CLOB. The value of the expression must not be the null value, an empty string, or a string with trailing blanks. The expression must have an actual length that is no greater than 48 bytes. The value of the expression refers to a service point that is the logical MQSeries destination from which the message is read. A service point is defined in the DSNAME repository file, and it represents a logical end-point from which a message is sent or received. A service point definition includes the name of the MQSeries queue manager and the name of the queue. See *MQSeries Application Messaging Interface* for more details.

#

If *receive-service* is not specified, DB2.DEFAULT.SERVICE is used.

#

service-policy

An expression that returns a value that is a built-in character-string data type that is not a CLOB. The value of the expression must not be the null value, an empty string, or a string with trailing blanks. The expression must have an actual length that is no greater than 48 bytes. The value of the expression refers to an MQSeries AMI service policy that is used in handling this message. A service policy is defined in the DSNAME repository file, and it specifies a set of quality-of-service options that are to be applied to this messaging operation. These options include message priority and message persistence. See *MQSeries Application Messaging Interface* for more details.

#

If *service-policy* is not specified, DB2.DEFAULT.POLICY is used.

#

correl-id

An expression that returns a value that is a built-in string character-string data type that is not a CLOB. The expression must have an actual length that is no greater than 24 bytes. The value of the expression specifies the correlation identifier that is associated with this message. A correlation identifier is often specified in request-and-reply scenarios to associate requests with replies. The first message with a matching correlation identifier is returned.

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

A null value, an empty string, and a fixed length string with trailing blanks are all considered valid values. However, when the *correl-id* is specified on another request such as MQSEND, the *correl-id* must be specified the same to be recognized as a match. For example, specifying a value of 'test' for *correl-id* on MQRECEIVE does not match a *correl-id* value of 'test' (with trailing blanks) specified earlier on an MQSEND request.

MQRECEIVE

```
# If correl-id is not specified, a correlation identifier is not used, and the message  
# at the beginning of the queue is returned.
```

```
# The result of the function is a varying-length string with a length attribute of 4000.  
# The result can be null. If no messages are available to be returned, the result is the  
# null value.
```

```
# Example 1: Retrieve the message from the head of the queue specified by the  
# default service (DB2.DEFAULT.SERVICE), using the default policy  
# (DB2.DEFAULT.POLICY).
```

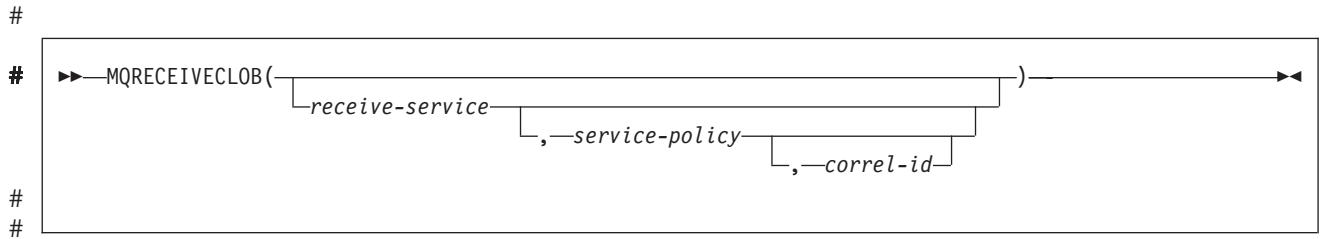
```
#  
# SELECT MQRECEIVE()  
#       FROM table;
```

```
# The message at the head of the queue specified by the default service and using  
# the default policy is returned as VARCHAR(4000) and is deleted from the queue.
```

```
# Example 2: Retrieve the first message with a correlation identifier that matches  
# '1234' from the head of the queue specified by the service MYSERVICE, using the  
# policy MYPOLICY.
```

```
#  
# SELECT MQRECEIVE('MYSERVICE','MYPOLICY','1234')  
#       FROM table;
```

```
# The message with CORRELID of '1234' from the head of the queue specified by  
# MYSERVICE and using MYPOLICY is returned as VARCHAR(4000) and is deleted  
# from the queue.
```

MQRECEIVECLOB

The schema is DB2MQ1C or DB2MQ2C

The MQRECEIVECLOB function returns a message from the MQSeries location
that is specified by *receive-service*, using the quality-of-service policy that is defined
in *service-policy*. Performing this operation removes the message from the queue
that is associated with *receive-service*.

receive-service

An expression that returns a value that is a built-in character-string data type
that is not a CLOB. The value of the expression must not be the null value, an
empty string, or a string with trailing blanks. The expression must have an
actual length that is no greater than 48 bytes. The value of the expression
refers to a service point that is the logical MQSeries destination from which the
message is read. A service point is defined in the DSNAME repository file, and
it represents a logical end-point from which a message is sent or received. A
service point definition includes the name of the MQSeries queue manager and
the name of the queue. See *MQSeries Application Messaging Interface* for
more details.

If *receive-service* is not specified, DB2.DEFAULT.SERVICE is used.

service-policy

An expression that returns a value that is a built-in character-string data type
that is not a CLOB. The value of the expression must not be the null value, an
empty string, or a string with trailing blanks. The expression must have an
actual length that is no greater than 48 bytes. The value of the expression
refers to an MQSeries AMI service policy that is used in handling this message.
A service policy is defined in the DSNAME repository file, and it specifies a set
of quality-of-service options that are to be applied to this messaging operation.
These options include message priority and message persistence. See
MQSeries Application Messaging Interface for more details.

If *service-policy* is not specified, DB2.DEFAULT.POLICY is used.

correl-id

An expression that returns a value that is a built-in string character-string data
type that is not a CLOB. The expression must have an actual length that is no
greater than 24 bytes. The value of the expression specifies the correlation
identifier that is associated with this message. A correlation identifier is often
specified in request-and-reply scenarios to associate requests with replies. The
first message with a matching correlation identifier is returned.

A null value, an empty string, and a fixed length string with trailing blanks are all
considered valid values. However, when the *correl-id* is specified on another
request such as MQSEND, the *correl-id* must be specified the same to be
recognized as a match. For example, specifying a value of 'test' for *correl-id* on
MQRECEIVECLOB does not match a *correl-id* value of 'test' (with trailing
blanks) specified earlier on an MQSEND request.

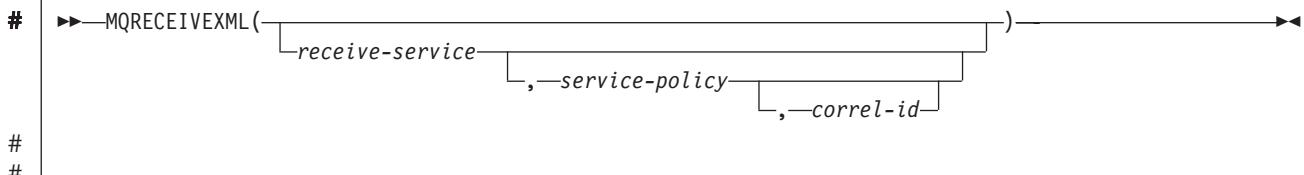
MQRECEIVECLOB

```
# If correl-id is not specified, a correlation identifier is not used, and the message  
# at the beginning of the queue is returned.  
  
# The result of the function is a VARCHAR with a length attribute of 1MB. The result  
# can be null. If no messages are available to be returned, the result is the null value.  
  
# Example 1: Retrieve the message from head of the queue specified by the default  
# service (DB2.DEFAULT.SERVICE), using the default policy  
# (DB2.DEFAULT.POLICY).  
#  
#      SELECT MQRECEIVECLOB()  
#            FROM table;  
  
# The message at the head of the queue specified by the default service and using  
# the default policy is returned as a CLOB and is deleted from the queue.  
  
# Example 2: Retrieve the message from the head of the queue specified by the  
# service MYSERVICE, using the default policy (DB2.DEFAULT.POLICY).  
#  
#      SELECT MQRECEIVECLOB('MYSERVICE')  
#            FROM table;  
  
# The message at the head of the queue specified by MYSERVICE and using the  
# default policy is returned as a CLOB and is deleted from the queue.
```

MQRECEIVEXML

#

#



#

#

#

#

The schema is DMQXML1C or DMQXML2C.

#

#

#

#

The MQRECEIVEXML function returns an XML message from the MQSeries location that is specified by *receive-service*, using the quality-of-service policy that is defined in *service-policy*. Performing this operation removes the message from the queue that is associated with *receive-service*.

#

receive-service

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

MQRECEIVEXML

```
# If correl-id is not specified, a correlation identifier is not used, and the message  
# at the beginning of the queue is returned.  
  
# The result of the function is a value with user-defined data type  
# DB2XML.XMLVARCHAR that contains the XML messages. The result can be null. If  
# no messages are available to be returned, the result is the null value.  
  
# The CCSID of the result is the system CCSID that was in effect at the time that the  
# MQSeries function was installed into DB2.  
  
# Example: Retrieve the message from the beginning of the queue specified by the  
# default service (DB2.DEFAULT.SERVICE), using the default policy  
# (DB2.DEFAULT.POLICY).  
#  
#      SELECT MQRECEIVEXML()  
#            FROM  SYSIBM.SYSDUMMY;  
  
# The message at the beginning of the queue specified by the default service and  
# using the default policy is returned. The message is deleted from the queue.
```

MQSEND

#

```
# ►►—MQSEND(—  
#   |—send-service,—|—msg-data—|—correl-id—|(1)|—  
#   |—service-policy—|  
# )—►►
```

#

Notes:

1 *correl-id* cannot be specified unless a send service and a service policy are also specified.

#

#

The schema is DB2MQ1C or DB2MQ2C

#

The MQSEND function sends the data that is contained in *msg-data* to the

MQSeries location that is specified by *send-service*, using the quality-of-service

policy that is defined in *service-policy*. The function returns a value of '1' if it is

successful or a '0' if it is not successful.

#

send-service

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

An expression that returns a value that is a built-in character-string string data type that is not a CLOB. The value of the expression must not be the null value, an empty string, or a string with trailing blanks. The expression must have an actual length that is no greater than 48 bytes. The value of the expression refers to a service point that is the logical MQSeries destination to which the message is sent. A service point is defined in the DSNAME repository file, and it represents a logical end-point from which a message is sent or received. A service point definition includes the name of the MQSeries queue manager and the name of the queue. See *MQSeries Application Messaging Interface* for more details.

#

If *send-service* is not specified, DB2.DEFAULT.SERVICE is used.

#

service-policy

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

An expression that returns a value that is a built-in character-string string data type that is not a CLOB. The value of the expression must not be the null value, an empty string, or a string with trailing blanks. The expression must have an actual length that is no greater than 48 bytes. The value of the expression specifies an MQSeries AMI service policy that is used in handling this message. A service policy is defined in the DSNAME repository file, and it specifies a set of quality-of-service options that are to be applied to this messaging operation. These options include message priority and message persistence. See *MQSeries Application Messaging Interface* for more details.

#

If *service-policy* is not specified, DB2.DEFAULT.POLICY is used.

#

msg-data

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

MQSEND

```
# specified in request-and-reply scenarios to associate requests with replies.  
# correl-id must not be specified unless send-service and service-policy are also  
# specified.  
# A null value, an empty string, and a fixed length string with trailing blanks are all  
# considered valid values. However, when the correl-id is specified on another  
# request such as MQRECEIVE, the correl-id must be specified the same to be  
# recognized as a match. For example, specifying a value of 'test' for correl-id on  
# MQSEND does not match a correl-id value of 'test ' (with trailing blanks)  
# specified subsequently on an MQRECEIVE request.  
# If correl-id is not specified, a correlation identifier is not sent.  
#  
# The result of the function is a varying-length string with a length attribute of 1. The  
# data type of the result is nullable, even though a null value is never returned.  
#  
# Example 1: Send the string "Testing msg" to the default service  
# (DB2.DEFAULT.SERVICE), using the default policy (DB2.DEFAULT.POLICY) and no  
# correlation identifier.  
#  
# SELECT MQSEND('Testing msg')  
#      FROM table;  
#  
# The message is sent to the default service, using the default policy.  
#  
# Example 2: Send the string "Testing 123" to the service MYSERVICE, using the  
# policy MYPOLICY and the correlation identifier "TEST3".  
#  
# SELECT MQSEND('MYSERVICE','MYPOLICY','Testing 123','TEST3')  
#      FROM table;  
#  
# The string "TESTING 123" is sent to MYSERVICE, using MYPOLICY and the  
# correlation identifier "TEST3".
```

MQSENDXML

#

```
# ►►MQSENDXML( send-service , service-policy , msg-data , correl-id (1) )►►
#
# Notes:
# 1 correl-id cannot be specified unless a send service and a service policy are also specified.
```

#

#

The schema is DMQXML1C or DMQXML2C.

The MQSENDXML function sends the XML data that is contained in *msg-data* to the MQSeries location that is specified by *send-service*, using the quality-of-service policy that is defined in *service-policy*. The function returns a value of '1' if it is successful or a '0' if it is unsuccessful.

send-service

An expression that returns a value that is a built-in character-string string data type that is not a CLOB. The value of the expression must not be the null value, an empty string, or a string with trailing blanks. The expression must have an actual length that is no greater than 48 bytes. The value of the expression refers to a service point that is the logical MQSeries destination to which the message is sent. A service point is defined in the DSNAME repository file, and it represents a logical end-point from which a message is sent or received. A service point definition includes the name of the MQSeries queue manager and the name of the queue. See *MQSeries Application Messaging Interface* for more details.

If *send-service* is not specified, DB2.DEFAULT.SERVICE is used.

service-policy

An expression that returns a value that is a built-in character-string string data type that is not a CLOB. The value of the expression must not be the null value, an empty string, or a string with trailing blanks. The expression must have an actual length that is no greater than 48 bytes. The value of the expression specifies an MQSeries AMI service policy that is used in handling this message. A service policy is defined in the DSNAME repository file, and it specifies a set of quality-of-service options that are to be applied to this messaging operation. These options include message priority and message persistence. See *MQSeries Application Messaging Interface* for more details.

If *service-policy* is not specified, DB2.DEFAULT.POLICY is used.

msg-data

An expression that returns a value with a user-defined data type of DB2XML.XMVARCHAR (for data up to 3k in size) or DB2XML.XMLCLOB (for data up to 1M in size). The value of the expression specifies the message data to be sent via MQSeries.

correl-id

An expression that returns a value that is a built-in character-string data type that is not a CLOB. The expression must have an actual length that is no greater than 24 bytes. The value of the expression specifies the correlation identifier to be associated with this message. The *correl-id* is often specified in request-and-reply scenarios to associate requests with replies.

MQSENDXML

```
# A null value, an empty string, and a fixed length string with trailing blanks are all  
# considered valid values. However, when the correl-id is specified on another  
# request such as MQSENDXML, the correl-id must be specified the same to be  
# recognized as a match. For example, specifying a value of 'test' for correl-id on  
# MQRECEIVEMXL does not match a correl-id value of 'test   ' (with trailing  
# blanks) specified subsequently on an MQSENDXML request.  
#  
# If correl-id is not specified, a correlation identifier is not sent.  
#  
# The result of the function is a varying-length string with a length attribute of 1. The  
# data type of the result is nullable, even though a null value is never returned.  
#  
# The CCSID of the result is the system CCSID that was in effect at the time that the  
# MQSeries function was installed into DB2.  
#  
# Example 1: The ORDER column in table ORDER_TABLE contains XML documents.  
# Send the XML documents in this column to the default service  
# (DB2.DEFAULT.SERVICE), using the default policy (DB2.DEFAULT.POLICY). Do  
# not specify a correlation identifier.  
#  
# SELECT MQSENDXML(ORDER)  
#      FROM ORDER_TABLE;  
#  
# The message is sent to the default service, using the default policy.  
#  
# Example 2: The CUSTOMER column in table CUSTOMER_TABLE contains XML  
# documents. Send the XML documents for the Midwestern customers to the service  
# 'MYSERVICE', using the default service policy (DB2.DEFAULT.POLICY) and a  
# correlation identifier of "MIDWESTERN".  
#  
# SELECT MQSENDXML('MYSERVICE', 'MYPOLICY', CUSTOMER, 'MIDWESTERN')  
#      FROM CUSTOMER_TABLE  
#      WHERE TERRITORY = 'MIDWESTERN';  
#  
# All of the examples return a value of '1' if they are successful.
```


MQSENDXMLFILE

```
# recognized as a match. For example, specifying a value of 'test' for correl-id on
# MQRECEIVEXML does not match a correl-id value of 'test   ' (with trailing
# blanks) specified subsequently on an MQSENDXML request.
#
# If correl-id is not specified, a correlation identifier is not sent.

#
# The result of the function is a varying-length string with a length attribute of 1. The
# data type of the result is nullable, even though a null value is never returned.

#
# The CCSID of the result is the system CCSID that was in effect at the time that the
# MQSeries function was installed into DB2.

#
# Example 1: Send the XML documents that are in file '/tmp/test1.xml' to the default
# service (DB2.DEFAULT.SERVICE), using the default policy
# (DB2.DEFAULT.POLICY). Do not specify a correlation identifier.
#
#      SELECT MQSENDXMLFILE('/tmp/test1.xml')
#            FROM  SYSIBM.SYSDUMMY1;

#
# Example 2: Send the XML documents that are in file '/tmp/test2.xml' to the service
# 'MYSERVICE', using the service policy 'MYPOLICY' and the correlation identifier
# 'Test 2'.
#
#      SELECT MQSENDXMLFILE('MYSERVICE', 'MYPOLICY', CUSTOMER, '/tmp/test2.xml','Test2')
#            FROM  SYSIBM.SYSDUMMY1;

#
# All of the examples return a value of '1' if they are successful.
```

MQSENDXMLFILECLOB

#

```
#    ►►—MQSENDXMLFILECLOB(—send-service—, —service-policy—, —xml-file—, —correl-id— (1) —►►
```

#

Notes:

```
# 1     correl-id cannot be specified unless a send service and a service policy are also specified.
```

#

#

The schema is DMQXML1C or DMQXML2C.

The MQSENDXMLFILECLOB function sends the XML data that is contained in *xml-file* to the MQSeries location that is specified by *send-service*, using the quality-of-service policy that is defined in *service-policy*. The function returns a value of '1' if it is successful or a '0' if it is unsuccessful.

send-service

An expression that returns a value that is a built-in character-string string data type that is not a CLOB. The value of the expression must not be the null value, an empty string, or a string with trailing blanks. The expression must have an actual length that is no greater than 48 bytes. The value of the expression refers to a service point that is the logical MQSeries destination to which the message is sent. A service point is defined in the DSNAME repository file, and it represents a logical end-point from which a message is sent or received. A service point definition includes the name of the MQSeries queue manager and the name of the queue. See *MQSeries Application Messaging Interface* for more details.

If *send-service* is not specified, DB2.DEFAULT.SERVICE is used.

service-policy

An expression that returns a value that is a built-in character-string string data type that is not a CLOB. The value of the expression must not be the null value, an empty string, or a string with trailing blanks. The expression must have an actual length that is no greater than 48 bytes. The value of the expression specifies an MQSeries AMI service policy that is used in handling this message. A service policy is defined in the DSNAME repository file, and it specifies a set of quality-of-service options that are to be applied to this messaging operation. These options include message priority and message persistence. See *MQSeries Application Messaging Interface* for more details.

If *service-policy* is not specified, DB2.DEFAULT.POLICY is used.

xml-file

An expression that returns the name of file that contains the XML data that is to be sent via MQSeries. The value must be no longer than 48 bytes.

correl-id

An expression that returns a value that is a built-in character-string data type that is not a CLOB. The expression must have an actual length that is no greater than 24 bytes. The value of the expression specifies the correlation identifier to be associated with this message. The *correl-id* is often specified in request-and-reply scenarios to associate requests with replies.

A null value, an empty string, and a fixed length string with trailing blanks are all considered valid values. However, when the *correl-id* is specified on another request such as MQSENDXML, the *correl-id* must be specified the same to be

MQSENDXMLFILECLOB

```
# recognized as a match. For example, specifying a value of 'test' for correl-id on
# MQRECEIVEXML does not match a correl-id value of 'test   ' (with trailing
# blanks) specified subsequently on an MQSENDXML request.
#
# If correl-id is not specified, a correlation identifier is not sent.
#
# The result of the function is a varying-length string with a length attribute of 1. The
# data type of the result is nullable, even though a null value is never returned.
#
# The CCSID of the result is the system CCSID that was in effect at the time that the
# MQSeries function was installed into DB2.
#
# Example: Send the XML documents that are in file '/tmp/test1.xml' to the default
# service (DB2.DEFAULT.SERVICE), using the default policy
# (DB2.DEFAULT.POLICY). Do not specify a correlation identifier.
#
#     SELECT MQSENDXMLFILECLOB('/tmp/test1.xml')
#           FROM  SYSIBM.SYSDUMMY1;
#
# The example returns a value of '1' if it is successful.
```

MQSUBSCRIBE

#

```
# ►►MQSUBSCRIBE( [subscriber-service], [service-policy], [topic-list])►►
```

#

#

#

The schema is DB2MQ1C or DB2MQ2C

The MQSUBSCRIBE function is used to register interest in MQSeries messages published on specified topics. The *subscriber-service* specifies a logical destination for messages that match the specified topics. Messages that match the specified topics are placed on the queue defined by *subscriber-service*. The messages can then be read or received through a subsequent invocation of the MQREAD, MQRECEIVE, MQREADALL, or MQRECEIVEALL functions. The MQSUBSCRIBE function requires the installation and configuration of an MQSeries-based publish and subscribe system, such as Websphere MQSeries Integrator. See www.ibm.com/software/MQSeries for more details.

The function returns a value of '1' if successful or '0' if unsuccessful. A successful execution of this function causes the publish and subscribe server to forward messages that match the specified topics to the service point defined by *subscriber-service*.

subscriber-service

An expression that returns a value that is a built-in character-string data type that is not a CLOB. The value of the expression must not be the null value, an empty string, or a string with trailing blanks. The expression must have an actual length that is no greater than 48 bytes. The value of the expression refers to a service point that is the logical MQSeries subscription point to which messages that match the specified topics are sent. A subscriber's service point is defined in the DSNAMT repository file, and it specifies a logical subscription point to which a message is sent. A service point definition includes the name of the MQSeries queue manager and the name of the queue. See *MQSeries Application Messaging Interface* for more details.

If *subscriber-service* is not specified, the DB2.DEFAULT.SUBSCRIBER is used.

service-policy

An expression that returns a value that is a built-in character-string data type that is not a CLOB. The value of the expression must not be the null value, an empty string, or a string with trailing blanks. The expression must have an actual length that is no greater than 48 bytes. The value of the expression refers to the MQSeries AMI service policy that is used in handling this message. A service policy is defined in the DSNAMT repository file, and it specifies a set of quality-of-service options that are to be applied to this messaging operation. These options include message priority and message persistence. See *MQSeries Application Messaging Interface* for more details.

If *service-policy* is not specified, the DB2.DEFAULT.POLICY is used.

topic-list

An expression that returns a value that is a built-in character-string data type that is not a CLOB. The value of the expression must not be the null value, an empty string, or a string with trailing blanks. The expression must have an actual length that is no greater than 40 bytes. The value of the expression

MQSUBSCRIBE

```
# specifies the types of messages to receive. Only messages published with the
# specified topics are received by this subscription. Multiple subscriptions can
# coexist. One or more topics can be specified, where the topics are separated
# with a colon. For example, 't1:t2:the third topic' indicates that the message is
# associated with all three topics: t1, t2, and 'the third topic'.
```

Example 1: Register an interest in messages that contain the topic 'Weather'. Allow
the message to be sent to the default subscriber service point
(DB2.DEFAULT.SUBSCRIBER), using the default service policy.

SELECT DB2MQ2C.MQSUBSCRIBE('Weather')
FROM SYSIBM.SYSDUMMY1;

Example 2: Registering an interest in messages that contain the topic 'Stocks'.
Specify "PORTFOLIO-UPDATES" as the subscriber service point to which the
message is to be sent, using service policy 'BASIC-POLICY'.

SELECT DB2MQ2C.MQSUBSCRIBE ('PORTFOLIO-UPDATES','BASIC-POLICY','Stocks')
FROM SYSIBM.SYSDUMMY1;

All of the examples return a value of '1' if they are successful.

MQUNSUBSCRIBE

#

```
# ─►—MQUNSUBSCRIBE(—  
    subscriber-service,—  
    service-policy,—  
    topic-list)—►
```

#

#

#

The schema is DB2MQ1C or DB2MQ2C

The MQUNSUBSCRIBE function is used to unregister an existing message
subscription. The *subscriber-service*, *service-policy*, and *topic-list* identify which
subscription is canceled. The MQUNSUBSCRIBE function requires the installation
and configuration of an MQSeries-based publish and subscribe system, such as
Websphere MQSeries Integrator. See www.ibm.com/software/MQSeries for more
details.

The function returns a value of '1' if successful or '0' if unsuccessful. A successful
execution of this function causes the publish and subscribe server to remove the
subscription that is identified by the given parameters. Messages that match the
specified topics are no longer sent to the service point defined by
subscriber-service.

subscriber-service

An expression that returns a value that is a built-in character string data type
that is not a CLOB. The value of the expression must not be the null value, an
empty string, or a string with trailing blanks. The expression must have an
actual length that is no greater than 48 bytes. The value of the expression
refers to a service point that is the logical MQSeries subscription point to which
messages that match the specified topics are sent. A subscriber's service point
is defined in the DSNAMT repository file, and it specifies a logical subscription
point to which a message is sent. A service point definition includes the name of
the MQSeries queue manager and the name of the queue. See *MQSeries*
Application Messaging Interface for more details.

If *subscriber-service* is not specified, the DB2.DEFAULT.SUBSCRIBER is used.# *service-policy*

An expression that returns a value that is a built-in character string data type
that is not a CLOB. The value of the expression must not be the null value, an
empty string, or a string with trailing blanks. The expression must have an
actual length that is no greater than 48 bytes. The value of the expression
refers to the MQSeries AMI service policy that is used in handling this message.
A service policy is defined in the DSNAMT repository file, and it specifies a set
of quality-of-service options that are to be applied to this messaging operation.
These options include message priority and message persistence. See
MQSeries Application Messaging Interface for more details.

If *service-policy* is not specified, the DB2.DEFAULT.POLICY is used.# *topic-list*

An expression that returns a value that is a built-in character string data type
that is not a CLOB. The value of the expression must not be the null value, an
empty string, or a string with trailing blanks. The expression must have an
actual length that is no greater than 40 bytes. The value of the expression
specifies the types of messages to receive. Only messages published with the
specified topics are received by this subscription. Multiple subscriptions can

MQUNSUBSCRIBE

```
#          coexist. One or more topics can be specified, where the topics are separated  
#          with a colon. For example, 't1:t2:the third topic' indicates that the message is  
#          associated with all three topics: t1, t2, and 'the third topic'.
```

```
#          Example 1: Cancel the subscription for messages that contain the topic 'Weather'.  
#          The subscriber is registered with the default subscriber service point  
#          (DB2.DEFAULT.SERVICE) with the default service policy  
#          (DB2.DEFAULT.SERVICE).  
#          SELECT DB2MQ2C.UNMQSUBSCRIBE('Weather')  
#              FROM SYSIBM.SYSDUMMY1;
```

```
#          Example 2: Cancel the subscription for messages that contain the topic 'Stocks'.  
#          The subscriber is registered with the subscriber service point 'PORTFOLIO-  
#          UPDATES' with the service policy 'BASIC-POLICY'.  
#          SELECT DB2MQ2C.MQUNSUBSCRIBE ('PORTFOLIO-UPDATES','BASIC-POLICY','Stocks')  
#              FROM SYSIBM.SYSDUMMY1;
```

```
#          All of the examples return a value of '1' if they are successful.
```

MULTIPLY_ALT

►►—MULTIPLY_ALT(exact-numeric-expression,exact-numeric-expression)————►►

The schema is SYSIBM.

The MULTIPLY_ALT scalar function returns the product of the two arguments as a decimal value. It is provided as an alternative to the multiplication operator, especially when the sum of the precisions of the arguments exceeds 31.

The arguments can be any built-in exact numeric data type (DECIMAL, INTEGER, or SMALLINT).

The result of the function is a DECIMAL. The precision and scale of the result are determined as follows, using the symbols p and s to denote the precision and scale of the first argument, and the symbols p' and s' to denote the precision and scale of the second argument.

- The precision is $\text{MIN}(31, p+p')$
- The scale is:
 - 0 if the scale of both arguments is 0
 - $\text{MIN}(31, s+s')$ if $p+p'$ is less than or equal to 31
 - $\text{MAX}(\text{MIN}(3, s+s'), 31-(p-s+p'-s'))$ if $p+p'$ is greater than 31.

The result can be null if at least one argument can be null; the result is the null value if one of the arguments is null.

The MULTIPLY_ALT function is a better choice than the multiplication operator when performing decimal arithmetic where a scale of at least 3 is desired and the sum of the precisions exceeds 31. In these cases, the internal computation is performed so that overflows are avoided and then assigned to the result type value using truncation for any loss of scale in the final result. Note that the possibility of overflow of the final result is still possible when the scale is 3.

The following table compares the result types using MULTIPLY_ALT and the multiplication operator:

Type of Argument1	Type of Argument2	Result using MULTIPLY_ALT	Result using multiplication operator
DECIMAL(31,3)	DECIMAL(15,8)	DECIMAL(31,3)	DECIMAL(31,11)
DECIMAL(26,23)	DECIMAL(10,1)	DECIMAL(31,19)	DECIMAL(31,24)
DECIMAL(18,17)	DECIMAL(20,19)	DECIMAL(31,29)	DECIMAL(31,31)
DECIMAL(16,3)	DECIMAL(17,8)	DECIMAL(31,9)	DECIMAL(31,11)
DECIMAL(26,5)	DECIMAL(11,0)	DECIMAL(31,3)	DECIMAL(31,5)
DECIMAL(21,1)	DECIMAL(15,1)	DECIMAL(31,2)	DECIMAL(31,2)

NEXT_DAY

NEXT_DAY

```
►►NEXT_DAY(datetime-expression,expression)►►
```

The schema is SYSIBM.

The NEXT_DAY scalar function returns a timestamp that represents the first weekday, named by *expression*, that is later than the date *datetime-expression*. If *datetime-expression* is a timestamp or valid string representation of a timestamp, the timestamp value has the same hours, minutes, seconds, and microseconds as *expression*. If *datetime-expression* is a date, or a valid string representation of a date, then the hours, minutes, seconds, and microseconds value of the result is 0.

The result of the function is a TIMESTAMP. The result can be null; if any argument is null, the result is the null value.

datetime-expression must be a date, a timestamp, or a valid string representation of a date or timestamp. A string representation must not be a BLOB, CLOB, or DBCLOB, and must have an actual length that is not greater than 255 bytes. For the valid formats of string representations of dates and timestamps, see “String representations of datetime values” on page 57.

expression is the day of the week specified as follows:

Day of week	Abbreviation
MONDAY	MON
TUESDAY	TUE
WEDNESDAY	WED
THURSDAY	THU
FRIDAY	FRI
SATURDAY	SAT
SUNDAY	SUN

expression can be specified as either the full name or the abbreviation. The minimum length of the input is the length of the abbreviated version. Any characters immediately following a valid abbreviation are ignored.

Example 1: Set the host variable NEXTDAY with the date of the Tuesday following April 24, 2000.

```
SET :NEXTDAY = NEXT_DAY(CURRENT_DATE, 'TUESDAY');
```

The host variable NEXTDAY is set with the value of '2000-04-25-00.00.00.000000', assuming that the value of the CURRENT_DATE special register is '2000-04-24'.

Example 2: Set the host variable NEXTDAY with the date of the first Monday in May, 2000. Assume the host variable DAYHV = 'MON'.

```
SET :NEXTDAY = NEXT_DAY(LAST_DAY(CURRENT_TIMESTAMP),:DAYHV);
```

The host variable NEXTDAY is set with the value of '2000-05-01-12.01.01.123456', assuming that the value of the CURRENT_TIMESTAMP special register is '2000-04-24-12.01.01.123456'.

NULLIF

```
►►NULLIF(expression,expression)►►
```

The schema is SYSIBM.

The NULLIF function returns null if the two arguments are equal; otherwise, it returns the value of the first argument.

The two arguments must be compatible. (See the compatibility matrix in Table 9 on page 65.) Neither argument can be a BLOB, CLOB, or DBCLOB. The attributes of the result are the attributes of the first argument. Any numbers specified must be of a built-in numeric data type.

For example, if the result of the first argument is a character string, the result of the other must also be a character string; if the result of the first argument is number, the result of the other must also be a number.

The result of using NULLIF(e1,e2) is the same as using the CASE expression:

```
CASE WHEN e1=e2 THEN NULL ELSE e1 END
```

When e1=e2 evaluates to unknown because one or both arguments is null, CASE expressions consider the evaluation not true. In this case, NULLIF returns the value of the first argument.

Example: Assume that host variables PROFIT, CASH, and LOSSES have decimal data types with the values of 4500.00, 500.00, and 5000.00 respectively. The following function returns a null value:

```
NULLIF (:PROFIT + :CASH , :LOSSES)
```

POSSTR

```
►►POSSTR(source-string,search-string)—►►
```

The schema is SYSIBM.

The POSSTR function returns the starting position of the first occurrence of one string (the *search-string*) within another string (the *source-string*). Numbers for the starting position begin at 1 and not 0.

source-string

An expression that specifies the source string that is to be searched. The source string can be a character, graphic, or binary string. The expression can be specified by any of the following:

- A constant
- A special register
- A host variable (including a LOB locator variable)
- A scalar function whose arguments are any of the above
- A column name
- A CAST specification whose arguments are any of the above
- An expression that concatenates (using CONCAT or II) any of the above

search-string

An expression that specifies the string that is to be searched for. The search string can be a character, graphic, or binary string with an actual length that is no greater than 4000 bytes. The expression can be specified by any of the following:

- A constant
- A special register
- A host variable (including a LOB locator variable)
- A scalar function whose arguments are any of the above
- A CAST specification whose arguments are any of the above
- An expression that concatenates (using CONCAT or II) any of the above

These rules are similar to those that are described for *pattern-expression* for the LIKE predicate.

The first and second arguments must have compatible string types. For more information on compatibility, see “Conversion rules for string comparison” on page 73.

Both *search-string* and *source-string* have zero or more contiguous positions. For character strings and binary strings, a position is a byte. For graphic strings, a position is a DBCS character. Graphic Unicode data is treated as UTF-16 data; a UTF-surrogate character takes two DBCS characters to represent and as such is counted as two DBCS characters.

The strings can contain mixed data.

- For ASCII data, if the search string or source string contains mixed data, the search string is found only if the same combination of single-byte and double-byte characters are found in the source string in exactly the same positions.

- For EBCDIC data, if the search string or source string contains mixed data, the search string is found only if any shift-in or shift-out characters are found in the source string in exactly the same positions, ignoring any redundant shift characters.
- For UTF-8 data, if the search string or source string contains mixed data, the search string is found only if the same combination of single-byte and multi-byte characters are found in the source string in exactly the same position.

The result of the function is a large integer. If either of the arguments can be null, the result can be null; if either of the arguments are null, the result is the null value. The value of the result is determined by applying these rules in the order in which they appear:

- If the length of the search string is zero, the result is 1.
- If the length of the source string is zero, the result is 0.
- If the value of the search string is equal to an identical length substring of contiguous positions from the value of the source string, the result is the starting position of the first such substring within the value of the source string.
- If none of the above conditions are met, the result is 0.

Example: Select the RECEIVED column, the SUBJECT column, and the starting position of the string 'GOOD BEER' within the NOTE_TEXT column for all rows in the IN_TRAY table that contain that string.

```
SELECT RECEIVED, SUBJECT, POSSTR(NOTE_TEXT, 'GOOD BEER')
  FROM IN_TRAY
 WHERE POSSTR(NOTE_TEXT, 'GOOD BEER') <> 0;
```

POWER

POWER

```
►►POWER(expression1,expression2)—►►
```

The schema is SYSIBM.

The POWER function returns the value of *expression1* to the power of *expression2*.

Each argument is an expression that returns the value of any built-in numeric data type. If either argument includes a DECIMAL or REAL data type, the arguments are converted to a double precision floating-point number for processing by the function.

The result of the function depends on the data type of the arguments:

- If both arguments are SMALLINT or INTEGER, the result is INTEGER.
- Otherwise, the result is DOUBLE.

The result can be null; if any argument is null, the result is the null value.

Example 1: Assume that host variable HPOWER is INTEGER with a value of 3. The following statement:

```
SELECT POWER(2,:HPOWER)
      FROM SYSIBM.SYSDUMMY1;
```

returns the value 8.

Example 2: The following statement:

```
SELECT POWER(0,0)
      FROM SYSIBM.SYSDUMMY1;
```

returns the value 1.

QUARTER

►►QUARTER(*expression*)—►►

The schema is SYSIBM.

The QUARTER function returns an integer in the range of 1 to 4 that represents the quarter of the year in which the date occurs. For example, the function returns a 1 for any dates in January, February, or March.

The argument must be a date, a timestamp, or a valid string representation of a date or timestamp. A string representation must not be a BLOB, CLOB, or DBCLOB and must have an actual length that is not greater than 255 bytes. (For the valid formats of string representations of dates and timestamps, see “String representations of datetime values” on page 57.)

The result of the function is a large integer. If the argument can be null, the result can be null; if the argument is null, the result is the null value.

Example 1: The following function returns 3 because August is in the third quarter of the year.

```
SELECT QUARTER('1996-08-25')
  FROM SYSIBM.SYSDUMMY1
```

Example 2: Using sample table DSN8710.PROJ, set the integer host variable QUART to the quarter of the year in which activity number 70 for project 'AD3111' occurred. Activity completion dates are recorded in column ACENDATE.

```
SELECT QUARTER(ACENDATE)
  INTO :QUART
  FROM DSN8710.PROJ
 WHERE PROJNO = 'AD3111' AND ACTNO = 70;
```

QUART is set to 4.

RADIANS

RADIANS

```
►►RADIANS(expression)—————►►
```

The schema is SYSIBM.

The RADIANS function returns the number of radians for an argument that is expressed in degrees.

The argument is an expression that returns the value of any built-in numeric data type. If the argument is not a double precision floating-point number, it is converted to one for processing by the function.

The result of the function is a double precision floating-point number. The result can be null; if the argument is null, the result is the null value.

Example: Assume that host variable HDEG is an INTEGER with a value of 180.

The following statement:

```
SELECT RADIANS(:HDEG)
   FROM SYSIBM.SYSDUMMY1;
```

returns a double precision floating-point number with an approximate value of 3.1415926536.

RAISE_ERROR

```
►►—RAISE_ERROR(sqlstate,diagnostic-string)—►►
```

The schema is SYSIBM.

The RAISE_ERROR function causes the statement that invokes the function to return an error with the specified SQLSTATE (along with SQLCODE -438) and error condition. The RAISE_ERROR function always returns NULL with an undefined data type.

sqlstate

An expression that returns a character string (CHAR or VARCHAR) of exactly 5 characters. The *sqlstate* value must follow these rules for application-defined SQLSTATEs:

- Each character must be from the set of digits ('0' through '9') or non-accented upper case letters ('A' through 'Z').
- The SQLSTATE class (first two characters) cannot be '00', '01', or '02' because these are not error classes.
- If the SQLSTATE class (first two characters) starts with the character '0' through '6' or 'A' through 'H', the subclass (last three characters) must start with a letter in the range 'I' through 'Z'.
- If the SQLSTATE class (first two characters) starts with the character '7', '8', '9', or 'I' through 'Z', the subclass (last three characters) can be any of '0' through '9' or 'A' through 'Z'.

diagnostic-string

An expression that returns a character string with a data type of CHAR or VARCHAR and a length of up to 70 bytes. The string contains EBCDIC data that describes the error condition. If the string is longer than 70 bytes, it is truncated.

To use this function in a context where “Rules for result data types” on page 77 do not apply, such as alone in a select list, you must use a cast specification to give a data type to the null value that is returned. The RAISE_ERROR function is most useful with CASE expressions.

Example: For each employee in sample table DSN8710.EMP, list the employee number and education level. List the education level as Post Graduate, Graduate and Diploma instead of the integer that it is stored as in the table. If an education level is greater than 20, raise an error ('70001') with a description.

```
SELECT EMPNO,
CASE WHEN EDLEVEL < 16 THEN 'Diploma'
      WHEN EDLEVEL < 18 THEN 'Graduate'
      WHEN EDLEVEL < 21 THEN 'Post Graduate'
      ELSE RAISE_ERROR('70001',
                       'EDUCLVL has a value greater than 20')
END
FROM DSN8710.EMP;
```

RAND

RAND

```
►►RAND(expression)►►
```

The schema is SYSIBM.

The RAND function returns a random floating-point value between 0 and 1. An argument can be used as an optional seed value. The function is defined as not-deterministic.

If an argument is specified, it must be an integer (SMALLINT or INTEGER) between 0 and 2 147 483 646.

The result of the function is a double precision floating-point number. The result can be null; if the argument is null, the result is the null value.

Example: Assume that host variable HRAND is an INTEGER with a value of 100. The following statement:

```
SELECT RAND(:HRAND)  
      FROM SYSIBM.SYSDUMMY1;
```

returns a random floating-point number between 0 and 1, such as the approximate value .0121398.

To generate values in a numeric interval other than 0 to 1, multiply the RAND function by the size of the desired interval. For example, to get a random number between 0 and 10, such as the approximate value 5.8731398, multiply the function by 10:

```
SELECT (RAND(:HRAND) * 10)  
      FROM SYSIBM.SYSDUMMY1;
```

REAL

```
►►REAL( [ numeric-expression ]
      [ string-expression ] )
```

The schema is SYSIBM.

The REAL function returns a single precision floating-point representation of a number or string in the form of a numeric constant.

numeric-expression

The argument is an expression that returns a value of any built-in numeric data type.

The result of the function is a single precision floating-point number. The result is the same number that would occur if the argument were assigned to a single precision floating-point column or variable. If the numeric value of the argument is not within the range of single precision floating-point, an error occurs.

string-expression

An expression that returns any type of string (except a BLOB, CLOB, or DBCLOB) with an actual length that is not greater than 255 bytes. Leading and trailing blanks are removed from the string, and the resulting substring must conform to the rules for forming a string representation of an SQL floating-point constant.

The result of the function is a single precision floating-point number. The result is the same number that would occur if the corresponding numeric constant were assigned to a single precision floating-point column or variable.

If the argument can be null, the result can be null; if the argument is null, the result is the null value.

Example: Using sample table DSN8710.EMP, find the ratio of salary to commission for employees whose commission is not zero. The columns involved, SALARY and COMM, have decimal data types. To express the result in single precision floating-point, apply REAL to SALARY so that the division is carried out in floating-point (actually double precision) and then apply REAL to the complete expression so that the results are returned in single precision floating-point.

```
SELECT EMPNO, REAL(REAL(SALARY)/COMM)
   FROM DSN8710.EMP
 WHERE COMM > 0;
```

REPEAT

REPEAT

```
►►REPEAT(expression,integer)—►►
```

The schema is SYSIBM.

The REPEAT function returns a string composed of *expression* repeated *integer* times.

expression

An expression that specifies the string to be repeated. The string must be any type of character string except a CLOB, or any type of graphic string except a DBCLOB. The actual length of the string must be 32767 bytes or less.

integer

An expression whose value is a positive integer. The integer specifies the number of times to repeat the string.

The result of the function depends on the data type of the first argument:

- VARCHAR if *expression* is a character string
- VARGRAPHIC if *expression* is graphic string

If *integer* is a constant, the length attribute of the result is the length attribute of *expression* times *integer*. Otherwise, the length attribute depends on the data type of the result:

- 4000 for VARCHAR
- 2000 for VARGRAPHIC

The actual length of the result is the actual length of *expression* times *integer*. If the actual length of the result string exceeds the maximum for the return type, an error occurs.

If any argument can be null, the result can be null; if any argument is null, the result is the null value.

The subtype and CCSID of the result are determined as follows:

- If *expression* is character bit data, the result is bit data and does not have an associated CCSID.
- If *expression* is character SBCS data, the result is SBCS data and the CCSID is the CCSID for ASCII, EBCDIC, or Unicode SBCS data, depending on the encoding scheme of the SQL statement.
- If *expression* is graphic data, the result is graphic data and the CCSID is the CCSID for ASCII, EBCDIC, or Unicode graphic data, depending on the encoding scheme of the SQL statement.
- Otherwise, the result is mixed data. The CCSID is the CCSID for ASCII, EBCDIC, or Unicode mixed data, depending on the encoding scheme of the SQL statement.

Example 1: Repeat 'abc' two times to create 'abcabc'.

```
SELECT REPEAT('abc',2)  
  FROM SYSIBM.SYSDUMMY1;
```

Example 2: List the phrase 'REPEAT THIS' five times. Use the CHAR function to limit the output to 60 bytes.

```
SELECT CHAR(REPEAT('REPEAT THIS',5), 60)
      FROM SYSIBM.SYSDUMMY1;
```

This example results in 'REPEAT THISREPEAT THISREPEAT THISREPEAT THISREPEAT THIS'.

Example 3: For the following query, the LENGTH function returns a value of 0 because the result of repeating a string zero times is an empty string, which is a zero-length string.

```
SELECT LENGTH(REPEAT('REPEAT THIS',0))
      FROM SYSIBM.SYSDUMMY1;
```

Example 4: For the following query, the LENGTH function returns a value of 0 because the result of repeating an empty string any number of times is an empty string, which is a zero-length string.

```
SELECT LENGTH(REPEAT('', 5))
      FROM SYSIBM.SYSDUMMY1;
```

REPLACE

REPLACE

```
►►REPLACE(expression1,expression2,expression3)►►
```

The schema is SYSIBM.

The REPLACE function replaces all occurrences of *expression2* in *expression1* with *expression3*. If *expression2* is not found in *expression1*, *expression1* is returned unchanged.

expression1

An expression that specifies the source string. The expression cannot be an empty string.

expression2

An expression that specifies the string to be removed from the source string. The expression cannot be an empty string.

expression3

An expression that specifies the replacement string.

The arguments must have compatible string types. For more information on compatibility, see “Conversion rules for string comparison” on page 73. If the arguments are character strings, none must be a CLOB. If the arguments are graphic strings, none must be a DBCLOB. The actual length of each string must be 32767 bytes or less.

The result of the function depends on the data type of the arguments:

- VARCHAR if the arguments are character strings
- VARGRAPHIC if the arguments are graphic strings

The length attribute of the result depends on the arguments:

- If the length attribute of *expression3* is less than or equal to the length attribute of *expression2*, the length attribute of the result is the length attribute of *expression1*.
- Otherwise, the length attribute of the result is:
$$(L3 * (L1/L2)) + MOD(L1,L2)$$

where:

L1 is the length attribute of *expression1*

L2 is the length attribute of *expression2*

L3 is the length attribute of *expression3*

If the result is a character string, the length attribute of the result must not exceed 4000. If the result is a graphic string, the length attribute of the result must not exceed 2000.

The actual length of the result is the actual length of *expression1* plus the number of occurrences of *expression2* that exist in *expression1* multiplied by the actual length of *expression3* minus the actual length of *expression2*. If the actual length of the result string exceeds the maximum for the return data type, an error occurs.

If any argument can be null, the result can be null; if any argument is null, the result is the null value.

The subtype and CCSID of the result are determined as follows:

- If *expression1*, *expression2*, or *expression3* is bit data, the result is bit data and does not have an associated CCSID.
- If all three expressions are character SBCS data, the result is SBCS data and the CCSID is the CCSID for ASCII, EBCDIC, or Unicode SBCS data, depending on the encoding scheme of the SQL statement.
- If all three expressions are graphic data, the result is graphic data and the CCSID is the CCSID for ASCII, EBCDIC, or Unicode graphic data, depending on the encoding scheme of the SQL statement.
- Otherwise, the result is mixed data. The CCSID is the CCSID for ASCII, EBCDIC, or Unicode mixed data, depending on the encoding scheme of the SQL statement.

Example 1: Replace all occurrences of the character 'N' in the string 'DINING' with 'VID'. Use the CHAR function to limit the output to 10 bytes.

```
SELECT CHAR(REPLACE('DINING','N','VID'),10)
      FROM SYSIBM.SYSDUMMY1;
```

The result is the string 'DIVIDIVIDG'.

Example 2: Replace string 'ABC' in the string 'ABCXYZ' with nothing, which is the same as removing 'ABC' from the string.

```
SELECT REPLACE('ABCXYZ','ABC','')
      FROM SYSIBM.SYSDUMMY1;
```

The result is the string 'XYZ'.

Example 3: Replace string 'ABC' in the string 'ABCCABC' with 'AB'. This example illustrates that the result can still contain the string that is to be replaced (in this case, 'ABC') because all occurrences of the string to be replaced are identified prior to any replacement.

```
SELECT REPLACE('ABCCABC','ABC','AB')
      FROM SYSIBM.SYSDUMMY1;
```

The result is the string 'ABCABC'.

RIGHT

RIGHT

```
►►RIGHT(string-expression,integer)-----►►
```

The schema is SYSIBM.

The RIGHT function returns a string consisting of the specified number of rightmost *integer* characters of *string-expression*. If *string-expression* is a character or binary string, a character is a byte. If *string-expression* is a graphic string, a character is a DBCS character.

The CCSID of the result is the same as that of the *string-expression*.

string-expression

An expression that specifies the string from which the result is derived. The string must be a character, graphic, or binary string. A substring of *string-expression* is zero or more contiguous bytes of *string-expression*.

The string can contain mixed data. However, because the function operates on a strict byte-count basis, the result is not necessarily a properly formed mixed data character string.

integer

An expression that specifies the length of the result. The value must be an integer between 0 and *n*, where *n* is the length attribute of *string-expression*.

The *string-expression* is effectively padded on the right with the necessary number of blank characters so that the specified substring of *string-expression* always exists. The encoding scheme of the data determines the padding character:

- For ASCII SBCS data or ASCII mixed data, the padding character is X'20'.
- For ASCII DBCS data, the padding character depends on the CCSID; for example, for Japan (CCSID 301) the padding character is X'8140', while for simplified Chinese it is X'A1A1'.
- For EBCDIC SBCS data or EBCDIC mixed data, the padding character is X'40'.
- For EBCDIC DBCS data, the padding character is X'4040'.
- For Unicode SBCS data or UTF-8 data (Unicode mixed data), the padding character is X'20'.
- For UTF-16 data (Unicode DBCS data), the padding character is X'0020'.
- For binary data, the padding character is X'00'.

The result of the function is a varying-length string with a length attribute that is the same as the length attribute of *string-expression* and a data type that depends on the data type of *string-expression*:

- VARCHAR if *string-expression* is CHAR or VARCHAR
- CLOB if *string-expression* is CLOB
- VARGRAPHIC if *string-expression* is GRAPHIC or VARGRAPHIC
- DBCLOB if *string-expression* is DBCLOB
- BLOB if *string-expression* is BLOB

If any argument of the function can be null, the result can be null; if any argument is null, the result is the null value. The CCSID of the result is the same as that of *string-expression*.

Example 1: Assume that host variable ALPHA has a value of 'ABCDEF'. The following statement:

```
SELECT RIGHT(ALPHA,3)
      FROM SYSIBM.SYSDUMMY1;
```

returns the value 'DEF', which are the three rightmost characters in ALPHA.

Example 2: The following statement returns a zero length string.

```
SELECT RIGHT('ABCABC',0)
      FROM SYSIBM.SYSDUMMY1;
```

ROUND

ROUND

```
►►ROUND(expression1,expression2)─
```

The schema is SYSIBM.

The ROUND function returns *expression1* rounded to *expression2* places to the right of the decimal point if *expression2* is positive or to the left of the decimal point if *expression2* is zero or negative. For example, ROUND(748.58,-3) returns 700.

expression1

An expression that returns a value of any built-in numeric data type.

expression2

An expression that returns a small or large integer. The value of integer specifies the number of places to the right of the decimal point for the result if *expression2* is not negative. If *expression2* is negative, *expression1* is rounded to the sum of the absolute value of *expression2*+1 number of places to the left of the decimal point.

If the absolute value of *expression2* is larger than the number of digits to the left of the decimal point, the result is 0. (For example, ROUND(748.58,-4) returns 0.)

If *expression1* is positive, a value of 5 is rounded to the next higher positive number. If *expression1* is negative, a value of 5 is rounded to the next lower negative number.

The result of the function has the same data type and length attribute as the first argument except that the precision is increased by one if the argument is DECIMAL and the precision is less than 31. For example, an argument with a data type of DECIMAL(5,2) results in DECIMAL(06,2). An argument with a data type of DECIMAL(31,2) results in DECIMAL(031,2).

The result can be null. If any argument is null, the result is the null value.

Example 1: Calculate the number 873.726 rounded to 2, 1, 0, -1, -2, -3, and -4 decimal places respectively.

```
SELECT ROUND(873.726,2),  
       ROUND(873.726,1),  
       ROUND(873.726,0),  
       ROUND(873.726,-1),  
       ROUND(873.726,-2)  
       ROUND(873.726,-3),  
       ROUND(873.726,-4),  
  FROM SYSIBM.SYSDUMMY1;
```

This example returns the values 0873.730, 0873.700, 0874.000, 0870.000, 0900.000, 1000.000, and 0000.000.

Example 2: To demonstrate how numbers are rounded in positive and negative values, calculate the numbers 3.5, 3.1, -3.1, -3.5 rounded to 0 decimal places.

```
SELECT ROUND(3.5,0),
       ROUND(3.1,0),
       ROUND(-3.1,0),
       ROUND(-3.5,0)
  FROM SYSIBM.SYSDUMMY1;
```

This example returns the values 04.0, 03.0, -03.0, and -04.0. (Notice that in the positive value 3.5, 5 is rounded up to the next higher number while in the negative value -3.5, 5 is rounded down to the next lower negative number.)

ROUND_TIMESTAMP

ROUND_TIMESTAMP

```
►►ROUND_TIMESTAMP(timestamp-expression [,format-string])►►
```

The schema is SYSIBM.

The ROUND_TIMESTAMP scalar function returns a timestamp that is the *timestamp-expression* rounded to the unit specified by the *format-string*. If *format-string* is not specified, *timestamp-expression* is rounded to the nearest day, as if 'DD' is specified for *format-string*.

timestamp-expression must be a timestamp, or a valid string representation of a timestamp. A string representation must not be a BLOB, CLOB, or DBCLOB, and must have an actual length that is not greater than 255 bytes. For the valid formats of string representations of dates and timestamps, see "String representations of datetime values" on page 57.

Allowable values for *format-string* are listed in Table 29.

The result of the function is a TIMESTAMP. The result can be null; if any argument is null, the result is the null value.

Notes

The following format models are used with the ROUND_TIMESTAMP and TRUNC_TIMESTAMP functions.

Table 29. ROUND_TIMESTAMP and TRUNC_TIMESTAMP Format Models

Format Model	Rounding or Truncating Unit	ROUND_TIMESTAMP Example	TRUNC_TIMESTAMP Example
CC	One greater than the first two digits of a four digit year. (Rounds up on the 50th year of the century)	Input Value: 1897-12-04-12.22.22.000000 Result: 1900-01-01-00.00.00.000000	Input Value: 1897-12-04-12.22.22.000000 Result: 1800-01-01-00.00.00.000000
YYYY	Year (Rounds up on July 1st)	Input Value: 1897-12-04-12.22.22.000000 Result: 1898-01-01-00.00.00.000000	Input Value: 1897-12-04-12.22.22.000000 Result: 1897-01-01-00.00.00.000000
YY			
Y			
IYYY	ISO Year (Rounds up on July 1st)	Input Value: 1897-12-04-12.22.22.000000 Result: 1898-01-01-00.00.00.000000	Input Value: 1897-12-04-12.22.22.000000 Result: 1897-01-01-00.00.00.000000
IYY			
IY			
Q	Quarter (Rounds up on the sixteenth day of the second month of the quarter)	Input Value: 1999-06-04-12.12.30.000000 Result: 1999-07-01-00.00.00.000000	Input Value: 1999-06-04-12.12.30.000000 Result: 1999-04-01-00.00.00.000000

Table 29. ROUND_TIMESTAMP and TRUNC_TIMESTAMP Format Models (continued)

Format Model	Rounding or Truncating Unit	ROUND_TIMESTAMP Example	TRUNC_TIMESTAMP Example
MONTH	Month (Rounds up on the sixteenth day of the month)	Input Value: 1999-06-18-12.12.30.000000 Result: 1999-07-01-00.00.00.000000	Input Value: 1999-06-18-12.15.00.000000 Result: 1999-06-01-00.00.00.000000
MON			
MM			
RM			
WW	Same day of the week as the first day of the year (Rounds up on the 12th hour of the 3rd day of the week, with respect to the first day of the year)	Input Value: 2000-05-05-12.12.30.000000 Result: 2000-05-06-00.00.00.000000	Input Value: 2000-05-05-12.15.00.000000 Result: 2000-04-29-00.00.00.000000
IW	Same day of the week as the first day of the ISO year (Rounds up on the 12th hour of the 3rd day of the week, with respect to the first day of the ISO year)	Input Value: 2000-05-05-12.12.30.000000 Result: 2000-05-08-00.00.00.000000	Input Value: 2000-05-05-12.15.00.000000 Result: 2000-05-01-00.00.00.000000
W	Same day of the week as the first day of the month (Rounds up on the 12th hour of the 3rd day of the week, with respect to the first day of the month)	Input Value: 2000-05-17-12.12.30.000000 Result: 2000-05-15-00.00.00.000000	Input Value: 2000-05-17-12.15.00.000000 Result: 2000-05-15-00.00.00.000000
DDD	Day (Rounds up on the 12th hour of the day)	Input Value: 2000-05-17-12.59.59.000000 Result: 2000-05-18-00.00.00.000000	Input Value: 2000-05-17-12.59.59.000000 Result: 2000-05-17-00.00.00.000000
DD			
J			
DAY	Starting day of the week (Rounds up with respect to the 12th hour of the third day of the week. The first day of the week is always Sunday).	Input Value: 2000-05-17-12.59.59.000000 Result: 2000-05-14-00.00.00.000000	Input Value: 2000-05-17-12.59.59.000000 Result: 2000-05-14-00.00.00.000000
DY			
D			
HH	Hour (Rounds up at 30 minutes)	Input Value: 2000-05-17-23.59.59.000000 Result: 2000-05-18-00.00.00.000000	Input Value: 2000-05-17-23.59.59.000000 Result: 2000-05-17-23.00.00.000000
HH12			
HH24			
MI	Minute (Rounds up at 30 seconds)	Input Value: 2000-05-17-23.58.45.000000 Result: 2000-05-17-23.59.00.000000	Input Value: 2000-05-17-23.58.45.000000 Result: 2000-05-17-23.58.00.000000
SS	Second (Rounds up at 500000 microseconds)	Input Value: 2000-05-17-23.58.45.500000 Result: 2000-05-17-23.58.46.000000	Input Value: 2000-05-17-23.58.45.500000 Result: 2000-05-17-23.58.45.000000

Example

Set the host variable RND_TMSTMP with the input timestamp rounded to the nearest year value.

```
SET :RND_TMSTMP = ROUND_TIMESTAMP(TIMESTAMP_FMT('2000-08-14 17:30:00',
'YYYY-MM-DD HH24:MM:SS', 'YEAR');
```

ROUND_TIMESTAMP

| The value set is 2001-01-01-00.00.00.000000.

ROWID

```
►►ROWID(expression)►►
```

The schema is SYSIBM.

The ROWID function casts the input argument to a row ID.

The argument can be any type of character string, except a CLOB, with a maximum length that is no greater than 255 bytes. Although the character string can contain any value, it is recommended that the character string contain a row ID value that was previously generated by DB2 to ensure a valid row ID value is returned. For example, the function can be used to convert a ROWID value that was cast to a CHAR value back to a ROWID value.

If the actual length of *expression* is less than 40, the result is not padded. If the actual length of *expression* is greater than 40, the result is truncated. If non-blank characters are truncated, a warning is returned.

The result of the function is a row ID.

The length attribute of the result is 40. The actual length of the result is the length of *expression*.

If the argument can be null, the result can be null; if the argument is null, the result is the null value. However, a null row ID value cannot be used as the value for a row ID column in the database.

Example: Assume that table EMPLOYEE contains a ROWID column EMP_ROWID. Also assume that the table contains a row that is identified by a row ID value that is equivalent to X'F0DFD230E3C0D80D81C201AA0A280100000000000203'. Using direct row access, select the employee number for that row.

```
SELECT EMPNO  
      FROM EMPLOYEE  
     WHERE EMP_ROWID=ROWID(X'F0DFD230E3C0D80D81C201AA0A280100000000000203');
```

RTRIM

```
►►RTRIM(string-expression)►►
```

The schema is SYSIBM.

The RTRIM function removes blanks from the end of a string expression. The RTRIM function returns the same results as the STRIP function with TRAILING specified:

```
STRIP(string-expression,TRAILING)
```

string-expression must be any character string expression other than a CLOB or any graphic string expression other than a DBCLLOB. The characters that are interpreted as trailing blanks depend on the encoding scheme of the data and the data type:

- If the argument is a graphic string, then the trailing DBCS blanks are removed. If the data is encoded in ASCII, the ASCII CCSID determines the hex value that represents a double-byte blank. For example, for Japan (CCSID 301), X'8140' represents a double-byte blank, while it is X'A1A1' for Simplified Chinese. For EBCDIC-encoded data, X'4040' represents a double-byte blank. For Unicode-encoded data, X'0020' represents a double-byte blank.
- Otherwise, the trailing SBCS blanks are removed. For data that is encoded in ASCII, X'20' represents a blank. For EBCDIC-encoded data, X'40' represents a blank. For Unicode-encoded data, X'20' represents an SBCS or UTF-8 blank.

The result of the function depends on the data type of its argument:

- VARCHAR if the argument is a character string
- VARGRAPHIC if the argument is a graphic string

The length attribute of the result is the same as the length attribute of *string-expression*. The actual length of the result is the length of the expression minus the number of characters removed. If all of the characters are removed, the result is an empty string.

If the argument can be null, the result can be null; if the argument is null, the result is the null value. The CCSID of the result is the same as that of *string-expression*.

Example: Assume that host variable HELLO is defined as CHAR(9) and has a value of 'Hello '.

```
SELECT RTRIM(:HELLO)
   FROM SYSIBM.SYSDUMMY1;
```

This example removes the trailing blanks and results in 'Hello'.

SECOND

```
►►SECOND(expression)►►
```

The schema is SYSIBM.

The SECOND function returns the seconds part of its argument.

The argument must be a time, a timestamp, time duration, timestamp duration, or a valid string representation of a time or timestamp. A string representation must not be a BLOB, CLOB, or DBCLOB and must have an actual length that is not greater than 255 bytes. (For the valid formats of string representations of times and timestamps, see “String representations of datetime values” on page 57.)

The result of the function is a large integer. If the argument can be null, the result can be null; if the argument is null, the result is the null value.

The other rules depend on the data type of the argument:

If the argument is a time, timestamp, or string representation of either, the result is the seconds part of the value, which is an integer between 0 and 59.

If the argument is a time duration or timestamp duration, the result is the seconds part of the value, which is an integer between -99 and 99. A nonzero result has the same sign as the argument.

Example 1: Assume that the variable TIME_DUR is declared in a PL/I program as DECIMAL(6,0) and can therefore be interpreted as a time duration. Then, when TIME_DUR has the value 153045:

```
SECOND(:TIME_DUR)
```

returns the value 45.

Example 2: Assume that RECEIVED is a TIMESTAMP column and that one of its values is the internal equivalent of '1988-12-25-17.12.30.000000'. Then, for this value:

```
SECOND(RECEIVED)
```

returns the value 30.

SIGN

SIGN

```
►►SIGN(expression)►►
```

The schema is SYSIBM.

The SIGN function returns an indicator of the sign of the argument. The returned value is:

- 1 if the argument is less than zero
- 0 if the argument is zero
- 1 if the argument is greater than zero

The argument is an expression that returns a value of any built-in numeric data type, except DECIMAL(31,31).

The result of the function has the same data type and length attribute as the argument except that when the argument is DECIMAL, the precision is increased by one if the argument's precision and scale are equal. For example, an argument with a data type of DECIMAL(5,5) results in DECIMAL(6,5).

If the argument can be null, the result can be null; if the argument is null, the result is the null value.

Example: Assume that host variable PROFIT is a large integer with a value of 50000.

```
SELECT SIGN(:PROFIT)
   FROM SYSIBM.SYSDUMMY1;
```

This example returns the value 1.

SIN

```
►►—SIN(expression)—►►
```

The schema is SYSIBM.

The SIN function returns the sine of the argument, where the argument is an angle expressed in radians. The SIN and ASIN functions are inverse operations.

The argument is an expression that returns the value of any built-in numeric data type. If the argument is not a double precision floating-point number, it is converted to one for processing by the function.

The result of the function is a double precision floating-point number. The result can be null; if the argument is null, the result is the null value.

Example: Assume that host variable SINE is DECIMAL(2,1) with a value of 1.5. The following statement:

```
SELECT SIN(:SINE)
      FROM SYSIBM.SYSDUMMY1;
```

returns a double precision floating-point number with an approximate value of 0.99.

SINH

SINH

```
►►SINH(expression)--►►
```

The schema is SYSIBM.

The SINH function returns the hyperbolic sine of the argument, where the argument is an angle expressed in radians.

The argument is an expression that returns the value of any built-in numeric data type. If the argument is not a double precision floating-point number, it is converted to one for processing by the function.

The result of the function is a double precision floating-point number. The result can be null; if the argument is null, the result is the null value.

Example: Assume that host variable HSINE is DECIMAL(2,1) with a value of 1.5. The following statement:

```
SELECT SINH(:HSINE)
   FROM SYSIBM.SYSDUMMY1;
```

returns a double precision floating-point number with an approximate value of 2.12.

SMALLINT

```
►►—SMALLINT(—numeric-expression—)
          |—string-expression—)————►►
```

The schema is SYSIBM.

The SMALLINT function returns a small integer representation of a number or character string in the form of a numeric constant.

numeric-expression

An expression that returns a value of any built-in numeric data type.

The result of the function is a small integer. The result is the same number that would occur if the argument were assigned to a small integer column or variable. If the whole part of the argument is not within the range of small integers, an error occurs. If present, the decimal part of the argument is truncated.

string-expression

An expression that returns any type of string, (except a BLOB, CLOB, or DBCLOB), with an actual length that is not greater than 255 bytes. Leading and trailing blanks are removed from the string, and the resulting substring must conform to the rules for forming a string representation of an SQL integer constant.

The result of the function is a small integer. The result is the same number that would occur if the corresponding numeric constant were assigned to a small integer column or variable.

If the argument can be null, the result can be null; if the argument is null, the result is the null value.

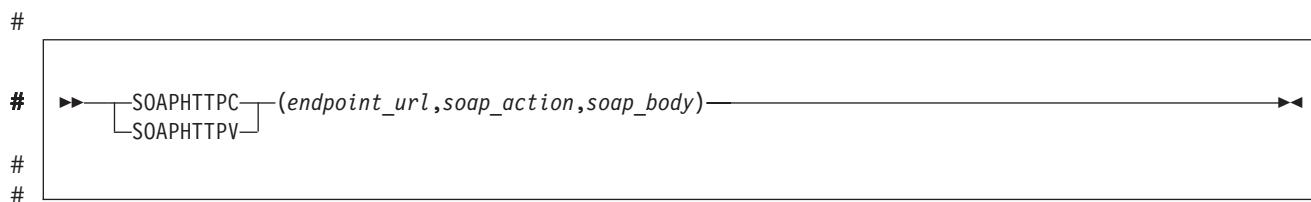
Example: Using sample table DSN8710.EMP, find the average education level (EDLEVEL) of the employees in department 'A00'. Round the result to the nearest full education level.

```
SELECT SMALLINT(AVG(EDLEVEL)+.5)
      FROM DSN8710.EMP
      WHERE DEPT = 'A00';
```

Assuming that the five employees in the department have education levels of 19, 18, 14, 18, and 14, the result is 17.

SOAPHTTPC and SOAPHTTPV

SOAPHTTPC and SOAPHTTPV



The schema is DB2XML.

The SOAPHTTPC function returns a CLOB representation of XML data that results
from a SOAP request to the web service specified by the first argument. The
SOAPHTTPV function returns a VARCHAR representation of XML data that results
from a SOAP request to the web service specified by the first argument.

endpoint_url

The URL of the web service endpoint for which DB2 is acting as a client.

soap_action

A SOAP action URI reference. This parameter is optional depending on the web
service that is specified in *endpoint_url*. If it is required, the required value is
defined in the WSDL of the specified web service.

soap_body

The name of an operation with requested namespace URI, an encoding style,
and input arguments. Can include some well-formed XML content for the SOAP
body. The specific operations and arguments for a web service are defined in
the WSDL of the specified web service.

The input for *soap_body* must be either VARCHAR(3072) or CLOB(1M) data.

If the arguments can be null, the result can be null; if all of the arguments are null,
the result is the null value.

Example 1: The following SQL statement retrieves information (as VARCHAR data)
about a web service:

```
#     SELECT DB2XML.SOAPHTTPV(  
#         'http://www.myserver.com/services/db2sample/ivt.dadx/SOAP',  
#         'http://tempuri.org/db2sample/ivt.dadx',  
#         '<testInstallation xmlns="http://tempuri.org/db2sample/ivt.dadx" />')  
#     FROM SYSIBM.SYSDUMMY1
```

Example 2: The following SQL statement inserts the results (as CLOB data) from a
request to a web service into a table:

```
#     INSERT INTO EMPLOYEE(XMLCOL)  
#     VALUES (DB2XML.SOAPHTTPC(  
#         'http://www.myserver.com/services/db2sample/list.dadx/SOAP',  
#         'http://tempuri.org/db2sample/list.dadx',  
#         '<listDepartments xmlns="http://tempuri.org/db2sample/list.dadx">  
#             <deptNo>A00</deptNo>  
#         </listDepartments>'))  
#
```

SPACE

```
►►—SPACE(expression)—————►►
```

The schema is SYSIBM.

The SPACE function returns a character string that consists of the number of SBCS blanks that the argument specifies.

The argument is an expression that results in an integer. The integer specifies the number of SBCS blanks for the result, and it must be between 0 and 32767.

The result of the function is a varying-length character string (VARCHAR) that contains SBCS data.

If *expression* is a constant, the length attribute of the result is the constant. Otherwise, the length attribute of the result is 4000. The actual length of the result is the value of *expression*. The actual length of the result must not be greater than the length attribute of the result.

If the argument can be null, the result can be null; if the argument is null, the result is the null value.

Example: The following statement returns a character string that consists of 5 blanks followed by a zero-length string.

```
SELECT SPACE(5), SPACE(0)
      FROM SYSIBM.SYSDUMMY1;
```

SQRT

SQRT

```
►►SQRT(expression1)—►►
```

The schema is SYSIBM.

The SQRT function returns the square root of the argument.

The argument can be any built-in numeric data type. If the argument is not double precision floating point, it is converted to a double precision floating-point number for processing by the function.

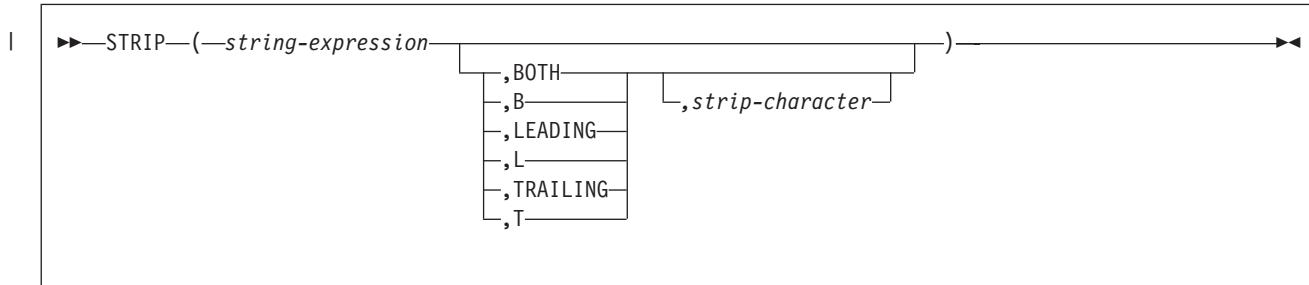
The result of the function is a double precision floating-point number. The result can be null; if the argument is null, the result is the null value.

Example: Assume that host variable SQUARE is defined as DECIMAL(2,1) and has a value of 9.0. Find the square root of SQUARE.

```
SELECT SQRT(:SQUARE)
   FROM SYSIBM.SYSDUMMY1;
```

This example returns a double precision floating-point number with an approximate value of 3.

STRIP



The schema is SYSIBM.

The STRIP function removes blanks or another specified character from the end, the beginning, or both ends of a string expression.

The first argument is an expression that returns any type of string except a BLOB, CLOB, or DBCLOB.

The second argument indicates whether characters are removed from the beginning, the end, or both ends of the string. If you do not specify a second argument, blanks are removed from both the beginning and end of the string.

The third argument is a single-character constant that indicates the SBCS or DBCS character that is to be removed. The first and third argument must have compatible string types. For more information on compatibility, see “Conversion rules for string comparison” on page 73. If the data type is not appropriate or the value contains more than one character, an error is returned.

If you do not specify the third argument, the following occurs:

- If the first argument is a graphic string, then the default strip character is a DBCS blank. The hex representation of a DBCS blank depends on the encoding scheme and CCSID of the data. For example, for data encoded in ASCII, a DBCS blank for Japan (CCSID 301) is X'8140', while for simplified Chinese it is X'A1A1'. For EBCDIC DBCS, X'4040' is interpreted as a DBCS blank. For UTF-16 (Unicode DBCS), X'0020' is interpreted as a DBCS blank.
- If the first argument is SBCS data, then the default strip character is an SBCS blank. If the data is encoded in ASCII, then X'20' represents a blank. If the data is encoded in EBCDIC, then X'40' represents a blank. If the data is encoded in UTF-8 (Unicode mixed), then X'20' represents a blank.

The result of the function is a varying-length string with the same maximum length as the length attribute of the string. The actual length of the result is the length of the expression minus the number of characters removed. If all of the characters are removed, the result is an empty, varying-length string.

The CCSID of the result is the same as that of the string. If the first argument can be null, the result can be null; if the first argument is null, the result is the null value.

Example 1: Assume that host variable HELLO is defined as CHAR(9) and has a value of 'Hello':

```
STRIP(:HELLO)
```

STRIP

This example results in 'Hello'. If there had been any ending blanks, they would have been removed, too.

Rewrite the example so that no beginning blanks are removed.

```
STRIP(:HELLO,TRAILING)
```

This results in 'Hello'.

Example 2: Assume that host variable BALANCE is defined as CHAR(9) and has a value of '000345.50':

```
STRIP(:BALANCE,L,'0')
```

This example results in '345.50'.

SUBSTR

```
►►—SUBSTR(string-expression,start  
          └, length┘)————►►
```

The schema is SYSIBM.

The SUBSTR function returns a substring of a string.

string-expression

An expression that specifies the string from which the result is derived. The string must be a character, graphic, or binary string. If *string-expression* is a character string, the result of the function is a character string. If it is a graphic string, the result of the function is a graphic string. If it is a binary string, the result of the function is a binary string.

A substring of *string-expression* is zero or more contiguous characters of *string*. If *string-expression* is a graphic string, a character is a DBCS character. If *string-expression* is a character string or a binary string, a character is a byte. The SUBSTR function accepts mixed data strings. However, because SUBSTR operates on a strict byte-count basis, the result will not necessarily be a properly formed mixed data string.

start

An expression that specifies the position within *string-expression* to be the first character of the result. The value of integer must be between 1 and the length attribute of *string-expression*. (The length attribute of a varying-length string is its maximum length.) A value of 1 indicates that the first character of the substring is the first character of *string-expression*.

length

An expression that specifies the length of the resulting substring. The length must be an integer in the range 0 to *n*, where *n* is equal to L-S+1 (L is the length attribute of *string-expression* and S is the value of *start*). The specified length must not be the integer constant 0.

If *string-expression* is a varying-length string and if *length* is explicitly specified, *string-expression* is effectively padded on the right with the necessary number of characters so that the specified substring of *string-expression* always exists. Hexadecimal zeroes are used as the padding character when *string-expression* is BLOB data. Otherwise, a blank that is appropriate for *string-expression*²³ is used as the padding character.

If *string-expression* is a fixed-length string, omission of *length* is an implicit specification of LENGTH(*string-expression*) - *start* + 1, which is the number of characters from the character specified by *start* to the last character of *string-expression*.

If *string-expression* is a varying-length string, omission of *length* is an implicit specification of zero or LENGTH(*string-expression*) - *start* + 1, whichever is greater.

23. The appropriate blank is defined by the data type and sub-type (if necessary) of *string-expression*

SUBSTR

```
# If length is explicitly specified by an integer constant that is 255 or less, and  
# string-expression is not a LOB, the result is a fixed-length string with a length  
# attribute of length.  
#  
# If length is not explicitly specified, but string-expression is a fixed-length string  
# and start is an integer constant, the result is a fixed-length string with a length  
# attribute equal to LENGTH(string-expression) - start +1.  
#  
# In all other cases, the result is a varying-length string. If length is explicitly  
# specified by an integer constant, the length attribute of the result is length;  
# otherwise, the length attribute of the result is the same as the length attribute of  
# string-expression.
```

The result is subject to the restrictions that apply to long strings if its maximum length is greater than 255. These restrictions also apply if it is a graphic string whose maximum length is greater than 127.

If any argument of SUBSTR can be null, the result can be null. If any argument is null, the result is the null value. The CCSID of the result is the CCSID of *string-expression*.

Example 1: FIRSTNAME is a VARCHAR(12) column in sample table DSN8710.EMP. One of its values is the 5-character string 'MAUDE'. When FIRSTNAME has this value:

Function ...	Returns ...
SUBSTR(FIRSTNAME,2,3)	'AUD'
SUBSTR(FIRSTNAME,2)	'AUDE'
SUBSTR(FIRSTNAME,2,6)	'AUDE' followed by two blanks
SUBSTR(FIRSTNAME,6)	a zero-length string
SUBSTR(FIRSTNAME,6,4)	four blanks

Example 2: Sample table DSN8710.PROJ contains column PROJNAME, which is defined as VARCHAR(24). Select all rows from that table for which the string in PROJNAME begins with 'W L PROGRAM'.

```
SELECT * FROM DSN8710.PROJ  
WHERE SUBSTR(PROJNAME,1,12) = 'W L PROGRAM ';
```

Assume that the table has only the rows that were supplied by DB2. Then the predicate is true for just one row, for which PROJNAME has the value 'W L PROGRAM DESIGN'. The predicate is not true for the row in which PROJNAME has the value 'W L PROGRAMMING' because, in the predicate's string constant, 'PROGRAM' is followed by a blank.

Example 3: Assume that a LOB locator named my_loc represents a LOB value that has a length of 1 gigabyte. Assign the first 50 bytes of the LOB value to host variable PORTION.

```
SET :PORTION = SUBSTR(:my_loc,1,50);
```

Example 4: Assume that host variable RESUME has a CLOB data type and holds an employee's resume. This example shows some of the statements that find the section of department information in the resume and assign it to host variable DeptBuf. First, the POSSTR function is used to find the beginning and ending location of the department information. Within the resume, the department information starts with the string 'Department Information Section' and ends immediately before the string 'Education Section'. Then, using these beginning and ending positions, the SUBSTR function assigns the information to the host variable.

```
SET :DInfoBegPos = POSSTR(:RESUME, 'Department Information Section');
SET :DInfoEnPos = POSSTR(:RESUME, 'Education Section');
SET :DeptBuf = SUBSTR(:RESUME, :DInfoBegPos, :DInfoEnPos - :DInfoBegPos);
```

TAN

TAN

```
►►TAN(expression)►►
```

The schema is SYSIBM.

The TAN function returns the tangent of the argument, where the argument is an angle expressed in radians. The TAN and ATAN functions are inverse operations.

The argument is an expression that returns the value of any built-in numeric data type. If the argument is not a double precision floating-point number, it is converted to one for processing by the function.

The result of the function is a double precision floating-point number. The result can be null; if the argument is null, the result is the null value.

Example: Assume that host variable TANGENT is DECIMAL(2,1) with a value of 1.5. The following statement:

```
SELECT TAN(:TANGENT)
      FROM SYSIBM.SYSDUMMY1;
```

returns a double precision floating-point number with an approximate value of 14.10.

TANH

```
►►—TANH(expression)—————►►
```

The schema is SYSIBM.

The TANH function returns the hyperbolic tangent of the argument, where the argument is an angle expressed in radians. The TANH and ATANH functions are inverse operations.

The argument is an expression that returns the value of any built-in numeric data type. If the argument is not a double precision floating-point number, it is converted to one for processing by the function.

The result of the function is a double precision floating-point number. The result can be null; if the argument is null, the result is the null value.

Example: Assume that host variable HTANGENT is DECIMAL(2,1) with a value of 1.5. The following statement:

```
SELECT TANH(:HTANGENT)
      FROM SYSIBM.SYSDUMMY1;
```

returns a double precision floating-point number with an approximate value of 0.90.

TIME

TIME

```
►►TIME(expression)—►►
```

The schema is SYSIBM.

The TIME function returns a time derived from its argument.

The argument must be a time, a timestamp, or a valid string representation of a time or timestamp. A string representation must not be a BLOB, CLOB, or DBCLOB and must have an actual length that is not greater than 255 bytes. (For the valid formats of string representations of times and timestamps, see “String representations of datetime values” on page 57.)

If the argument is a string, the result is the time or time part of the timestamp represented by the string. If the CCSID of the string is not the same as the corresponding default CCSID at the server, the string is first converted to that CCSID. If the argument can be null, the result can be null; if the argument is null, the result is the null value.

The other rules depend on the data type of the argument:

If the argument is a time, the result is that time.

If the argument is a timestamp, the result is the time part of the timestamp.

If the argument is a character string, the result is the time or time part of the timestamp represented by the character string. If the CCSID of the string is not the same as the corresponding default CCSID at the server, the string is first converted to that CCSID.

Example: Assume that a table named CLASSES contains one row for each scheduled class. Assume also that the class starting times are in the TIME column named STARTTM. Using these assumptions, select those rows in CLASSES that represent classes that start at 1:30 P.M.

```
SELECT *
  FROM CLASSES
 WHERE TIME(STARTTM) = '13:30:00';
```

TIMESTAMP

```
►►—TIMESTAMP(expression
    [ ,expression ] )—►►
```

The schema is SYSIBM.

The TIMESTAMP function returns a timestamp derived from its argument or arguments.

The rules for the arguments depend on whether the second argument is specified.

If only one argument is specified, it must be a timestamp, a valid string representation of a timestamp, a character string of length 8, or a string of length 14. The argument cannot be a BLOB, CLOB, or DBCLOB. (String representations for a timestamp are described in “String representations of datetime values” on page 57.)

A string of length 8 is assumed to be a System/390 Store Clock value.

A string of length 14 must be a string of digits that represents a valid date and time in the form *yyyymmddhhmmss*, where *yyyy* is the year, *mm* is the month, *dd* is the day, *hh* is the hour, *mm* is the minute, and *ss* is the seconds.

If both arguments are specified, the first argument must be a date or a valid string representation of a date and the second argument must be a time or a valid string representation of a time. Neither argument can be a BLOB, CLOB, or DBCLOB. (Table 4 on page 58 and Table 5 on page 59 list the valid formats for string representations for dates and times.)

The result of the function is a timestamp. If either argument can be null, the result can be null; if either argument is null, the result is the null value.

The other rules depend on whether the second argument is specified:

If both arguments are specified, the result is a timestamp with the date specified by the first argument and the time specified by the second argument. The microsecond part of the timestamp is zero.

If only one argument is specified and it is a timestamp, the result is that timestamp.

If only one argument is specified and it is a string, the result is the timestamp represented by that string. The timestamp represented by a string of length 14 has a microsecond part of zero. The interpretation of a string as a Store Clock value will yield a timestamp with a year between 1900 to 2042 as described in *ESA/390 Principles of Operation*.

If an argument is a string with a CCSID that is not the same as the corresponding default CCSID at the server, the string is first converted to that CCSID.

Example: Assume that table TABLEX contains a DATE column named DATECOL and a TIME column named TIMECOL. For some row in the table, assume that DATECOL represents 25 December 1988 and TIMECOL represents 17 hours, 12 minutes, and 30 seconds after midnight. Then, for this row:

TIMESTAMP(DATECOL, TIMECOL)

returns the value '1988-12-25-17.12.30.000000'.

TIMESTAMP_FORMAT

TIMESTAMP_FORMAT

```
(1)
►►►TIMESTAMP_FORMAT(string-expression,format-string)►►►
```

Notes:

- 1 TO_DATE can be specified as synonym for TIMESTAMP_FORMAT.

The schema is SYSIBM.

The TIMESTAMP_FORMAT function returns a timestamp.

string-expression

An expression that returns any type of string (except a BLOB, CLOB, or DBCLOB) with a maximum length that is not greater than 255 bytes. Leading and trailing blanks are removed from the string, and the resulting substring is interpreted as a timestamp using the format specified by *format-string*.

format-string

An expression that returns a character string constant with a maximum length that is not greater than 255 bytes. *format-string* contains a template of how *string-expression* is to be interpreted as a timestamp value. Leading and trailing blanks are removed from the string, and the resulting substring must be a valid template for a timestamp. The only valid format for the function is:

'YYYY-MM-DD HH24:MI:SS'

where:

YYYY 4-digit year

MM Month (01-12, January = 01)

DD Day of month (01-31)

HH24 Hour of day (00–24, when the value is 24, the minutes and seconds must be 0).

MI Minutes (00–59)

SS Seconds (00–59)

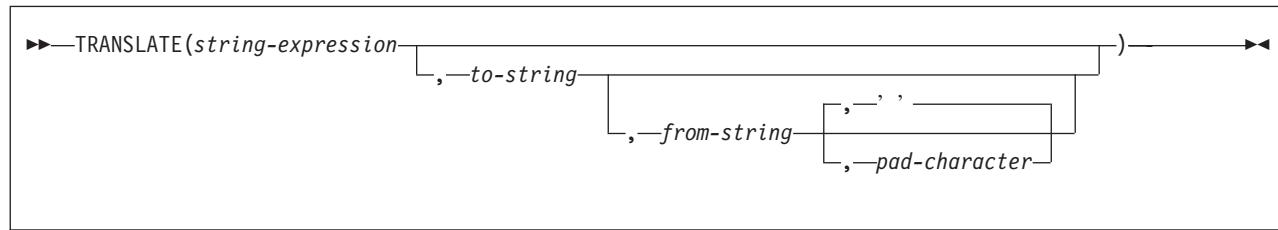
The result of the function is a timestamp.

If the argument can be null, the result can be null; if the argument is null, the result is the null value.

Example: Set the character variable TVAR to the value of CREATEDTS from SYSIBM.SYSDATABASE if it is equal to one second before the beginning of the year 2000 ('1999-12-31 23:59:59'). The character string should be interpreted in the only format that can be specified for the function.

```
SELECT VARCHAR_FORMAT(CREATEDTS,'YYYY-MM-DD HH24:MI:SS')
  INTO :TVAR
  FROM SYSIBM.SYSDATABASE;
 WHERE CREATEDTS =
  TIMESTAMP_FORMAT('1999-12-31 23:59:59','YYYY-MM-DD HH24:MI:SS');
```

TRANSLATE



The schema is SYSIBM.

The TRANSLATE function translates one or more characters of *string-expression*.

string-expression

An expression that specifies the string to be translated. The string must be a character or graphic string. A character string argument must not be a CLOB and must have an actual length that is not greater than 255. A graphic string argument must not be a DBCLOB and must have an actual length that is not greater than 127.

to-string

A string that specifies the characters to which certain characters in *string-expression* are to be translated. This string is sometimes called the *output translation table*. The string must be a character or graphic string. A character string argument must not be a CLOB and must have an actual length that is not greater than 255. A graphic string argument must not be a DBCLOB and must have an actual length that is not greater than 127.

If the length of *to-string* is less than the length of *from-string*, *to-string* is padded to the length of *from-string* with the *pad-character* or a blank. If the length of *to-string* is greater than *from-string*, the extra characters in *to-string* are ignored without warning.

from-string

A string that specifies the characters that if found in *string-expression* are to be translated. This string is sometimes called the *input translation table*. When a character in *from-string* is found, the character in *string-expression* is translated to the character in *to-string* that is in the corresponding position of the character in *from-string*.

from-string must be a character or a graphic string. A character string argument must not be a CLOB and must have an actual length that is not greater than 255. A graphic string argument must not be a DBCLOB and must have an actual length that is not greater than 127.

If *from-string* contains duplicate characters, the first occurrence of the character is used, and no warning is issued. The default value for *from-string* is a string that starts with the character X'00' and ends with the character X'FF' (decimal 255).

pad-character

A string that specifies the character with which to pad *to-string* if its length is less than *from-string*. The string must be a character string (except a CLOB) or a graphic string (except a DBCLOB) with a length of 1. A length of 1 is one single byte for character strings and one double byte string for graphic strings. The default is a blank that is appropriate for *string-expression*.

|
|
|

TRANSLATE

All of the arguments must have the same string type. They must all be character strings or all be graphic strings.

If *string-expression* is the only argument that is specified, the characters of its value are translated to uppercase based on the LC_CTYPE locale in effect for the statement, which is determined by special register CURRENT LOCALE LC_CTYPE. For example, a-z are translated to A-Z, and characters with diacritical marks are translated to their uppercase equivalent, if any. (For a description of the uppercase tables that are used for this translation, see *IBM National Language Support Reference Manual Volume 2*.)

If the LC_CTYPE locale is blank when the function is executed, the result of the function depends on the data type of *string-expression*.

- For ASCII and EBCDIC, if *string-expression* specifies a graphic string expression, then an error occurs. For a character string expression, characters a-z are translated to A-Z and characters with diacritical marks are not translated. If the string contains mixed or DBCS characters, fullwidth Latin capital letters A-Z are converted to fullwidth Latin small letters a-z.
- For Unicode, *string-expression* can be either a character string expression or a graphic string expression, and LOCALE LC_CTYPE must be blank (no locale specified). The characters a-z are translated to A-Z and all other characters, including characters with diacritic marks, are left unchanged. If LOCALE LC_CTYPE is not blank, an error occurs. Fullwidth Latin capital letters A-Z are converted to Fullwidth Latin small letters a-z.

If more than one argument is specified, the result string is built character-by-character from *string-expression* with each character in *from-string* being translated to the corresponding character in *to-string*. For each character in *string-expression*, the *from-string* is searched for the same character. If the character is found to be the *n*th character in *from-string*, the resulting string will contain the *n*th character from *to-string*. If *to-string* is less than *n* characters long, the resulting string will contain the *pad-character*. If the character is not found in *from-string*, it is moved to the result string without being translated.

The string can contain mixed data. If only one argument is specified, the UPPER function is performed on the argument, and the rules for operating on mixed data in the UPPER function are observed. Fullwidth Latin small letters a-z are converted to fullwidth Latin capital letters A-Z. Otherwise, the function operates on a strict byte-count basis, and the result is not necessarily a properly formed mixed data character string.

The length attribute, data type, subtype, and CCSID of the result are the same as *string-expression*. If the first argument can be null, the result can be null. If the argument is null, the result is the null value.

Example 1: Return the string 'abcdef' in uppercase characters. Assume that the locale in effect is blank.

```
SELECT TRANSLATE ('abcdef')
   FROM SYSIBM.SYSDUMMY1
```

The result is the value 'ABCDEF'.

Example 2: Assume that host variable SITE has a data type of VARCHAR(30) and contains 'Hanauma Bay'.

```
SELECT TRANSLATE (:SITE)
  FROM SYSIBM.SYSDUMMY1
```

Returns the value 'HANAUMA BAY'. The result is all uppercase characters because only one argument is specified.

```
SELECT TRANSLATE (:SITE, 'j', 'B')
  FROM SYSIBM.SYSDUMMY1
```

Returns the value 'Hanauma jay'.

```
SELECT TRANSLATE (:SITE, 'ei', 'aa')
  FROM SYSIBM.SYSDUMMY1
```

Returns the value 'Heneume Bey'.

```
SELECT TRANSLATE (:SITE, 'bA', 'Bay', '%')
  FROM SYSIBM.SYSDUMMY1
```

Returns the value 'HAnAumA bA%'.

```
SELECT TRANSLATE (:SITE, 'r', 'Bu')
  FROM SYSIBM.SYSDUMMY1
```

Returns the value 'Hana ma ray'.

Example 3: Assume that host variable SITE has a data type of VARCHAR(30) and contains 'Pivabiska Lake Place'.

```
SELECT TRANSLATE (:SITE, '$$', 'L1')
  FROM SYSIBM.SYSDUMMY1
```

Returns the value 'Pivabiska \$ake P\$ace'.

```
SELECT TRANSLATE (:SITE, 'pLA', 'Place', '.')
  FROM SYSIBM.SYSDUMMY1
```

Returns the value 'pivAbiskA LAk. pLA..'.

TRUNCATE or TRUNC

TRUNCATE or TRUNC

►► [TRUNCATE] (*expression1*,*expression2*) —————►►

The schema is SYSIBM.

The TRUNCATE or TRUNC function returns *expression1* truncated to *expression2* places to the right of the decimal point. If *expression2* is negative, *expression1* is truncated to the absolute value of *expression2* places to the left of the decimal point. If the absolute value of *expression2* is larger than the number of digits to the left of the decimal point, the result is 0. For example, TRUNCATE(748.58,-4) returns 0.

expression1

An expression that results in a value of any built-in numeric data type.

expression2

An expression that results in a small or large integer.

The result of the function has the same data type and length attribute as the first argument. The result can be null. If any argument is null, the result is the null value.

Example 1: Using sample employee table DSN8710.EMP, calculate the average monthly salary for the highest paid employee. Truncate the result to two places to the right of the decimal point.

```
SELECT TRUNCATE(MAX(SALARY)/12,2)  
  FROM DSN8710.EMP;
```

Because the highest paid employee in the sample employee table earns \$52750.00 per year, the example returns the value 4395.83.

Example 2: Return the number 873.726 truncated to 2, 1, 0, -1, and -2 decimal places respectively.

```
SELECT TRUNC(873.726,2),  
       TRUNC(873.726,1),  
       TRUNC(873.726,0),  
       TRUNC(873.726,-1),  
       TRUNC(873.726,-2)  
  FROM TABLEX  
 WHERE INTCOL = 1234;
```

This example returns the values 873.720, 873.700, 873.000, 870.000, and 800.000.

TRUNC_TIMESTAMP

```
►►—TRUNC_TIMESTAMP(timestamp-expression [ ,format-string ])—►►
```

The schema is SYSIBM.

The TRUNC_TIMESTAMP scalar function returns a timestamp that is the *timestamp-expression* truncated to the unit specified by the *format-string*. If *format-string* is not specified, *timestamp-expression* is truncated to the nearest day, as if 'DD' was specified for *format-string*.

timestamp-expression must be a timestamp, or a valid string representation of a timestamp. A string representation must not be a BLOB, CLOB, or DBCLOB, and must have an actual length that is not greater than 255 bytes. For the valid formats of string representations of dates and timestamps, see "String representations of datetime values" on page 57.

Allowable values for *format-string* are listed in Table 29 on page 294.

The result of the function is a TIMESTAMP. The result can be null; if any argument is null, the result is the null value.

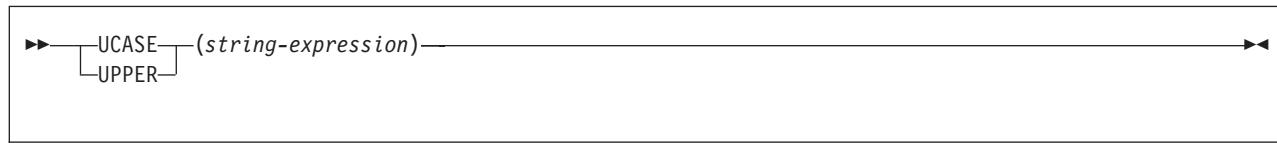
Example: Set the host variable TRNK_TMSTMP with the current year rounded to the nearest year value.

```
SET :TRNK_TMSTMP = TRUNC_TIMESTAMP('2000-03-14-17.30.00', 'YEAR');
```

The host variable TRNK_TMSTMP is set with the value 2000-01-01-00.00.00.000000.

UCASE or UPPER

UCASE or UPPER



The schema is SYSIBM.

The UCASE or UPPER function returns a string in which all the characters have been converted to uppercase characters.

string-expression

An expression that specifies the string to be converted. The string must be a character or graphic string. A character string argument must not be a CLOB and must have an actual length that is not greater than 255. A graphic string argument must not be a DBCLOB and must have an actual length that is not greater than 127.

The alphabetic characters of the argument are translated to uppercase characters based on the value of the LC_CTYPE locale in effect for the statement. For example, characters a-z are translated to A-Z, and characters with diacritical marks are translated to their uppercase equivalent, if any. The locale is determined by special register CURRENT LOCALE LC_CTYPE. For information about the special register, see "CURRENT LOCALE LC_CTYPE" on page 87.

If the LC_CTYPE locale is blank when the function is executed, the result of the function depends on the data type of *string-expression*.

- For ASCII and EBCDIC, if *string-expression* specifies a graphic string expression, then an error occurs. For a character string expression, characters a-z are translated to A-Z and characters with diacritical marks are not translated. If the string contains mixed or DBCS characters, fullwidth Latin small letters a-z are converted to fullwidth Latin capital letters A-Z.
- For Unicode, *string-expression* can be either a character string expression or a graphic string expression. The characters a-z are translated to A-Z and all other characters, including characters with diacritic marks, are left unchanged. If LOCALE LC_CTYPE is not blank, an error occurs. Fullwidth Latin small letters a-z are converted to fullwidth Latin capital letters A-Z.

The length attribute, data type, subtype, and CCSID of the result are the same as the expression. If the argument can be null, the result can be null; if the argument is null, the result is the null value.

Example: Return the string 'abcdef' in uppercase characters. Assume that the locale in effect is blank.

```
SELECT TRANSLATE ('abcdef')
   FROM SYSIBM.SYSDUMMY1
```

The result is the value 'ABCDEF'.

VARCHAR

Character to Varchar:

►►—VARCHAR(*character-expression*)
 └,—*integer*┘

Graphic to Varchar:

►►—VARCHAR(*graphic-expression*)
 └,—*integer*┘

Datetime to Varchar:

►►—VARCHAR(*datetime-expression*)

Integer to Varchar:

►►—VARCHAR(*integer-expression*)

Decimal to Varchar:

►►—VARCHAR(*decimal-expression*)
 └,—*decimal-character*┘

Floating-point to Varchar:

►►—VARCHAR(*floating-point-expression*)

Row ID to Varchar:

►►—VARCHAR(*row-ID-expression*)

The schema is SYSIBM.

The VARCHAR function returns a varying-length character string representation of a character string, graphic string, datetime value, integer number, decimal number, floating-point number, or row ID value.

The result of the function is a varying-length character string (VARCHAR). If the first argument can be null, the result can be null; if the first argument is null, the result is the null value.

Character to Varchar

VARCHAR

character-expression

An expression that returns a value that is CHAR, VARCHAR, or CLOB data type.

integer

The length attribute for the resulting varying-length character string. The value must be between 1 and 32767. If the length is not specified, the length of the result is the same as the length of *character-expression*. If the first argument is mixed data, the second argument cannot be less than 4.

If the second argument is not specified and if the *character-expression* is an empty string constant, the length attribute of the result is 1 and the result is an empty string. Otherwise, the length attribute of the result is the same as the length attribute of the first argument.

The actual length of the result is the minimum of the length attribute of the result and the actual length of *character-expression*. If the length of *character-expression* is greater than the length attribute of the result, the result is truncated. Unless all the truncated characters are blanks appropriate for *character-expression*, a warning is returned.

If *character-expression* is bit data, the result is bit data. Otherwise, the CCSID of the result is the same as the CCSID of *character-expression*.

Graphic to Varchar

graphic-expression

An expression that returns a value that is a GRAPHIC, VARGRAPHIC, or DBCLOB data type.

integer

The length attribute for the resulting varying-length character string. The value must be between 1 and 32767.

If second argument is not specified, the length attribute of the result is determined as follows (where *n* is the length attribute of the first argument):

- If the *graphic-expression* is the empty graphic string constant, the length attribute of the result is 1.
- If the result is SBCS data, the result length is *n*.
- If the result is mixed data, the result length is $(3 * \text{length}(\text{string-expression}))$.

The actual length of the result is the minimum of the length attribute of the result and the actual length of *graphic-expression*. If the length of the character expression is greater than the length attribute of the result, the result is truncated. Unless all the truncated characters were blanks appropriate for *graphic-expression*, a warning is returned .

Datetime to Varchar

datetime-expression

An expression whose value has one of the following three data types:

- date** The result is a varying-length character string representation of the date in the format that is specified by the DATE precompiler option, if one is provided, or else field DATE FORMAT on installation panel DSNTIP4 specifies the format. If the format is to be LOCAL, field LOCAL DATE LENGTH on installation panel DSNTIP4 specifies the length of the result. Otherwise, the length attribute and actual length of the result is 10.

LOCAL denotes the local format at the DB2 that executes the SQL statement.

time The result is a varying-length character string representation of the time in the format specified by the TIME precompiler option, if one is provided, or else field TIME FORMAT on installation panel DSNTIP4 specifies the format. If the format is to be LOCAL, the field LOCAL TIME LENGTH on installation panel DSNTIP4 specifies the length of the result. Otherwise, the length attribute and actual length of the result is 8.

LOCAL denotes the local format at the DB2 that executes the SQL statement.

timestamp

The result is the varying-length character string representation of the timestamp. The length attribute and actual length of the result is 26.

The CCSID of the result is the SBCS CCSID of the appropriate encoding scheme.

Integer to Varchar

integer-expression

An expression that returns a value with a small or large integer data type.

The result is a varying-length character string representation (VARCHAR) of the argument in the form of an SQL integer constant.

The length attribute of the result depends on whether the argument is a small or large integer as follows:

- If the argument is a small integer, the length attribute of the result is 6 bytes.
- If the argument is a large integer, the length attribute of the result is 11 bytes.

The actual length of the result is the smallest number of characters that can be used to represent the value of the argument. Leading zeroes are not included. If the argument is negative, the first character of the result is a minus sign. Otherwise, the first character is a digit.

The CCSID of the result is the SBCS CCSID of the appropriate encoding scheme.

Decimal to Varchar

decimal-expression

An expression that returns a value that is a decimal data type. To change the precision and scale of the expression's value, apply the DECIMAL function to the expression before applying the VARCHAR function.

decimal-character

Specifies a single-byte character constant (CHAR or VARCHAR) that represents the decimal point in the resulting character string. The character cannot be a digit, a plus sign (+), a minus sign (-), or a blank. The default is the period (.) or comma (,). For information on what factors govern the choice, see "Options affecting SQL" on page 146.

VARCHAR

The result is a varying-length character string representation (VARCHAR) of the argument in the form of an SQL decimal constant. The result includes a character that represents the decimal point and p digits where p is the precision of *decimal-expression*.

The length attribute of the result is $2+p$ where p is the precision of *decimal-expression*. The actual length of the result is the smallest number of characters that can be used to represent the result, except that trailing zeros are included. Leading zeros are not included. If the argument is negative, the result begins with a minus sign. Otherwise, the result begins with a digit.

The CCSID of the result is the SBCS CCSID of the appropriate encoding scheme.

Floating-Point to Varchar

floating-point-expression

An expression that returns a value that is a floating-point data type.

The result is a varying-length character string representation (VARCHAR) of the argument in the form of an SQL floating-point constant.

The length attribute of the result is 24. The actual length of the result is the smallest number of characters that can represent the value of the argument such that the mantissa consists of a single digit other than zero followed by a period and a sequence of digits. If the argument is negative, the first character of the result is a minus sign; otherwise, the first character is a digit. If the argument is zero, the result is 0E0.

The CCSID of the result is the SBCS CCSID of the appropriate encoding scheme.

Row ID to Varchar

row-ID-expression

An expression whose value must be of a row ID data type.

The result is a varying-length character string representation (VARCHAR) of the argument. It is bit data and does not have an associated CCSID.

The length attribute of the result is 40. The actual length of the result is the length of *row-ID-expression*.

Example: Assume that host variable JOB_DESC is defined as VARCHAR(8). Using sample table DSN8710.EMP, set JOB_DESC to the varying-length string equivalent of the job description (column JOB defined as CHAR(8)) for the employee with the last name of 'QUINTANA'.

```
SELECT VARCHAR(JOB)
  INTO :JOB_DESC
  FROM DSN8710.EMP
 WHERE LASTNAME = 'QUINTANA';
```

VARCHAR_FORMAT

(1)

```
►►—VARCHAR_FORMAT(timestamp-expression,format-string)—————►►
```

Notes:

- 1 TO_CHAR can be specified as a synonym for VARCHAR_FORMAT.

The schema is SYSIBM.

The VARCHAR_FORMAT function returns a character representation of a timestamp in the format indicated by *format-string*.

timestamp-expression

An expression that returns a timestamp.

format-string

An expression that returns a character string constant with a maximum length that is not greater than 255 bytes. *format-string* contains a template of how *timestamp-expression* is to be formatted. Leading and trailing blanks are removed from the string, and the resulting substring must conform to the rules for formatting a timestamp. The only valid format that can be specified for the function is:

'YYYY-MM-DD HH24:MI:SS'

where:

YYYY 4-digit year

MM Month (01-12, January = 01)

DD Day of month (01-31)

HH24 Hour of day (00–24, when the value is 24, the minutes and seconds must be 0).

MI Minutes (00–59)

SS Seconds (00–59)

The result is the varying-length character string that contains the argument in the format specified by *format-string*. *format-string* also determines the length attribute and actual length of the result.

If the argument can be null, the result can be null; if the argument is null, the result is the null value.

Example: Set the character variable TVAR to the timestamp value of CREATEDTS from SYSIBM.SYSDATABASE, using the character string format supported by the function to specify the format of the value for TVAR.

```
SELECT VARCHAR_FORMAT(CREATEDTS,'YYYY-MM-DD HH24:MI:SS')
  INTO :TVAR
  FROM SYSIBM.SYSDATABASE;
```

VARGRAPHIC

VARGRAPHIC

Character to Vargraphic:

```
►►—VARGRAPHIC(character-expression—  
                  └,—integer—)—————►►
```

Graphic to Vargraphic:

```
►►—VARGRAPHIC(graphic-expression—  
                  └,—integer—)—————►►
```

The schema is SYSIBM.

The VARGRAPHIC function returns a varying-length graphic string representation of a character string value, with the single-byte characters converted to double-byte characters, or a graphic string value.

The result of the function is a varying-length graphic string (VARGRAPHIC). If the length attribute of the result is greater than 127 double-byte characters, the result is a long string and subject to the restrictions that apply to long strings.

If the first argument can be null, the result can be null; if the first argument is null, the result is the null value.

The length attribute and actual length of the result are measured in double-byte characters because the result is a graphic string.

Character to Vargraphic

character-expression

An expression whose value must be an EBCDIC-encoded or Unicode-encoded character string. The VARGRAPHIC function is not allowed for ASCII data. The argument does not need to be mixed data, but any occurrences of X'0E' and X'0F' in the string must conform to the rules for EBCDIC mixed data. (See “Character strings” on page 49 for these rules.)

integer

The length attribute of the resulting varying-length graphic string. The value of *integer* must be between 1 and 16352.

If the second argument is not specified and if the *character-expression* is an empty string constant or has a value X'0E0F', the length attribute of the result is 1 and the result is an empty string. Otherwise, the length attribute of the result is the same as the length attribute of the first argument.

The actual length of the result is the minimum of the length attribute of the result and the actual length of *character-expression*. If the length of *character-expression*, as measured in single-byte characters, is greater than the specified length of the result, as measured in double-byte characters, the result is truncated. Unless all the truncated characters are blanks appropriate for *character-expression*, a warning is returned.

| For EBCDIC input data:

Each character of *character-expression* determines a character of the result. The argument might need to be converted to the native form of mixed data before the result is derived. Let M denote the system EBCDIC CCSID for mixed data. The argument is not converted if any of the following conditions is true:

- The argument is mixed data and its CCSID is M.
- The argument is SBCS data and its CCSID is the same as the system CCSID for SBCS data. In this case, the operation proceeds as if the CCSID of the argument is M.
- The argument is BIT data. In this case, the operation proceeds as if the CCSID of the argument is M.

Otherwise, the argument is a new string S derived by converting the characters to the coded character set identified by M. If there is no system CCSID for mixed data, conversion is to the coded character set that the system CCSID for SBCS data identifies.

The result is derived from S using the following steps:

- Each shift character (X'OE' or X'OF') is removed.
- Each double-byte character remains as is.
- Each single-byte character is replaced by a double-byte character.

The replacement for a single-byte character is the equivalent DBCS character if an equivalent exists. Otherwise, the replacement is X'FEFE'. The existence of an equivalent character depends on M. If there is no system CCSID for mixed data, the DBCS equivalent of X'xx' for EBCDIC is X'42xx', except for X'40', whose DBCS equivalent is X'4040'.

| For Unicode input data:

| Each character of *character-expression* determines a character of the result. The argument might need to be converted to the native form of mixed data before the result is derived. Let M denote the system CCSID for mixed data. The argument is not converted if any of the following conditions is true:

- The argument is mixed data, and its CCSID is M.
- The argument is SBCS data, and its CCSID is the same as the system CCSID for SBCS data. In this case, the operation proceeds as if the CCSID of the argument is M.
- The argument cannot be BIT data.

| Otherwise, the argument is a new string S derived by converting the characters to the coded character set identified by M.

| The result is derived from S using the following steps:

- Each non-surrogate character is replaced by a Unicode double-byte character (a UTF-16 code point).
- Each surrogate character is replaced by a pair of Unicode double-byte characters (a pair of UTF-16 code points).

| The replacement for a single-byte character is the Unicode equivalent character if an equivalent exists. Otherwise, the replacement is X'FFFD'.

Graphic to Vargraphic

VARGRAPHIC

graphic-expression

An expression that returns a value that is an EBCDIC-encoded or Unicode-encoded graphic string.

integer

The length attribute for the resulting varying-length graphic string. The value must be an integer between 1 and 16352.

If the second argument is not specified and if the *graphic-expression* is an empty string constant, the length attribute of the result is 1 and the result is an empty string. Otherwise, the length attribute of the result is the same as the length attribute of the first argument.

If the first argument is an empty string, the result is an empty string.

The actual length of the result depends on the number of characters in *graphic-expression*. If the length of *graphic-expression* is greater than the length specified, the result is truncated. Unless all of the truncated characters are double-byte blanks, a warning is returned.

The CCSID of the result is the same as the CCSID of *graphic-expression*.

Example: Assume that GRPHCOL is a VARGRAPHIC column in table TABLEX and MIXEDSTRING is a character-string host variable that contains mixed data. For various rows in TABLEX, an application uses a positioned UPDATE statement to replace the value of GRPHCOL with the value of MIXEDSTRING. Before GRPHCOL can be updated, the current value of MIXEDSTRING must be converted to a varying-length graphic string. The following statement shows how to code the VARGRAPHIC function within the UPDATE statement to ensure this conversion.

```
EXEC SQL UPDATE TABLEX  
  SET GRPHCOL = VARGRAPHIC(:MIXEDSTRING)  
 WHERE CURRENT OF CRSNAME;
```

WEEK

```
►►—WEEK(expression)—►►
```

The schema is SYSIBM.

The WEEK function returns an integer in the range of 1 to 54 that represents the week of the year. The week starts with Sunday, and January 1 is always in the first week.

The argument must be a date, a timestamp, or a valid string representation of a date or timestamp. A string representation must not be a CLOB and must have an actual length that is not greater than 255 bytes. For the valid formats of string representations of dates and timestamps, see “String representations of datetime values” on page 57.

The result of the function is a large integer. If the argument can be null, the result can be null; if the argument is null, the result is the null value.

Example: Using sample table DSN8710.PROJ, set the integer host variable WEEK to the week of the year that project 'AD2100' ended.

```
SELECT WEEK(PRENDATE)
  INTO :WEEK
  FROM DSN8710.PROJ
 WHERE PROJNO = 'AD2100';
```

The result is that WEEK is set 6.

WEEK_ISO

WEEK_ISO

```
►►WEEK_ISO(expression)►►
```

The schema is SYSIBM.

The WEEK function returns an integer in the range of 1 to 53 that represents the week of the year. The week starts with Monday and includes 7 days. Week 1 is the first week of the year to contain a Thursday, which is equivalent to the first week containing January 4. Thus, it is possible to have up to 3 days at the beginning of the year appear as the last week of the previous year, or to have up to 3 days at the end of a year appear as the first week of the next year.

The argument must be a date, a timestamp, or a valid string representation of a date or timestamp. A string representation must not be a BLOB, CLOB, or DBCLOB and must have an actual length that is not greater than 255 bytes. For the valid formats of string representations of dates and timestamps, see “String representations of datetime values” on page 57.

The result of the function is a large integer. If the argument can be null, the result can be null; if the argument is null, the result is the null value.

Example 1: Using sample table DSN8710.PROJ, set the integer host variable WEEK to the week of the year that project 'AD2100' ended.

```
SELECT WEEK_ISO(PRENDATE)
  INTO :WEEK
  FROM DSN8710.PROJ
 WHERE PROJNO = 'AD2100';
```

Example 2: The following list shows what is returned by the WEEK_ISO function for various dates.

DATE	WEEK_ISO
1997-12-28	'52'
1997-12-31	'1'
1998-01-01	'1'
1999-01-01	'53'
1999-01-04	'1'
1999-12-31	'52'
2000-01-01	'52'
2000-01-03	'1'

YEAR

```
►►YEAR(expression)►►
```

The schema is SYSIBM.

The YEAR function returns the year part of its argument.

The argument must be a date, a timestamp, date duration, timestamp duration, or a valid string representation of a date or timestamp. A string representation must not be a BLOB, CLOB, or DBCLOB and must have an actual length that is not greater than 255 bytes. For the valid formats of string representations of dates and timestamps, see “String representations of datetime values” on page 57.

The result of the function is a large integer. If the argument can be null, the result can be null; if the argument is null, the result is the null value.

The other rules depend on the data type of the argument specified:

If the argument is a date, a timestamp, or a string representation of either, the result is the year part of the value, which is an integer between 1 and 9999.

If the argument is a date duration or a timestamp duration, the result is the year part of the value, which is an integer between -9999 and 9999. A nonzero result has the same sign as the argument.

Example: From the table DSN8710.EMP, select all rows for employees who were born in 1941.

```
SELECT *
  FROM DSN8710.EMP
 WHERE YEAR(BIRTHDATE) = 1941;
```

Table functions

A table function can be used only in the FROM clause of a statement. Table
functions return columns of a table and resemble a table created through a
CREATE TABLE statement. Table functions can be qualified with a schema name.
Following in alphabetic order is a definition of each of the table functions.

MQREADALL

#

```
# ►►MQREADALL( [receive-service] [num-rows]
# , [service-policy] )
```

#

#

#

The schema is DB2MQ1C or DB2MQ2C

The MQREADALL function returns a table containing the messages and message
meta-data from the MQSeries location that is specified by *receive-service*, using the
quality-of-service policy that is defined in *service-policy*. Performing this operation
does not remove the messages from the queue that is associated with
receive-service.

receive-service

An expression that returns a value that is a built-in character-string data type
that is not a CLOB. The value of the expression must not be the null value, an
empty string, or a string with trailing blanks. The expression must have an
actual length that is no greater than 48 bytes. The value of the expression
refers to a service point that is the logical MQSeries destination from which the
message is read. A service point is defined in the DSNAMT repository file, and it
represents a logical end-point from which a message is sent or received. A
service point definition includes the name of the MQSeries queue manager and
the name of the queue. See *MQSeries Application Messaging Interface* for
more details.

If *receive-service* is not specified, DB2.DEFAULT.SERVICE is used.

service-policy

An expression that returns a value that is a built-in character-string data type
that is not a CLOB. The value of the expression must not be the null value, an
empty string, or a string with trailing blanks. The expression must have an
actual length that is no greater than 48 bytes. The value of the expression
refers to an MQSeries AMI service policy that is used in handling this message.
A service policy is defined in the DSNAMT repository file, and it specifies a set
of quality-of-service options that are to be applied to this messaging operation.
These options include message priority and message persistence. See
MQSeries Application Messaging Interface for more details.

If *service-policy* is not specified, DB2.DEFAULT.POLICY is used.

num-rows

An expression that specifies the maximum number of messages to return. It
must be an integer that is greater than or equal to zero.

If *num-rows* is not specified or the value of expression is zero, all available
messages are returned.

The result of the function is a table with the format shown in Table 30. All the
columns are nullable.

Table 30. Format of the resulting table for MQREADALL

Column name	Data type	Contains
MSG	VARCHAR(4000)	The contents of the MQSeries message

MQREADALL

<i>Table 30. Format of the resulting table for MQREADALL (continued)</i>		
Column name	Data type	Contains
CORRELID	VARCHAR(24)	The correlation ID that is used to relate messages
TOPIC	VARCHAR(40)	The topic that the message was published with, if available
QNAME	VARCHAR(48)	The name of the queue from which the message was received
MSGID	CHAR(24)	The unique, MQSeries-assigned identifier for the message
MSGFORMAT	VARCHAR(8)	The format of the message, as defined by MQSeries

Example 1: Retrieve all the messages from the queue specified by the default service (DB2.DEFAULT.SERVICE), using the default policy (DB2.DEFAULT.POLICY).

```
SELECT *
  FROM table (MQREADALL()) T;
```

The messages and all the metadata are returned as a table.

Example 2: Retrieve the first 10 messages from the head of the queue specified by the default service (DB2.DEFAULT.SERVICE), using the default policy (DB2.DEFAULT.POLICY).

```
SELECT *
  FROM table (MQREADALL(10)) T;
```

The first 10 messages and all the columns are returned as a table.

MQREADALLCLOB

#

```
# ►►—MQREADALLCLOB(—  
#   receive-service ————  
#   ,—service-policy—  
#   num-rows ————)  
#
```

#

#

The schema is DB2MQ1C or DB2MQ2C

The MQREADALLCLOB function returns a table containing the messages and
message meta-data from the MQSeries location that is specified by *receive-service*,
using the quality-of-service policy that is defined in *service-policy*. Performing this
operation does not remove the messages from the queue that is associated with
receive-service.#
receive-service# An expression that returns a value that is a built-in character-string data type
that is not a CLOB. The value of the expression must not be the null value, an
empty string, or a string with trailing blanks. The expression must have an
actual length that is no greater than 48 bytes. The value of the expression
refers to a service point that is the logical MQSeries destination from which the
message is read. A service point is defined in the DSNAMT repository file, and
it represents a logical end-point from which a message is sent or received. A
service point definition includes the name of the MQSeries queue manager and
the name of the queue. See *MQSeries Application Messaging Interface* for
more details.#
If *receive-service* is not specified, DB2.DEFAULT.SERVICE is used.#
service-policy# An expression that returns a value that is a built-in character-string data type
that is not a CLOB. The value of the expression must not be the null value, an
empty string, or a string with trailing blanks. The expression must have an
actual length that is no greater than 48 bytes. The value of the expression
refers to an MQSeries AMI service policy that is used in handling this message.
A service policy is defined in the DSNAMT repository file, and it specifies a set
of quality-of-service options that are to be applied to this messaging operation.
These options include message priority and message persistence. See
MQSeries Application Messaging Interface for more details.#
If *service-policy* is not specified, DB2.DEFAULT.POLICY is used.#
num-rows# An expression that specifies the maximum number of messages to return. It
must be an integer that is greater than or equal to zero.#
If *num-rows* is not specified or the value of expression is zero, all available
messages are returned.#
The result of the function is a table with the format shown in Table 31. All the
columns in the table are nullable.# *Table 31. Format of the resulting table for MQREADALLCLOB*

Column name	Data type	Contains
MSG	CLOB(1M)	The contents of the MQSeries message
CORRELID	VARCHAR(24)	The correlation ID that is used to relate messages

MQREADALLCLOB

Table 31. Format of the resulting table for MQREADALLCLOB (continued)

Column name	Data type	Contains
TOPIC	VARCHAR(40)	The topic that the message was published with, if available
QNAME	VARCHAR(48)	The name of the queue from which the message was received
MSGID	CHAR(24)	The unique, MQSeries-assigned identifier for the message
MSGFORMAT	VARCHAR(8)	The format of the message, as defined by MQSeries

Example 1: Retrieve all the messages from the queue specified by the default service (DB2.DEFAULT.SERVICE), using the default policy (DB2.DEFAULT.POLICY).

```
#  
#  
#  
#  
#  
SELECT *  
FROM table (MQREADALLCLOB()) T;
```

The messages and all the metadata are returned as a table.

Example 2: Retrieve all the messages from the queue specified by the service MYSERVICE, using the default policy (DB2.DEFAULT.POLICY).

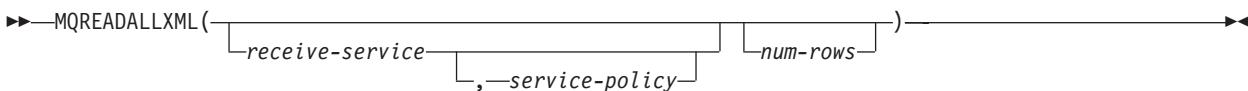
```
#  
#  
#  
#  
SELECT T.MSG, T.CORRELID  
FROM table (MQREADALLCLOB('MYSERVICE')) T;
```

Only the MSG and CORRELID columns are returned as a table.

MQREADALLXML

#

#



#

#

#

The schema is DMQXML1C or DMQXML2C.

The MQREADALLXML function returns a table containing the messages and
message meta-data from the MQSeries location that is specified by *receive-service*,
using the quality-of-service policy that is defined in *service-policy*. Performing this
operation does not remove the messages from the queue that is associated with
receive-service.

receive-service

An expression that returns a value that is a built-in character-string data type
that is not a CLOB. The value of the expression must not be the null value, an
empty string, or a string with trailing blanks. The expression must have an
actual length that is no greater than 48 bytes. The value of the expression
refers to a service point that is the logical MQSeries destination from which the
message is read. A service point is defined in the DSNAMT repository file, and
it represents a logical end-point from which a message is sent or received. A
service point definition includes the name of the MQSeries queue manager and
the name of the queue. See *MQSeries Application Messaging Interface* for
more details.

#

If *receive-service* is not specified, DB2.DEFAULT.SERVICE is used.

service-policy

An expression that returns a value that is a built-in character-string data type
that is not a CLOB. The value of the expression must not be the null value, an
empty string, or a string with trailing blanks. The expression must have an
actual length that is no greater than 48 bytes. The value of the expression
refers to an MQSeries AMI service policy that is used in handling this message.
A service policy is defined in the DSNAMT repository file, and it specifies a set
of quality-of-service options that are to be applied to this messaging operation.
These options include message priority and message persistence. See
MQSeries Application Messaging Interface for more details.

#

If *service-policy* is not specified, DB2.DEFAULT.POLICY is used.

#

num-rows

#

An expression that specifies the maximum number of messages to return. It
must be an integer that is greater than or equal to zero.

#

If *num-rows* is not specified or the value of expression is zero, all available
messages are returned.

#

The result of the function is a table with the format shown in Table 31 on page 337.
All the columns in the table are nullable.

#

Table 32. Format of the resulting table for MQREADALLXML

#

#

Column name	Data type	Contains
MSG	DB2XML.XMLVARCHAR	The contents of the MQSeries message

MQREADALLXML

Table 32. Format of the resulting table for MQREADALLXML (continued)

Column name	Data type	Contains
CORRELID	VARCHAR(24)	The correlation ID that is used to relate messages
TOPIC	VARCHAR(40)	The topic that the message was published with, if available
QNAME	VARCHAR(48)	The name of the queue from which the message was received
MSGID	CHAR(24)	The unique, MQSeries-assigned identifier for the message
MSGFORMAT	VARCHAR(8)	The format of the message, as defined by MQSeries

The CCSID of the result is the system CCSID that was in effect at the time that the MQSeries function was installed into DB2.

Example 1: Retrieve all the messages from the queue specified by the default service (DB2.DEFAULT.SERVICE), and read using the default policy (DB2.DEFAULT.POLICY).

```
#  
# SELECT *  
#        FROM TABLE (MQREADALLXML()) T;
```

The messages and all the metadata are returned as a table.

Example 2: Retrieve all the messages from the queue specified by the service MYSERVICE, using the default policy (DB2.DEFAULT.POLICY). Have only the MSG and CORRELID columns of the table returned.

```
#  
# SELECT T.MSG, T.CORRELID  
#       FROM table (MQREADALLXML('MYSERVICE')) T;
```

Example 3: Retrieve the first 10 messages from the queue specified by the default service (DB2.DEFAULT.SERVICE), using the default policy (DB2.DEFAULT.POLICY).

```
#  
# SELECT *  
#        FROM TABLE (MQREADALLXML(10)) T;
```

The first 10 messages and all the columns are returned as a table.

MQRECEIVEALL

```
# If correl-id is not specified, a correlation identifier is not used, and the message  
# at the beginning of the queue is returned.  
#  
# num-rows  
# An expression that specifies the maximum number of messages to return. It  
# must be an integer that is greater than or equal to zero.  
#  
# If num-rows is not specified or the value of expression is zero, all available  
# messages are returned.
```

```
# The result of the function is a table with the format shown in Table 33. All the  
# columns are nullable.
```

Table 33. Format of resulting table for MQRECEIVEALL

Column name	Data type	Contains
MSG	VARCHAR(4000)	The contents of the MQSeries message
CORRELID	VARCHAR(24)	The correlation ID that is used to relate messages
TOPIC	VARCHAR(40)	The topic that the message was published with, if available
QNAME	VARCHAR(48)	The name of the queue from which the message was received
MSGID	CHAR(24)	The unique, MQSeries-assigned identifier for the message
MSGFORMAT	VARCHAR(8)	The format of the message, as defined by MQSeries

```
# Example 1: Retrieve all the messages from the queue specified by the default  
# service (DB2.DEFAULT.SERVICE), using the default policy  
# (DB2.DEFAULT.POLICY).  
#  
# SELECT *  
#        FROM table (MQRECEIVEALL()) T;
```

```
# The messages and all the metadata are returned as a table and deleted from the  
# queue.
```

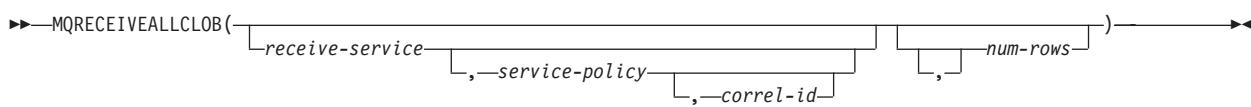
```
# Example 2: Retrieve all the messages from the head of the queue specified by the  
# service MSERVICE, using the policy MYPOLICY.  
#  
# SELECT T.MSG, T.CORRELID  
#        FROM table (MQRECEIVEALL('MYSERVICE','MYPOLICY','1234')) t;
```

```
# Only messages with CORRELID of '1234' and only the MSG and CORRELID  
# columns are returned as a table and removed from the queue.
```

MQRECEIVEALLCLOB

#

#



#

#

#

The schema is DB2MQ1C or DB2MQ2C

The MQRECEIVEALLCLOB function returns a table containing the messages and
message metadata from the MQSeries location that is specified by *receive-service*,
using the quality-of-service policy that is defined in *service-policy*. Performing this
operation removes the messages from the queue that is associated with
receive-service.

receive-service

An expression that returns a value that is a built-in character-string data type that is not a CLOB. The value of the expression must not be the null value, an empty string, or a string with trailing blanks. The expression must have an actual length that is no greater than 48 bytes. The value of the expression refers to a service point that is the logical MQSeries destination from which the message is read. A service point is defined in the DSNAME repository file, and it represents a logical end-point from which a message is sent or received. A service point definition includes the name of the MQSeries queue manager and the name of the queue. See *MQSeries Application Messaging Interface* for more details.

If *receive-service* is not specified, DB2.DEFAULT.SERVICE is used.

service-policy

An expression that returns a value that is a built-in character-string data type that is not a CLOB. The value of the expression must not be the null value, an empty string, or a string with trailing blanks. The expression must have an actual length that is no greater than 48 bytes. The value of the expression refers to an MQSeries AMI service policy that is used in handling this message. A service policy is defined in the DSNAME repository file, and it specifies a set of quality-of-service options that are to be applied to this messaging operation. These options include message priority and message persistence. See *MQSeries Application Messaging Interface* for more details.

If *service-policy* is not specified, DB2.DEFAULT.POLICY is used.

correl-id

An expression that returns a value that is a built-in character-string data type that is not a CLOB. The expression must have an actual length that is no greater than 24 bytes. The value of the expression specifies the correlation identifier that is associated with this message. A correlation identifier is often specified in request-and-reply scenarios to associate requests with replies. Only those messages with a matching correlation identifier are returned.

A null value, an empty string, and a fixed length string with trailing blanks are all considered valid values. However, when the *correl-id* is specified on another request such as MQSEND, the *correl-id* must be specified the same to be recognized as a match. For example, specifying a value of 'test' for *correl-id* on MQRECEIVEALLCLOB does not match a *correl-id* value of 'test ' (with trailing blanks) specified earlier on an MQSEND request.

MQRECEIVEALLCLOB

```
# If correl-id is not specified, a correlation identifier is not used, and the message  
# at the beginning of the queue is returned.  
#  
# num-rows  
# An expression that specifies the maximum number of messages to return. It  
# must be an integer that is greater than or equal to zero.  
#  
# If num-rows is not specified or the value of expression is zero, all available  
# messages are returned.
```

```
# The result of the function is a table with the format shown in Table 34. All the  
# columns are nullable.
```

Table 34. Format of resulting table for MQRECEIVEALLCLOB

Column name	Data type	Contains
MSG	CLOB(1M)	The contents of the MQSeries message
CORRELID	VARCHAR(24)	The correlation ID that is used to relate messages
TOPIC	VARCHAR(40)	The topic that the message was published with, if available
QNAME	VARCHAR(48)	The name of the queue from which the message was received
MSGID	CHAR(24)	The unique, MQSeries-assigned identifier for the message
MSGFORMAT	VARCHAR(8)	The format of the message, as defined by MQSeries

```
# Example 1: Retrieve all the messages from the queue specified by the default  
# service (DB2.DEFAULT.SERVICE), using the default policy  
# (DB2.DEFAULT.POLICY).  
#  
# SELECT *  
#       FROM table (MQRECEIVEALLCLOB()) T;
```

```
# The messages and all the metadata are returned as a table and deleted from the  
# queue.
```

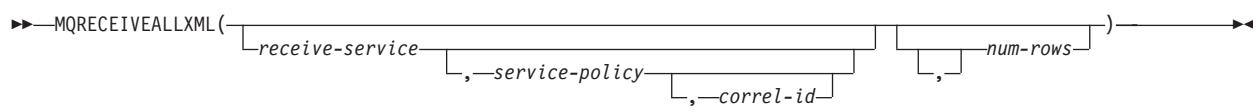
```
# Example 2: Retrieve all the messages from the head of the queue specified by the  
# service MSERVICE, using the default policy (DB2.DEFAULT.POLICY).  
#  
# SELECT T.MSG, T.CORRELID  
#       FROM table (MQRECEIVEALLCLOB('MYSERVICE')) T;
```

```
# Only the MSG and CORRELID columns are returned as a table and removed from  
# the queue.
```

MQRECEIVEALLXML

#

#



#

#

#

The schema is DMQXML1C or DMQXML2C.

The **MQRECEIVEALLXML** function returns a table containing the messages and message metadata from the MQSeries location that is specified by *receive-service*, using the quality-of-service policy that is defined in *service-policy*. Performing this operation removes the messages from the queue that is associated with *receive-service*.

receive-service

An expression that returns a value that is a built-in character-string data type
that is not a CLOB. The value of the expression must not be the null value, an
empty string, or a string with trailing blanks. The expression must have an
actual length that is no greater than 48 bytes. The value of the expression
refers to a service point that is the logical MQSeries destination from which the
message is read. A service point is defined in the DSNAME repository file, and
it represents a logical end-point from which a message is sent or received. A
service point definition includes the name of the MQSeries queue manager and
the name of the queue. See *MQSeries Application Messaging Interface* for
more details.

#

If *receive-service* is not specified, DB2.DEFAULT.SERVICE is used.**service-policy**

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

An expression that returns a value that is a built-in character-string data type that is not a CLOB. The value of the expression must not be the null value, an empty string, or a string with trailing blanks. The expression must have an actual length that is no greater than 48 bytes. The value of the expression refers to an MQSeries AMI service policy that is used in handling this message. A service policy is defined in the DSNAME repository file, and it specifies a set of quality-of-service options that are to be applied to this messaging operation. These options include message priority and message persistence. See *MQSeries Application Messaging Interface* for more details.

#

If *service-policy* is not specified, DB2.DEFAULT.POLICY is used.**correl-id**

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

#

MQRECEIVEALLXML

```
# If correl-id is not specified, a correlation identifier is not used, and the message  
# at the beginning of the queue is returned.
```

num-rows

```
# An expression that specifies the maximum number of messages to return. It  
# must be an integer that is greater than or equal to zero.
```

```
# If num-rows is not specified or the value of expression is zero, all available  
# messages are returned.
```

```
# The result of the function is a table with the format shown in Table 34 on page 344.  
# All the columns are nullable.
```

Table 35. Format of resulting table for MQRECEIVEALLXML

Column name	Data type	Contains
MSG	DB2XML.XMLVARCHAR	The contents of the MQSeries message
CORRELID	VARCHAR(24)	The correlation ID that is used to relate messages
TOPIC	VARCHAR(40)	The topic that the message was published with, if available
QNAME	VARCHAR(48)	The name of the queue from which the message was received
MSGID	CHAR(24)	The unique, MQSeries-assigned identifier for the message
MSGFORMAT	VARCHAR(8)	The format of the message, as defined by MQSeries

```
# Example 1: Retrieve all the messages from the queue specified by the default  
# service (DB2.DEFAULT.SERVICE), using the default policy  
(DB2.DEFAULT.POLICY).
```

```
SELECT *  
FROM TABLE (MQRECEIVEALLXML()) T;
```

```
# The messages and all the metadata are returned as a table. The messages are  
# deleted from the queue.
```

```
# Example 2: Retrieve all the messages from the beginning of the queue specified by  
# the service MYSERVICE, using the default policy (DB2.DEFAULT.POLICY). Have  
only the MSG and CORRELID columns returned.
```

```
SELECT T.MSG, T.CORRELID  
FROM TABLE (MQRECEIVEALLXML('MYSERVICE')) T;
```

```
# The messages and the data for the CORRELID column are returned as a table.  
# The messages are deleted from the queue.
```

```
# Example 3: Retrieve the first 10 the messages from the beginning of the queue  
# specified by the default service (DB2.DEFAULT.SERVICE), using the default policy  
(DB2.DEFAULT.POLICY).
```

```
SELECT *  
FROM table (MQRECEIVEALLXML(10)) T;
```

```
# The messages and all the metadata for the first 10 messages are returned as a  
# table. The messages are deleted from the queue.
```

Chapter 4. Queries

Authorization	348
subselect	349
select-clause	349
from-clause.	352
table-spec	353
joined-table.	356
join-condition	357
Join operations	358
where-clause	358
group-by-clause	359
having-clause	359
Examples of subselects	360
fullselect.	365
Character conversion in unions and concatenations	366
Selecting the result CCSID	366
Examples of fullselects	367
select-statement	369
order-by-clause	370
read-only-clause	371
update-clause	372
optimize-for-clause	372
with-clause	373
queryno-clause	374
fetch-first-clause	374
Examples of select statements	375

|

Queries

A *query* specifies a result table. A query is a component of certain SQL statements. There are three forms of a query:

- A *subselect*
- A *fullselect*
- A *select-statement*

A subselect is a subset of a fullselect, and a fullselect is a subset of a select-statement.

Another SQL statement called SELECT INTO is described in “SELECT INTO” on page 900. SELECT INTO is not a subselect, fullselect, or a select-statement.

Authorization

The privilege set that is defined below must include one of the following:

- Ownership of the table or view
- The SELECT privilege on the table or view
- DBADM authority for the database (tables only)
- SYSADM authority
- SYSCTRL authority (catalog tables only)

For each user-defined function that is referenced in a query, the EXECUTE privilege on the user-defined function is also required.

If the *select-statement* is part of a DECLARE CURSOR statement, the privilege set is the privileges that are held by the authorization ID of the owner of the plan or package.

For dynamically prepared statements, the privilege set depends on the dynamic SQL statement behavior, which is specified by bind option DYNAMICRULES:

Run behavior

The privilege set is the union of the privilege sets that are held by each authorization ID of the process.

Bind behavior

The privilege set is the privileges that are held by the authorization ID of the owner of the plan or package.

Define behavior

The privilege set is the privileges that are held by the authorization ID of the owner of the stored procedure or user-defined function.

Invoke behavior

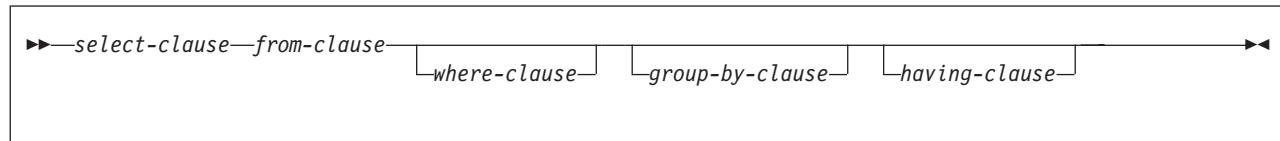
The privilege set is the privileges that are held by the authorization ID of the invoker of the stored procedure or user-defined function.

For a list of the DYNAMICRULES values that specify run, bind, define, or invoke behavior, see Table 2 on page 44.

When any form of a query is used as a component of another statement, the authorization rules that apply to the query are specified in the description of that statement. For example, see “CREATE VIEW” on page 711 for the authorization rules that apply to the subselect component of CREATE VIEW.

If your installation uses the access control authorization exit (DSNX@XAC), that exit may be controlling the authorization rules instead of the rules that are listed here.

subselect



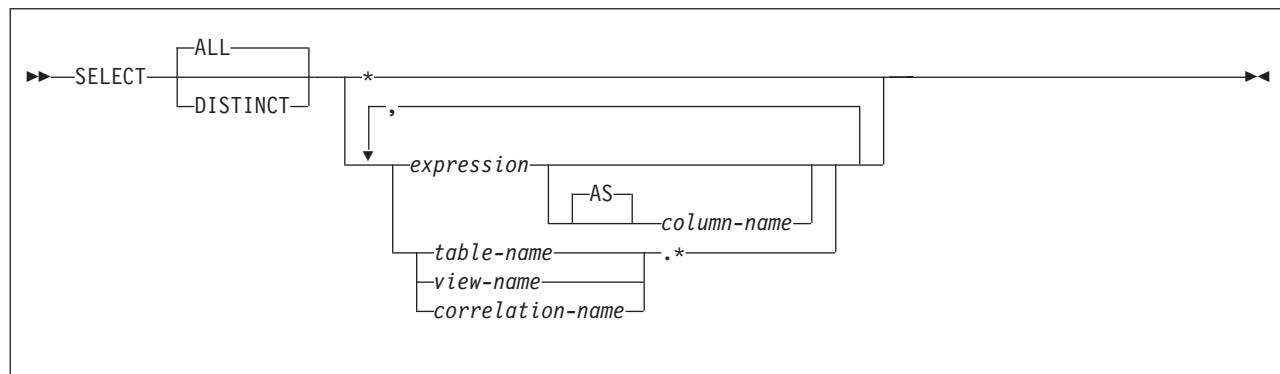
The *subselect* is a component of the fullselect.

A subselect specifies a result table derived from the result of its first FROM clause. The derivation can be described as a sequence of operations in which the result of each operation is input for the next. (This is only a way of describing the subselect. The method used to perform the derivation may be quite different from this description.)

The clauses of the subselect are processed in the following sequence:

1. FROM clause
2. WHERE clause
3. GROUP BY clause
4. HAVING clause
5. SELECT clause

select-clause



The SELECT clause specifies the columns of the final result table. The column values are produced by the application of the *select list* to R. The select list is a list of names and expressions specified in the SELECT clause, and R is the result of the previous operation of the subselect. For example, if SELECT, FROM, and WHERE are the only clauses specified, then R is the result of that WHERE clause.

ALL

Retains all rows of the final result table and does not eliminate redundant duplicates. This is the default.

DISTINCT

Eliminates all but one of each set of duplicate rows of the final result table. DISTINCT must not be used more than once in a subselect, with the exception of its use with a column function whose expression is a column. The same DISTINCT column function with the same column expression can be referred to more than once in a subselect. This restriction includes SELECT DISTINCT and the use of DISTINCT in a column function of the select list or HAVING clause. It does not include occurrences of DISTINCT in subqueries of the subselect.

subselect

Two rows are duplicates of one another only if each value in the first row is equal to the corresponding value in the second row. For determining duplicate rows, two null values are considered equal.

Select list notation:

- * Represents a list of names that identify the columns of table R. The first name in the list identifies the first column of R, the second name identifies the second column of R, and so on.

The list of names is established when the statement containing the SELECT clause is prepared. Therefore, * does not identify any columns that have been added to a table after the statement has been prepared.

expression

Can be any expression of the type that is described in “Expressions” on page 111. Each *column-name* in the expression must unambiguously identify a column of R.

AS *column-name*

Names or renames the result column. The name must not be qualified and does not have to be unique.

*name.**

Represents a list of names that identify the columns of *name*. *name* can be a table name, view name, or correlation name, and must designate a table or view named in the FROM clause. If a table is specified, it must not be an auxiliary table. The first name in the list identifies the first column of the table or view, the second name in the list identifies the second column of the table or view, and so on.

The list of names is established when the statement containing the SELECT clause is prepared. Therefore, * does not identify any columns that have been added to a table after the statement has been prepared.

SQL statements can be implicitly or explicitly rebound (prepared again). The effect of a rebind on statements that include * or *name.** is that the list of names is re-established. Therefore, the number of columns returned by the statement may change.

The number of columns in the result of SELECT is the same as the number of expressions in the operational form of the select list (that is, the list established at the time the statement is prepared), and cannot exceed 750. The result of a subquery must be a single column unless the subquery is used in an EXISTS predicate.

Limitation on long string columns: The result of an expression must not be a character string with a maximum length greater than 255 or a graphic string with a maximum length greater than 127 if:

- SELECT DISTINCT is used.
- The subselect is a subquery.
- The subselect is an operand of UNION.

Applying the select list: Some of the results of applying the select list to R depend on whether GROUP BY or HAVING is used. The next two separate lists describe the results.

If neither GROUP BY nor HAVING is used:

- ```
• The select list can include column functions only if it includes other column
functions, constants, or expressions that only involve constants.
 • If the select list does not include column functions, it is applied to each row of R
 and the result contains as many rows as there are rows in R.
 • If the select list includes column functions, R is the source of the arguments of
 the functions and the result of applying the select list is one row, even when R
 has no rows.
```

**If GROUP BY or HAVING is used:**

- ```
#          • Each column-name in the select list must either identify a grouping column or be
#            specified within a column function. Constants or expressions that involve only
#            constants can also be in the select list.
          • The select list is applied to each group of R, and the result contains as many
            rows as there are groups in R. When the select list is applied to a group of R,
            that group is the source of the arguments of the column functions in the select
            list.
          • You cannot use GROUP BY with a name defined using the AS clause unless the
            name is defined in a nested table expression. “Example 6” on page 361
            demonstrates the valid use of AS and GROUP BY in a SELECT statement.
```

In either case, the *n*th column of the result contains the values specified by applying the *n*th expression in the operational form of the select list.

Null attributes of result columns: Result columns allow null values if they are derived from one of the following:

- Any column function except COUNT or COUNT_BIG
- A column that allows null values
- A view column in an outer select list that is derived from an arithmetic expression
- An arithmetic expression in an outer select list
- An arithmetic expression that allows nulls
- A scalar function or string expression that allows null values
- A host variable that has an indicator variable
- A result of a UNION if at least one of the corresponding items in the select list is nullable

Names of result columns: The name of a result column of a subselect is determined as follows:

- If the AS clause is specified, the name of the result column is the name specified on the AS clause. The name need not be unique.
- If the AS clause is not specified and the result column is derived from a column name, the result column name is the unqualified name of that column.
- All other result columns are unnamed.

Names of result columns are placed into the SQL descriptor area (SQLDA) when the DESCRIBE statement is executed. This allows an interactive SQL processor such as SPUFI or QMF to use the column names when displaying the results. The names in the SQLDA include those specified by the AS clause.

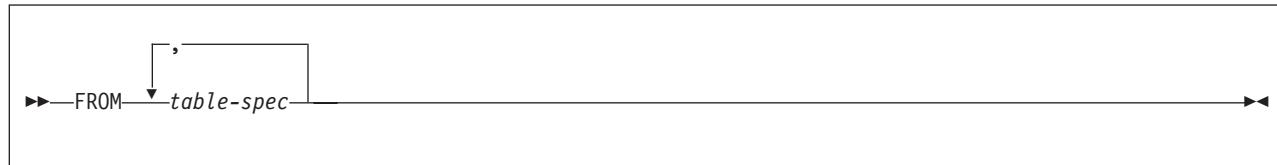
subselect

Data types of result columns: Each column of the result of SELECT acquires a data type from the expression from which it is derived.

Table 36. Data types of result columns

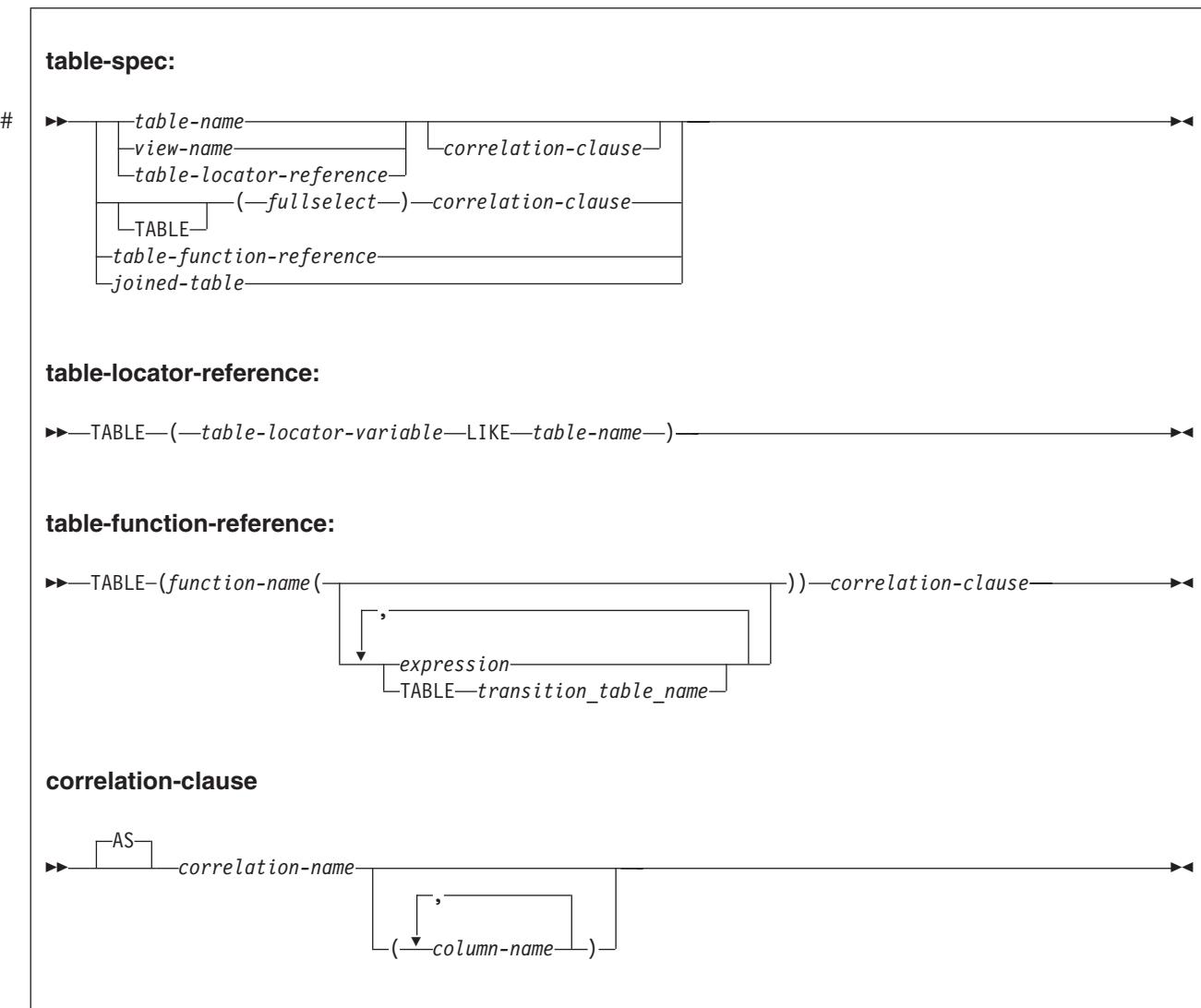
When the expression is...	The data type of the result column is...
The name of any numeric column	The same as the data type of the column, with the same precision and scale for decimal columns.
An integer constant	INTEGER.
A decimal or floating-point constant	The same as the data type of the constant, with the same precision and scale for decimal constants. For floating-point constants, the data type is DOUBLE PRECISION.
The name of any numeric host variable	The same as the data type of the variable, with the same precision and scale for decimal variables. The result is decimal if the data type of the host variable is not an SQL data type; for example, DISPLAY SIGN LEADING SEPARATE in COBOL.
An arithmetic or string expression	The same as the data type of the result, with the same precision and scale for decimal results as described in “Expressions” on page 111.
Any function	The data type of the result of the function. For a built-in function, see Chapter 3, “Functions,” on page 155 to determine the data type of the result. For a user-defined function, the data type of the result is what was defined in the CREATE FUNCTION statement for the function.
The name of any string column	The same as the data type of the column, with the same length attribute.
The name of any string host variable	The same as the data type of the variable, with a length attribute equal to the length of the variable. The result is a varying-length character string if the data type of the host variable is not an SQL data type; for example, a NUL-terminated string in C.
A character string constant of length <i>n</i>	VARCHAR(<i>n</i>).
A graphic string constant of length <i>n</i>	VARGRAPHIC(<i>n</i>).
The name of a datetime column	The same as the data type of the column.
The name of a ROWID column	Row ID.
The name of a distinct type column	The same as the distinct type of the column, with the same length, precision, and scale attributes, if any.

from-clause



The FROM clause specifies an intermediate result table, R. If a single *table-spec* is specified, R is the result of that *table-spec*. If more than one *table-spec* is specified, R consists of all possible combinations of the rows of the result of each *table-spec*. Each row of R is a row from the result of the first *table-spec* concatenated with a row from the result of the second *table-spec*, concatenated with a row from the result of the third *table-spec*, and so on. The number of rows in R is the product of the number of rows in the result of each *table-spec*. Thus, if the result of any *table-spec* is empty, R is empty.

table-spec



A *table-spec* specifies an intermediate result table:

- If a single table or view is identified, the intermediate result table is simply that table or view.
 - If a table locator is identified, the host variable represents the intermediate table. The intermediate table has the same structure as the table identified in *table-name*.
 - A *fullselect* in parentheses is called a *nested table expression*. If a nested table expression is specified, the result table is the result of that nested table expression. The columns of the result do not need unique names, but a column with a non-unique name cannot be referenced. At any time, the table consists of the rows that would result if the fullselect were executed.
 - If a *function-name* is specified, the intermediate result table is the set of rows returned by the table function.
 - If a *joined-table* is specified, the intermediate result table is the result of one or more join operations as explained below.

subselect

Each *table-name* or *view-name* specified in every FROM clause of the same SQL statement must identify a table or view that exists at the same DB2 subsystem. The tables that are identified must not be auxiliary tables. The tables, table functions, or underlying tables of the views that are identified must have the same encoding scheme—either all ASCII, all EBCDIC, or all Unicode. If a FROM clause is specified in a subquery of a basic predicate, a view that includes GROUP BY or HAVING must not be identified.

Each *table-locator-variable* must specify a host variable with a table locator type. The only way to assign a value to a table locator is to pass the old or new transition table of a trigger to a user-defined function or stored procedure. A table locator host variable must not have a null indicator and must not be a parameter marker. In addition, a table locator can be used only in a manipulative SQL statement.

Each *function-name*, together with the types of its arguments, must resolve to a table function that exists at the same DB2 subsystem. An algorithm called function resolution, which is described on page 107, uses the function name and the arguments to determine the exact function to use. Unless given column names in the *correlation-clause*, the column names for a table function are those specified on the RETURNS clause of the CREATE FUNCTION statement. This is analogous to the column names of a table, which are defined in the CREATE TABLE.

Each *correlation-name* in a *correlation-clause* defines a designator for the immediately preceding intermediate result table (*table-name*, *view-name*, nested table expression, or *function-name* reference), which can be used to qualify references to the columns of the table. Using *column-names* to list and rename the columns is optional. A correlation name must be specified for nested table expressions and references to table functions.

If a list of *column-names* is specified in a *correlation-clause*, the number of names must be the same as the number of columns in the corresponding table, view, nested table expression, or table function. Each name must be unique and unqualified. If columns are added to an underlying table of a *table-spec*, the number of columns in the result of the *table-spec* no longer matches the number of names in its *correlation-clause*. Therefore, when a rebind of a package containing the query in question is attempted, DB2 returns an error and the rebind fails. At that point, change the *correlation-clause* of the embedded SQL statement in the application program so that the number of names matches the number of columns. Then, precompile, compile, bind, and link-edit the modified program.

An exposed name is a *correlation-name* or a *table-name* or *view name* that is not followed by a *correlation-name*. The exposed names in a FROM clause should be unique, and only exposed names should be used as qualifiers of column names. Thus, if the same table name is specified twice, at least one specification of the table name should be followed by a unique correlation name. That correlation name should be used to qualify references to columns of that instance of the table. In addition, if column names are listed for the correlation name in the FROM clause, those column names should be used to reference the columns. For more information, see “Column name qualifiers in correlated references” on page 97.

Correlated references in table-specs: In general, nested table expressions and table functions can be specified in any FROM clause. Columns from the nested table expressions and table functions can be referenced in the select list and in the rest of the fullselect using the correlation name. The scope of this correlation name is the same as correlation names for other table or view names in the FROM

clause. The basic rule that applies for both these cases is that the correlated reference must be from a *table-spec* at a higher level in the hierarchy of subqueries.

Nested table expressions can be used in place of a view to avoid creating a view when general use of the view is not required. They can also be used when the desired result table is based on host variables.

For table functions, an additional capability exists. A table function can contain one or more correlated references to other tables in the same FROM clause if the referenced tables precede the reference in the left-to-right order of the tables in the FROM clause. The same capability exists for nested table expressions if the optional keyword TABLE is specified; otherwise, only references to higher levels in the hierarchy of subqueries is allowed.

A nested table expression or table function that contains correlated references to other tables in the same FROM clause:

- Cannot participate in a FULL OUTER JOIN or a RIGHT OUTER JOIN
- Can participate in LEFT OUTER JOIN or an INNER JOIN if the referenced tables precede the reference in the left-to-right order of the tables in the FROM clause

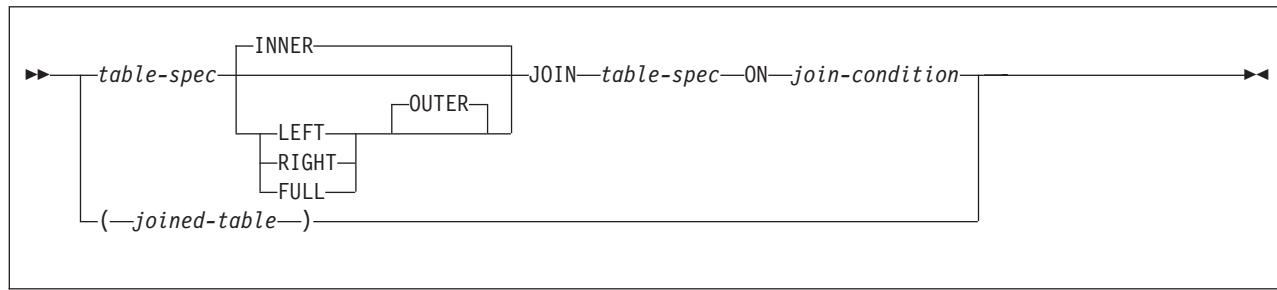
Table 37 shows some examples of valid and invalid correlated references. TABF1 and TABF2 represent table functions.

Table 37. Examples of correlated references

Subselect	Valid	Reason
<pre>SELECT T.C1, Z.C5 FROM TABLE(TABF1(T.C2)) AS Z, T WHERE T.C3 = Z.C4;</pre>	No	T.C2 cannot be resolved because T does not precede TABF1 in FROM
<pre>SELECT T.C1, Z.C5 FROM T, TABLE(TABF1(T.C2)) AS Z WHERE T.C3 = Z.C4;</pre>	Yes	T precedes TABF1 in FROM, making T.C2 known
<pre>SELECT A.C1, B.C5 FROM TABLE(TABF2(B.C2)) AS A, TABLE(TABF1(A.C6)) AS B WHERE A.C3 = B.C4;</pre>	No	B in B.C2 cannot be resolved because the table function that would resolve it, TABF1, follows its reference in TABF2 in FROM
<pre>SELECT D.DEPTNO, D.DEPTNAME, EMPINFO.AVGSAL, EMPINFO.EmpCount FROM DEPT D, (SELECT AVG(E.Salary) AS AVGSAL, COUNT(*) AS EmpCount FROM EMP E WHERE E.WorkDept = D.DeptNo) AS EMPINFO;</pre>	No	DEPT precedes nested table expression, but keyword TABLE is not specified, making D.DEPTNO unknown
<pre>SELECT D.DEPTNO, D.DEPTNAME, EMPINFO.AVGSAL, EMPINFO.EmpCount FROM DEPT D, TABLE (SELECT AVG(E.Salary) AS AVGSAL, COUNT(*) AS EmpCount FROM EMP E WHERE E.WorkDept = D.DeptNo) AS EMPINFO;</pre>	Yes	DEPT precedes nested table expression and keyword TABLE is specified, making D.DEPTNO known

subselect

joined-table



A *joined-table* specifies an intermediate result table that is the result of either an inner equi-join or an outer join. The table is derived by applying one of the join-operators: INNER, RIGHT OUTER, LEFT OUTER, or FULL OUTER to its operands. If a join-operator is not specified, INNER is implicit. The order in which a LEFT OUTER JOIN or RIGHT OUTER JOIN is performed can affect the result.

As described in more detail under “Join operations” on page 358 an inner join combines each row of the left table with every row of the right table keeping only the rows where the join-condition is true. Thus, the result table may be missing rows from either or both of the joined tables. Outer joins include the rows produced by the inner join as well as the missing rows, depending on the type of outer join as follows:

Left outer. Includes the rows from the left table that were missing from the inner join.

Right Outer. Includes the rows from the right table that were missing from the inner join.

Full Outer. Includes the rows from both tables that were missing from the inner join.

A joined-table can be used in any context in which any form of the SELECT statement is used. Both a view and a cursor is read-only if its SELECT statement includes a joined-table.

join-condition

For INNER, LEFT OUTER, and RIGHT OUTER joins:

►►—*search-condition*—►►

For FULL OUTER joins:

►►—*full-join-expression*—AND—*full-join-expression*—►►

full-join-expression:

►►—*column-name*—(1)—*cast-function*—
 —COALESCE—*column-name*—(1)—*cast-function*—
 —VALUE—*column-name*—(1)—*cast-function*—
 —, *column-name*—(1)—*cast-function*—
 —, *column-name*—(1)—*cast-function*—)—►►

Notes:

- 1 *cast-function* must only contain a column and the casting data type must be a distinct type or the data type upon which the distinct type was based.

For INNER, LEFT OUTER, and RIGHT OUTER joins, the *join-condition* is a *search-condition* that must conform to these rules:

- # It cannot contain any subqueries. However, if the join-table that contains the *join-condition* in the associated FROM clause is composed of only INNER joins, then the *join-condition* may contain subqueries.
- # Any column that is referenced in an expression of the join-condition must be a column of one of the operand tables of the associated join operator (in the scope of the same joined-table clause).

For a FULL OUTER (or FULL) join, the *join-condition* is a search condition in which the predicates can only be combined with AND. In addition, each predicate must have the form 'expression = expression', where one expression references only columns of one of the operand tables of the associated join operator, and the other expression references only columns of the other operand table. The values of the expressions must be comparable.

Each *full-join-expression* in a FULL OUTER join must include a column name or a cast function that references a column. The COALESCE and VALUE functions are allowed.

For any type of join, column references in an expression of the join-condition are resolved using the rules for resolution of column name qualifiers specified in

subselect

“Resolution of column name qualifiers and column names” on page 98 before any rules about which tables the columns must belong to are applied.

Join operations

A *join-condition* specifies pairings of T1 and T2, where T1 and T2 are the left and right operand tables of its associated JOIN operator. For all possible combinations of rows T1 and T2, a row of T1 is paired with a row of T2 if the join-condition is true. When a row of T1 is joined with a row of T2, a row in the result consists of the values of that row of T1 concatenated with the values of that row of T2. The execution might involve the generation of a “null row”. The null row of a table consists of a null value for each column of the table, regardless of whether the columns allow null values.

The following summarizes the results of the join operations:

- The result of T1 INNER JOIN T2 consists of their paired rows.
- The result of T1 LEFT OUTER JOIN T2 consists of their paired rows and, for each unpaired row of T1, the concatenation of that row with the null row of T2. All columns derived from T2 allow null values.
- The result of T1 RIGHT OUTER JOIN T2 consists of their paired rows and, for each unpaired row of T2, the concatenation of that row with the null row of T1. All columns derived from T1 allow null values.
- The result of T1 FULL OUTER JOIN T2 consists of their paired rows and, for each unpaired row of T1, the concatenation of that row with the null row of T2, and for each unpaired row of T2, the concatenation of that row with the null row in T1. All columns of the result table allow null values.

A join operation is part of a FROM clause; therefore, for the purpose of predicting which rows will be returned from a SELECT statement containing a join operation, assume that the join operation is performed before the other clauses in the statement.

where-clause

►—WHERE—*search-condition*—►

The WHERE clause specifies an intermediate result table that consists of those rows of R for which the search condition is true. R is the result of the FROM clause of the subselect.

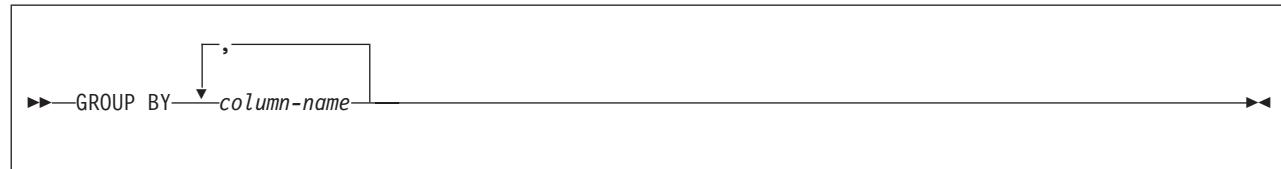
The search condition must conform to the following rules:

- Each column name must unambiguously identify a column of R or be a correlated reference. A column name is a correlated reference if it identifies a column of a table or view that is identified in an outer subselect.
- A column function must not be specified unless the WHERE clause is specified in a subquery of a HAVING clause and the argument of the function is a correlated reference to a group.

Any subquery in the *search-condition* is effectively executed for each row of R and the results are used in the application of the *search-condition* to the given row of R. A subquery is actually executed for each row of R only if it includes a correlated

reference. In fact, a subquery with no correlated references is executed just once, whereas a subquery with a correlated reference may have to be executed once for each row.

group-by-clause



The GROUP BY clause specifies an intermediate result table that consists of a grouping of the rows of R. R is the result of the previous clause.

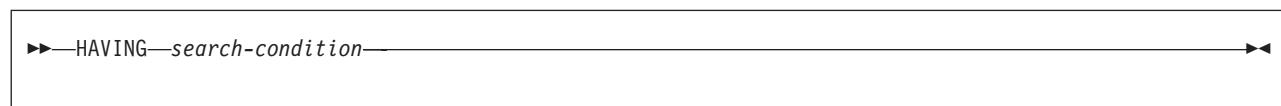
Each *column-name* must unambiguously identify a column of R other than a long string column. Each identified column is called a *grouping column*.

The result of GROUP BY is a set of groups of rows. In each group of more than one row, all values of each grouping column are equal; and all rows with the same set of values of the grouping columns are in the same group. For grouping, all null values within a grouping column are considered equal.

Because every row of a group contains the same value of any grouping column, the name of a grouping column can be used in a search condition in a HAVING clause or an expression in a SELECT clause. In each case, the reference specifies only one value for each group. However, if the grouping column contains varying-length strings with trailing blanks, the values in the group can differ in the number of trailing blanks and may not all have the same length. In that case, a reference to the grouping column still specifies only one value for each group, but the value for a group is chosen arbitrarily from the available set of values. Thus, the actual length of the result value is unpredictable.

The GROUP BY clause must not be used in a subquery of a basic predicate or must not be used if R is derived from a view whose outer subselect includes GROUP BY or HAVING clauses.

having-clause



The HAVING clause specifies an intermediate result table that consists of those groups of R for which the search-condition is true. R is the result of the previous clause. If this clause is not GROUP BY, R is considered a single group with no grouping columns.

Each *column-name* in *search-condition* must:

- Unambiguously identify a grouping column of R, or
- Be specified within a column function²⁴, or

24. See Chapter 3, “Functions,” on page 155 for restrictions that apply to the use of column functions.

subselect

- Be a correlated reference. A *column-name* is a correlated reference if it identifies a column of a table or view identified in an outer subselect.

A group of R to which the search condition is applied supplies the argument for each function in the search condition, except for any function whose argument is a correlated reference.

If the search condition contains a subquery, the subquery can be thought of as being executed each time the search condition is applied to a group of R, and the results used in applying the search condition. In actuality, the subquery is executed for each group only if it contains a correlated reference. For an illustration of the difference, see Example 4 and Example 5 in “Examples of subselects” below.

A correlated reference to a group of R must either identify a grouping column or be contained within a column function.

The HAVING clause must not be used in a subquery of a basic predicate. When HAVING is used without GROUP BY, any column name in the select list must appear within a column function.

Examples of subselects

Example 1: Show all rows of the table DSN8710.EMP.

```
SELECT * FROM DSN8710.EMP;
```

Example 2: Show the job code, maximum salary, and minimum salary for each group of rows of DSN8710.EMP with the same job code, but only for groups with more than one row and with a maximum salary greater than 50000.

```
SELECT JOB, MAX(SALARY), MIN(SALARY)
  FROM DSN8710.EMP
 GROUP BY JOB
 HAVING COUNT(*) > 1 AND MAX(SALARY) > 50000;
```

Example 3: For each employee in department E11, get the following information from the table DSN8710.EMPPROJECT: employee number, activity number, activity start date, and activity end date. Using the CHAR function, convert the start and end dates to their USA formats. Get the needed department information from the table DSN8710.EMP.

```
SELECT EMPNO, ACTNO, CHAR(EMSTDATE,USA), CHAR(EMENDATE,USA)
  FROM DSN8710.EMPPROJECT
 WHERE EMPNO IN (SELECT EMPNO FROM DSN8710.EMP
                  WHERE WORKDEPT = 'E11');
```

Example 4: Show the department number and maximum departmental salary for all departments whose maximum salary is less than the average salary for all employees. (In this example, the subquery would be executed only once.)

```
SELECT WORKDEPT, MAX(SALARY)
  FROM DSN8710.EMP
 GROUP BY WORKDEPT
 HAVING MAX(SALARY) < (SELECT AVG(SALARY)
                           FROM DSN8710.EMP);
```

Example 5: Show the department number and maximum departmental salary for all departments whose maximum salary is less than the average salary for employees in all other departments. (In contrast to Example 4, the subquery in this statement, containing a correlated reference, would need to be executed for each group.)

```

SELECT WORKDEPT, MAX(SALARY)
  FROM DSN8710.EMP Q
 GROUP BY WORKDEPT
 HAVING MAX(SALARY) < (SELECT AVG(SALARY)
                           FROM DSN8710.EMP
                          WHERE NOT WORKDEPT = Q.WORKDEPT);

```

Example 6: For each group of employees hired during the same year, show the year-of-hire and current average salary. (This example demonstrates how to use the AS clause in a FROM clause to name a derived column that you want to refer to in a GROUP BY clause.)

```

SELECT HIREYEAR, AVG(SALARY)
  FROM (SELECT YEAR(HIREDATE) AS HIREYEAR, SALARY
            FROM DSN8710.EMP) AS NEWEMP
 GROUP BY HIREYEAR;

```

Example 7: For an example of how to group the results of a query by an expression in the SELECT clause without having to retype the expression, see “Example 4” on page 126 for CASE expressions.

Example 8: Get the employee number and employee name for all the employees in DSN8710.EMP. Order the results by the date of hire.

```

SELECT EMPNO, FIRSTNME, LASTNAME
  FROM DSN8710.EMP
 ORDER BY HIREDATE;

```

Example 9: Assume that an external function named ADDYEARS exists. For a given date, the function adds a given number of years and returns a new date. (The data types of the two input parameters to the function are DATE and INTEGER.) Get the employee number and employee name for all employees who have been hired within the last 5 years.

```

SELECT EMPNO, FIRSTNME, LASTNAME
  FROM DSN8710.EMP
 WHERE ADDYEARS(HIREDATE, 5) > CURRENT DATE;

```

To distinguish the different types of joins, to show nested table expressions, and to demonstrate how to combine join columns, the remaining examples use these two tables:

The PARTS table

PART	PROD#	SUPPLIER
WIRE	10	ACWF
OIL	160	WESTERN_CHEM
MAGNETS	10	BATEMAN
PLASTIC	30	PLASTIK_CORP
BLADES	205	ACE_STEEL

The PRODUCTS table

PROD#	PRODUCT	PRICE
505	SCREWDRIVER	3.70
30	RELAY	7.55
205	SAW	18.90
10	GENERATOR	45.75

Example 10: Join the tables on the PROD# column to get a table of parts with their suppliers and the products that use the parts:

```

SELECT PART, SUPPLIER, PARTS.PROD#, PRODUCT
  FROM PARTS, PRODUCTS
 WHERE PARTS.PROD# = PRODUCTS.PROD#;

```

or

```

SELECT PART, SUPPLIER, PARTS.PROD#, PRODUCT
  FROM PARTS INNER JOIN PRODUCTS
    ON PARTS.PROD# = PRODUCTS.PROD#;

```

Either one of these two statements give this result:

subselect

PART	SUPPLIER	PROD#	PRODUCT
WIRE	ACWF	10	GENERATOR
MAGNETS	BATEMAN	10	GENERATOR
PLASTIC	PLASTIK_Corp	30	RELAY
BLADES	ACE_STEEL	205	SAW

Notice two things about the example:

- There is a part in the parts table (OIL) whose product (#160) is not listed in the products table. There is a product (SCREWDRIVER, #505) that has no parts listed in the parts table. Neither OIL nor SCREWDRIVER appears in the result of the join.
- An *outer join*, however, includes rows where the values in the joined columns do not match.
- There is explicit syntax to express that this familiar join is not an outer join but an inner join. You can use INNER JOIN in the FROM clause instead of the comma. Use ON when you explicitly join tables in the FROM clause.

You can specify more complicated join conditions to obtain different sets of results. For example, eliminate the suppliers that begin with the letter A from the table of parts, suppliers, product numbers and products:

```
SELECT PART, SUPPLIER, PARTS.PROD#, PRODUCT
      FROM PARTS INNER JOIN PRODUCTS
            ON PARTS.PROD# = PRODUCTS.PROD#
            AND SUPPLIER NOT LIKE 'A%';
```

The result of the query is all rows that do not have a supplier that begins with A:

PART	SUPPLIER	PROD#	PRODUCT
MAGNETS	BATEMAN	10	GENERATOR
PLASTIC	PLASTIK_Corp	30	RELAY

Example 11: Join the tables on the PROD# column to get a table of all parts and products, showing the supplier information, if any.

```
SELECT PART, SUPPLIER, PARTS.PROD#, PRODUCT
      FROM PARTS FULL OUTER JOIN PRODUCTS
            ON PARTS.PROD# = PRODUCTS.PROD#;
```

The result is:

PART	SUPPLIER	PROD#	PRODUCT
WIRE	ACWF	10	GENERATOR
MAGNETS	BATEMAN	10	GENERATOR
PLASTIC	PLASTIK_Corp	30	RELAY
BLADES	ACE_STEEL	205	SAW
OIL	WESTERN_CHEM	160	(null)
(null)	(null)	(null)	SCREWDRIVER

The clause FULL OUTER JOIN includes unmatched rows from both tables. Missing values in a row of the result table are filled with nulls.

Example 12: Join the tables on the PROD# column to get a table of all parts, showing what products, if any, the parts are used in:

```
SELECT PART, SUPPLIER, PARTS.PROD#, PRODUCT
      FROM PARTS LEFT OUTER JOIN PRODUCTS
            ON PARTS.PROD# = PRODUCTS.PROD#;
```

The result is:

PART	SUPPLIER	PROD#	PRODUCT
WIRE	ACWF	10	GENERATOR
MAGNETS	BATEMAN	10	GENERATOR
PLASTIC	PLASTIK_Corp	30	RELAY
BLADES	ACE_STEEL	205	SAW
OIL	WESTERN_CHEM	160	(null)

The clause LEFT OUTER JOIN includes rows from the table identified before it where the values in the joined columns are not matched by values in the joined columns of the table identified after it.

Example 13: Join the tables on the PROD# column to get a table of all products, showing the parts used in that product, if any, and the supplier.

```
SELECT PART, SUPPLIER, PRODUCTS.PROD#, PRODUCT
  FROM PARTS RIGHT OUTER JOIN PRODUCTS
    ON PARTS.PROD# = PRODUCTS.PROD#;
```

The result is:

PART	SUPPLIER	PROD#	PRODUCT
WIRE	ACWF	10	GENERATOR
MAGNETS	BATEMAN	10	GENERATOR
PLASTIC	PLASTIK_Corp	30	RELAY
BLADES	ACE_STEEL	205	SAW
(null)	(null)	505	SCREWDRIVER

The clause RIGHT OUTER JOIN includes rows from the table identified after it where the values in the joined columns are not matched by values in the joined columns of the table identified before it.

Example 14: The result of Example 11 (a full outer join) shows the product number for SCREWDRIVER as null, even though the PRODUCTS table contains a product number for it. This is because PRODUCTS.PROD# was not listed in the SELECT list of the query. Revise the query using COALESCE, a synonym for the VALUE function, so that all part numbers from both tables are shown.

```
SELECT PART, SUPPLIER,
      COALESCE(PARTS.PROD#, PRODUCTS.PROD#) AS PRODNUM, PRODUCT
        FROM PARTS FULL OUTER JOIN PRODUCTS
          ON PARTS.PROD# = PRODUCTS.PROD#;
```

In the result, notice that the AS clause (AS PRODNUM), provides a name for the result of the COALESCE function:

PART	SUPPLIER	PRODNUM	PRODUCT
WIRE	ACWF	10	GENERATOR
MAGNETS	BATEMAN	10	GENERATOR
PLASTIC	PLASTIK_Corp	30	RELAY
BLADES	ACE_STEEL	205	SAW
OIL	WESTERN_CHEM	160	(null)
(null)	(null)	505	SCREWDRIVER

Example 15: For all parts that are used in product numbers less than 200, show the part, the part supplier, the product number, and the product name. Use a nested table expression.

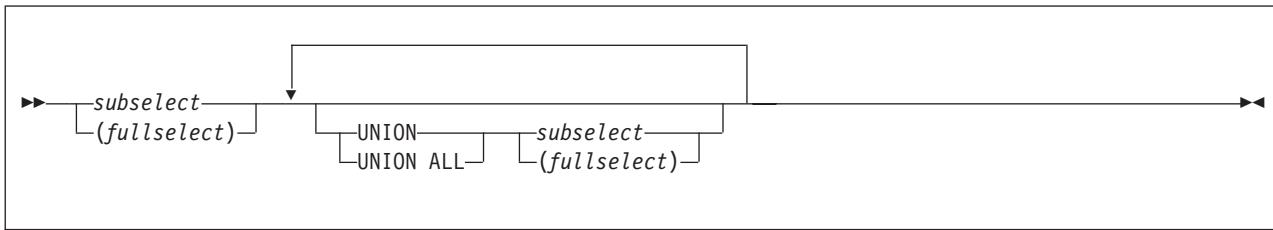
subselect

```
SELECT PART, SUPPLIER, PRODNUM, PRODUCT
  FROM (SELECT PART, PROD# AS PRODNUM, SUPPLIER
         FROM PARTS
        WHERE PROD# < 200) AS PARTX
  LEFT OUTER JOIN PRODUCTS
    ON PRODNUM = PROD#;
```

The result is:

PART	SUPPLIER	PRODNUM	PRODUCT
=====	=====	=====	=====
WIRE	ACWF	10	GENERATOR
MAGNETS	BATEMAN	10	GENERATOR
PLASTIC	PLASTIK_CORP	30	RELAY
OIL	WESTERN_CHEM	160	(null)

fullselect



The *fullselect* is a component of the *select-statement*, the CREATE VIEW statement, and the INSERT statement. The *fullselect* is also a component of certain predicates that, in turn, are components of a *subselect*. A subselect that is a component of a predicate is called a *subquery*.

A *fullselect* specifies a result table. If UNION is not used, the result of the fullselect is the result of the specified subselect.

UNION or UNION ALL

Derives a result table by combining two other result tables, R1 and R2. If UNION ALL is specified, the result consists of all rows in R1 and R2. If UNION is specified without the ALL option, the result is the set of all rows in either R1 or R2, with duplicate rows eliminated.

If the *n*th column of R1 and the *n*th column of R2 have the same result column name, the *n*th column of R has the same result column name. If the *n*th column of R1 and the *n*th column of R2 do not have the same name, the result column in R is unnamed.

Qualified column names cannot be used in the ORDER BY clause when UNION or UNION ALL is also specified.

Duplicate rows: Two rows are duplicates if each value in the first is equal to the corresponding value of the second. For determining duplicates, two null values are considered equal.

UNION and UNION ALL are associative operations. However, when UNION and UNION ALL are used in the same statement, the result depends on the order in which the operations are performed. Operations within parentheses are performed first. When the order is not specified by parentheses, operations are performed in order from left to right.

Rules for columns: R1 and R2 must have the same number of columns and the data type of the *n*th column of R1 must be compatible with the data type of the *n*th column of R2. If UNION is specified without the ALL option, R1 and R2 must not include a long string column.

The *n*th column of the result of UNION and UNION ALL is derived from the *n*th columns of R1 and R2.

For information on the valid combinations of operand columns and the data type of the result column, see “Rules for result data types” on page 77.

Character conversion in unions and concatenations

The SQL operations that combine strings include concatenation, UNION, UNION ALL, and the IN list of an IN predicate. Within an SQL statement, concatenation combines two or more strings into a new string. Within a fullselect, UNION, UNION ALL, or the IN list of an IN predicate combine two or more string columns resulting from the subselects into results column. All such operations have the following in common:

- The choice of a result CCSID for the string or column
- The possible conversion of one or more of the component strings or columns to the result CCSID

For all such operations, the rules for those two actions are the same, as described in “Selecting the result CCSID.” These rules also apply to the COALESCE (or VALUE) scalar function.

Selecting the result CCSID

The result CCSID is selected at bind time. The result CCSID is the CCSID of one of the operands.

Two operands: When two operands are used, the result CCSID is determined by the operand types, their CCSIDs, and their relative positions in the operation. The rules shown here apply when neither CCSID is X'FFFF'. When a CCSID is X'FFFF', the result CCSID is always X'FFFF', and no character conversions take place.

If one CCSID is for SBCS data and the other is for mixed data, the operand selected depends on the value of the MIXED DATA field on installation panel DSNTIPF at the DB2 where the operation takes place:

- If this value is YES, the operand MIXED furnishes the result CCSID.
- If this value is NO, the operand SBCS furnishes the result CCSID.

Unicode data can be mixed regardless of the setting of the MIXED field on installation panel DSNTIPF.

If both CCSIDs are the same type (both SBCS, both MIXED, or both GRAPHIC CCSIDs), then the operand that furnishes the result CCSID is as shown in Table 38.

For example, assume a concatenation of the form:

string-constant CONCAT derived-value

The value in the second row and fourth column shows that the first operand (*string-constant*) supplies the result CCSID.

Table 38. Operand that supplies the CCSID for character conversion

First operand	Second operand				
	Column value	String constant	Special register	Derived value	Host variable
Column value	first	first	first	first	first
String constant	second	first	first	first	first
Special register	second	first	first	first	first
Derived value	second	second	second	first	first
Host variable	second	second	second	second	neither ¹

Note: 1. Both operands are converted, if necessary, to the system CCSID of the server.

Three or more operands:

If all the operands have the same CCSID, the result CCSID is the common CCSID.

If at least one of the CCSIDs has the value X'FFFF', the result CCSID also has the value X'FFFF'.

Otherwise, selection proceeds as follows:

1. The rules for a pair of operands are applied to the first two operands. This picks a “candidate” for the second step. The candidate is the operand that would furnish the result CCSID if just the first two operands were involved in the operation.
2. The rules are applied to the Step 1 candidate and the third operand, thereby selecting a second candidate.
3. If a fourth operand is involved, the rules are applied to the second candidate and fourth operand, to select a third candidate, and so on.

The process continues until all operands have been used. The remaining candidate is the one that furnishes the result CCSID. Whenever the rules for a pair are applied to a candidate and an operand, the candidate is considered to be the first operand.

Consider, for example, the following concatenation:

A CONCAT B CONCAT C

Here, the rules are first applied to the strings A and B. Suppose that the string selected as candidate is A. Then the rules are applied to A and C. If the string selected is again A, then A furnishes the result CCSID. Otherwise, C furnishes the result CCSID.

Character conversion of components: An operand of concatenation or the selected argument of the COALESCE (or VALUE) scalar function is converted, if necessary, to the coded character set of the result string. Each string of an operand of UNION or UNION ALL is converted, if necessary, to the coded character set of the result column. In either case, the coded character set is the one identified by the result CCSID. Character conversion is necessary only if all of the following are true:

- The result and operand CCSIDs are different.
- Neither CCSID is X'FFFF' (neither string is defined as BIT data).
- The string is neither null nor empty.
- The SYSSTRINGS catalog table indicates that conversion is necessary.

An error occurs if a character of a string cannot be converted, SYSSTRINGS is used but contains no information about the CCSID pair, or DB2 cannot do the conversion through DB2 for OS/390 and z/OS support for Unicode or Language Environment. A warning occurs if a character of a string is converted to the substitution character.

Examples of fullselects

Example 1: A query specifies the union of result tables R1 and R2. A column in R1 has the data type CHAR(10) and the subtype BIT. The corresponding column in R2

fullselect

has the data type CHAR(15) and the subtype SBCS. Hence, the column in the union has the data type CHAR(15) and the subtype BIT. Values from the first column are converted to CHAR(15) by adding five trailing blanks.

Example 2: Show all the rows from DSN8710.EMP.

```
SELECT * FROM DSN8710.EMP;
```

Example 3: Using sample tables DSN8710.EMP and DSN8710.EMPPROJECT, list the employee numbers of all employees for which either of the following statements are true:

- Their department numbers begin with 'D'.
- They are assigned to projects whose project numbers begin with 'AD'.

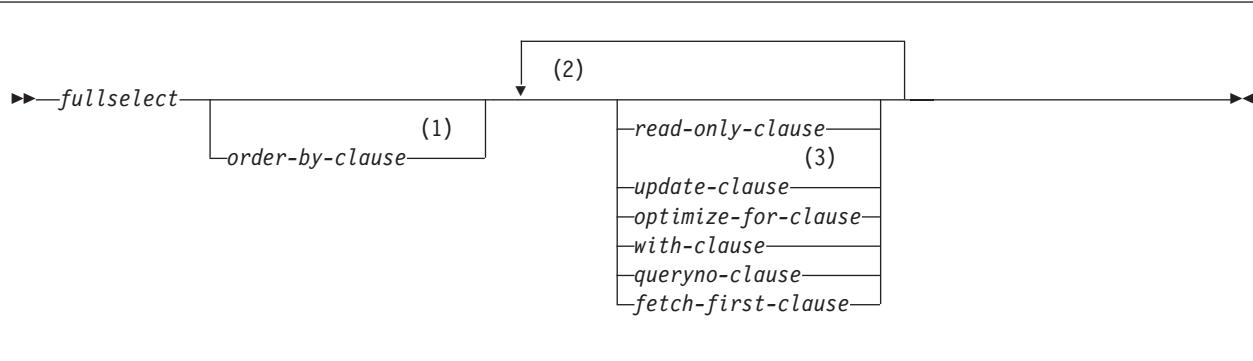
```
SELECT EMPNO FROM DSN8710.EMP
  WHERE WORKDEPT LIKE 'D%'
UNION
SELECT EMPNO FROM DSN8710.EMPPROJECT
  WHERE PROJNO LIKE 'AD%';
```

The result is the union of two result tables, one formed from the sample table DSN8710.EMP, the other formed from the sample table DSN8710.EMPPROJECT. The result—a one-column table—is a list of employee numbers. Because UNION, rather than UNION ALL, was used, the entries in the list are distinct. If instead UNION ALL were used, certain employee numbers would appear in the list more than once. These would be the numbers for employees in departments that begin with 'D' while their projects begin with 'AD'.

Example 4: Find the average charges for each subscriber (SNO) in the state of California during the last Friday of each month in the first quarter of 2000. Group the result according to SNO. Each MONTHnn table has columns for SNO, CHARGES, and DATE. The CUST table has columns for SNO and STATE.

```
SELECT V.SNO, AVG(V.CHARGES)
  FROM CUST, TABLE (
    SELECT SNO, CHARGES, DATE
      FROM MONTH1
     WHERE DATE BETWEEN '01/01/2000' AND '01/31/2000'
UNION ALL
    SELECT SNO, CHARGES, DATE
      FROM MONTH2
     WHERE DATE BETWEEN '02/01/2000' AND '02/29/00'
UNION ALL
    SELECT SNO, CHARGES, DATE
      FROM MONTH3
     WHERE DATE BETWEEN '03/01/2000' AND '03/31/2000'
  ) AS V(SNO, CHARGES, DATE)
 WHERE CUST.SNO=V.SNO
   AND CUST.STATE='CA'
   AND DATE IN ('01/28/2000','02/25/2000','03/31/2000')
 GROUP BY V.SNO;
```

select-statement



Notes:

- 1 If the *order-by-clause* is specified, the *update-clause* cannot be specified except when the *select-statement* is associated with an INSENSITIVE or SENSITIVE STATIC SCROLL CURSOR.
- 2 The same clause must not be specified more than once.
- 3 If the *update-clause* is specified, the *fetch-first-clause* cannot be specified.

The *select-statement* is the form of a query that can be directly specified in a DECLARE CURSOR statement, or prepared and then referenced in a DECLARE CURSOR statement. It can also be issued interactively using SPUFI causing a result table to be displayed at your terminal. In any case, the table specified by *select-statement* is the result of the fullselect.

The tables and view identified in a select statement can be at the current server or any DB2 subsystem with which the current server can establish a connection.

For local queries on DB2 for OS/390 and z/OS or remote queries in which the server and requester are DB2 for OS/390 and z/OS, if a table is encoded as ASCII or Unicode, the retrieved data is encoded in EBCDIC. For information on retrieving data encoded in ASCII or Unicode, see Part 6 of *DB2 Application Programming and SQL Guide*.

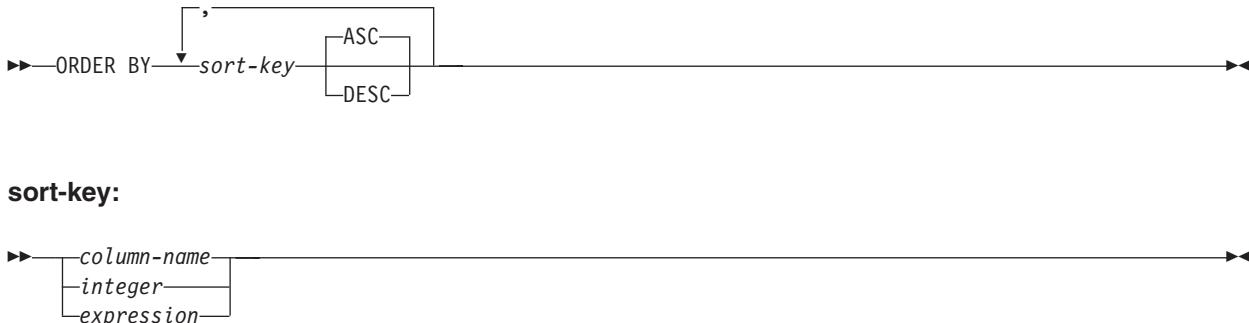
A select statement can implicitly or explicitly invoke user-defined functions or implicitly invoke stored procedures. This technique is known as *nesting* of SQL statements. A function or procedure is implicitly invoked in a select statement when it is invoked at a lower level. For instance, if you invoke a user-defined function from a select statement and the user-defined function invokes a stored procedure, you are implicitly invoking the stored procedure. When you execute a select statement on a table, no INSERT, UPDATE, or DELETE statement at a lower level of nesting must be executed on the same table.

For example, suppose that you execute this SQL statement at level 1 of nesting:
 SELECT UDF1(C1) FROM T1;

You cannot execute this SQL statement at a lower level of nesting:
 INSERT INTO T1 VALUES(...);

select-statement

order-by-clause



The ORDER BY clause specifies an ordering of the rows of the result table. If a single *sort-key* is identified, the rows are ordered by the values of that *sort-key*. If more than one *sort-key* is identified, the rows are ordered by the values of the first *sort-key*, then by the values of the second *sort-key*, and so on. A *sort-key* cannot be a long string column.

A named column in the select list can be identified by a *sort-key* that is an integer or a column name. An unnamed column in the select list must be identified by an integer or by an expression. A column is unnamed if the AS clause is not specified in the select list and the column is derived from a constant, an expression with operators, or a function. If the fullselect includes a UNION operator, the fullselect rules on named columns apply.

column-name

Usually identifies a column of the result table. In this case, *column-name* must be the name of a named column in the select list. If the *fullselect* includes a UNION or UNION ALL, the column name cannot be qualified.

If the query is a *subselect*, the *column-name* can also identify a column name of a table, view, or nested table expression identified in the FROM clause. An error occurs if the subselect includes any of the following:

- DISTINCT in the select list
- Column functions in the select list
- GROUP BY clause

integer

Must be greater than 0 and not greater than the number of columns in the result table. The integer *n* identifies the *n*th column of the result table.

expression

Specifies an expression with operators (that is, not simply a *column-name* or *integer*). The query to which ordering is applied must be a *subselect* to use this form of the *sort-key*.

The *expression* cannot include a non-deterministic function or a function with an external action. Any column name in the expression must conform to the rules described in “Column names in sort keys” on page 371. An expression cannot be specified if DISTINCT is used in the select list of the *subselect*.

If the *subselect* is grouped, the *expression* can be an expression in the select list of the subselect or can include a column function, constant, or host variable. If the expression is not in the select list, the following rules apply:

- If the subselect contains a GROUP BY clause, all columns in the *expression* must be in the GROUP BY clause.
- If the subselect does not contain a GROUP BY clause (the result is grouped because a column function is in the ORDER BY clause), all expressions in the select list must be grouped data, constants, or host variables.

ASC

Uses the values of the *sort-key* in ascending order. This is the default.

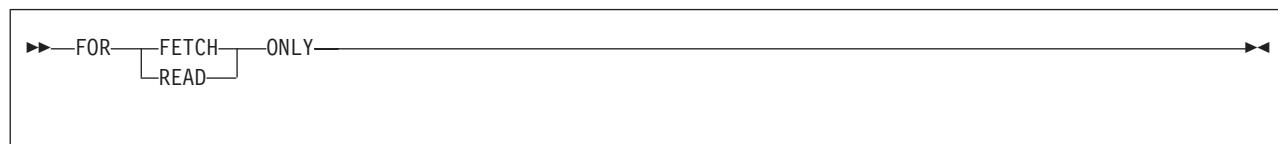
DESC

Uses the values of the *sort-key* in descending order.

Ordering is performed in accordance with the comparison rules described in Chapter 2, “Language elements,” beginning on page 72. The null value is higher than all other values. If your ordering specification does not determine a complete ordering, rows with duplicate values of the last identified *sort-key*s have an arbitrary order. If you do not specify ORDER BY, the rows of the result table have an arbitrary order.

Column names in sort keys: A column name in a *sort-key* must conform to the following rules:

- If the column name was also specified in the SELECT list with an AS clause, the name specified after the AS clause must not be referenced in the sort-key. Instead, the name of the column in the database should be used in the sort-key.
- If the column name is qualified, the query must be a *subselect*. The column name must unambiguously identify a column of a table, view, or nested table expression in the FROM clause of the subselect; its value is used to compute the value of the sort specification.
- If the column name is unqualified, the following two cases apply:
 - The query is a *subselect*. In this case, the column name must unambiguously identify a column of a table, view, or nested table expression in the FROM clause of the subselect. If the column name is identical to one column of the result table, its value is used to compute the value of the sort specification. If the column name is not identical to one column, it must unambiguously identify a column of a table, view, or nested table expression in the FROM clause of the fullselect.
 - The query is *not a subselect* (that is, the query includes a UNION or UNION ALL). The column name must be identical to exactly one column of the result table; its value is used to compute the value of the sort specification.

read-only-clause

The clause FOR FETCH ONLY²⁵ declares that the result table is read-only.

Some result tables are read-only by nature (for example, a table based on a read-only view.) FOR FETCH ONLY can still be specified for such tables, but the specification has no effect. For result tables for which updates and deletes are

25. Or, FOR READ ONLY is equivalent.

select-statement

possible, specifying FOR FETCH ONLY can possibly improve the performance of FETCH operations and distributed operations.

If factors other than specifying the FOR READ ONLY clause make the result table read-only, positioned updates or deletes cannot be done. A read-only result table must not be referred to in an UPDATE or DELETE statement, whether it is read-only by nature or specified as FOR FETCH ONLY.

update-clause



The optional FOR UPDATE clause identifies the columns that can be updated in a later positioned UPDATE statement. Each column name must be unqualified and must identify a column of the table or view identified in the first FROM clause of the fullselect. The clause must not be specified if the result table of the fullselect is read-only. For a discussion of read-only result tables, see "DECLARE CURSOR" on page 718. The clause must also not be specified if a created temporary table is referenced in the first FROM clause of the select-statement.

If the FOR UPDATE clause is specified without a list of columns, the columns that can be updated will include all the updatable columns of the table or view that is identified in the first FROM clause of the fullselect.

The declaration of a cursor referred to in a positioned UPDATE statement need not include a FOR UPDATE clause if the STDSQL(YES) or NOFOR option is specified when the program is precompiled. For more on the subject, see "Positioned updates of columns" on page 153.

When FOR UPDATE is used, FETCH operations referencing the cursor acquire U or X locks rather than S locks when:

- The isolation level of the statement is cursor stability.
- The isolation level of the statement is repeatable read or read stability and field U LOCK FOR RR/RS on installation panel DSNTIPI is set to get U locks.
- The isolation level of the statement is repeatable read or read stability and KEEP UPDATE LOCKS is specified in the SQL statement, an X lock, instead of a U lock, is acquired at FETCH time.

No locks are acquired on declared temporary tables. For a discussion of U locks and S locks, see Part 5 (Volume 2) of *DB2 Administration Guide*.

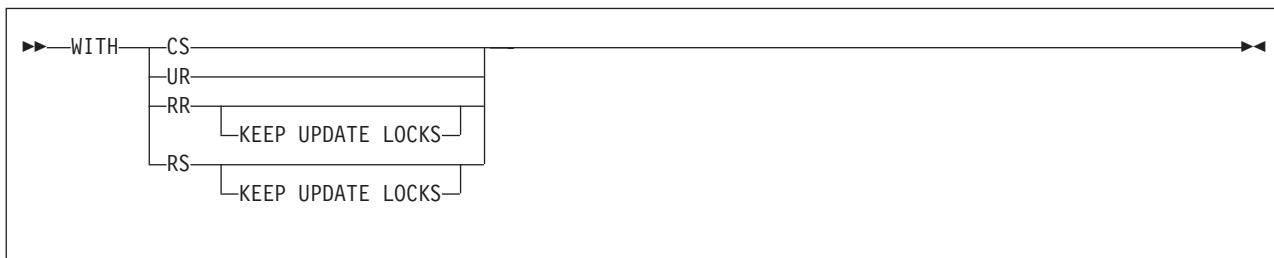
optimize-for-clause



The OPTIMIZE FOR clause requests special optimization of the *select-statement*. The clause specifies that optimization is based on the assumption that the number of rows retrieved will not exceed n , where n is the value of the integer. If the clause is omitted, optimization is based on the assumption that all rows of the result table will be retrieved unless the FETCH FIRST clause is specified. If the clause is omitted and FETCH FIRST is specified, OPTIMIZE FOR m ROWS is assumed, where m is the FETCH FIRST value.

The OPTIMIZE FOR clause does not limit the number of rows that can be fetched or affect the result in any way other than performance. In general, if you are retrieving only a few rows, use OPTIMIZE FOR 1 ROW to influence the access path that DB2 selects. For more information about using this clause, see *DB2 Application Programming and SQL Guide*.

with-clause



The WITH clause specifies the isolation level at which the statement is executed. (Isolation level does not apply to declared temporary tables because no locks are acquired.)

CS Cursor stability

UR Uncommitted read

RR Repeatable read

RR KEEP UPDATE LOCKS

Repeatable read keep update locks

RS Read stability

RS KEEP UPDATE LOCKS

Read stability keep update locks

WITH UR can be specified only if the result table is read-only.

WITH RR KEEP UPDATE LOCKS or WITH RS KEEP UPDATE LOCKS can be specified only if the FOR UPDATE clause is also specified. KEEP UPDATE LOCKS tells DB2 to acquire and hold an X lock instead of an U or S lock on all qualified pages and rows. Although this option can reduce concurrency, it can prevent some types of deadlocks.

The **default** isolation level of the statement depends on:

- The isolation of the package or plan that the statement is bound in
- Whether the result table is read-only

If package isolation is:	And plan isolation is:	And the result table is:	Then the default isolation is:
RR	Any	Any	RR
RS	Any	Any	RS
CS	Any	Any	CS

select-statement

If package isolation is:	And plan isolation is:	And the result table is:	Then the default isolation is:
UR	Any	Read-only	UR
		Not read-only	CS
Not specified	Not specified	Any	RR
	RR	Any	RR
	RS	Any	RS
	CS	Any	CS
	UR	Read-only	UR
		Not read-only	CS

See “Notes” on page 721 for a list of the characteristics that make a result table read-only. A simple way to ensure that a result table is read-only is to specify FOR FETCH ONLY or FOR READ ONLY in the SQL statement.

queryno-clause

```
►►QUERYNO—integer►►
```

The QUERYNO clause specifies the number to be used for this SQL statement in EXPLAIN output and trace records. The number is used for the QUERYNO columns of the plan tables for the rows that contain information about this SQL statement. This number is also used in the QUERYNO column of the SYSIBM.SYSSTMT and SYSIBM.SYSPACKSTMT catalog tables.

If the clause is omitted, the number associated with the SQL statement is the statement number assigned during precompilation. Thus, if the application program is changed and then precompiled, that statement number might change.

Using the QUERYNO clause to assign unique numbers to the SQL statements in a program is helpful:

- For simplifying the use of optimization hints for access path selection
- For correlating SQL statement text with EXPLAIN output in the plan table

For information on using optimization hints, such as enabling the system for optimization hints and setting valid hint values, and for information on accessing the plan table, see Part 5 (Volume 2) of *DB2 Administration Guide*.

fetch-first-clause

```
►►FETCH FIRST [integer] ROWS ONLY►►
```

The FETCH FIRST clause limits the number of rows that can be fetched. It improves the performance of queries with potentially large result sets when only a

limited number of rows are needed. If the clause is specified, the number of rows retrieved will not exceed n , where n is the value of the integer. An attempt to fetch $n+1$ rows is handled the same way as normal end of data. The value of *integer* must be positive and non-zero. The default is 1.

If both the FETCH FIRST clause and the ORDER BY clause are specified, the ordering is performed on the entire result set prior to returning the first n rows.

If the OPTIMIZE FOR clause is not specified, a default of OPTIMIZE FOR m ROWS, where m is the value of FETCH FIRST, is assumed. DB2 uses this value for access path optimization.

Examples of select statements

Example 1: Select all the rows from DSN8710.EMP.

```
SELECT * FROM DSN8710.EMP;
```

Example 2: Select all the rows from DSN8710.EMP, arranging the result table in chronological order by date of hiring.

```
SELECT * FROM DSN8710.EMP ORDER BY HIREDATE;
```

Example 3: Select the department number (WORKDEPT) and average departmental salary (SALARY) for all departments in the table DSN8710.EMP. Arrange the result table in ascending order by average departmental salary.

```
SELECT WORKDEPT, AVG(SALARY)
  FROM DSN8710.EMP
 GROUP BY WORKDEPT
 ORDER BY 2;
```

Example 4: Change various salaries, bonuses, and commissions in the table DSN8710.EMP. Confine the changes to employees in departments D11 and D21. Use positioned updates to do this with a cursor named UP_CUR. Use a FOR UPDATE clause in the cursor declaration to indicate that all updatable columns are updated. Below is the declaration for a PL/I program.

```
EXEC SQL DECLARE UP_CUR CURSOR FOR
      SELECT WORKDEPT, EMPNO, SALARY, BONUS, COMM
        FROM DSN8710.EMP
       WHERE WORKDEPT IN ('D11','D21')
          FOR UPDATE;
```

Beginning where the cursor is declared, all updatable columns would be updated. If only specific columns needed to be updated, such as only the salary column, the FOR UPDATE clause could be used to specify the salary column (FOR UPDATE OF SALARY).

Example 5: Find the maximum, minimum, and average bonus in the table DSN8710.EMP. Execute the statement with uncommitted read isolation, regardless of the value of ISOLATION with which the plan or package containing the statement is bound. Assign 13 as the query number for the SELECT statement.

```
EXEC SQL
  SELECT MAX(BONUS), MIN(BONUS), AVG(BONUS)
    INTO :MAX, :MIN, :AVG
    FROM DSN8710.EMP
   WITH UR
  QUERYNO 13;
```

If bind option EXPLAIN(YES) is specified, rows are inserted into the plan table. The value used for the QUERYNO column for these rows is 13.

select-statement

Example 6: The cursor declaration shown below is in a PL/I program. In the query within the declaration, X.RMT_TAB is an alias for a table at some other DB2. Hence, when the query is used, it is processed using DRDA access. See “Distributed data” on page 14.

The declaration indicates that no positioned updates or deletes will be done with the query’s cursor. It also specifies that the access path for the query be optimized for the retrieval of at most 50 rows. Even so, the program can retrieve more than 50 rows from the result table, which consists of the entire table identified by the alias. However, when more than 50 rows are retrieved, performance could possibly degrade.

```
EXEC SQL DECLARE C1 CURSOR FOR
  SELECT * FROM X.RMT_TAB
  OPTIMIZE FOR 50 ROWS
  FOR FETCH ONLY;
```

The `FETCH FIRST` clause could be used instead of the `OPTIMIZE FOR` clause to ensure that only 50 rows are retrieved as in the following example:

```
EXEC SQL DECLARE C1 CURSOR FOR
  SELECT * FROM X.RMT_TAB
  FETCH FIRST 50 ROWS Only;
```

Chapter 5. Statements

This chapter contains syntax diagrams, semantic descriptions, rules, and examples of the use of the SQL statements listed in the following table.

Table 39. SQL statements

SQL statement	Function	Page
ALLOCATE CURSOR	Defines and associates a cursor with a result set locator variable	386
ALTER DATABASE	Changes the description of a database	388
ALTER FUNCTION (external scalar)	Changes the description of a user-defined external scalar or table function	391
ALTER FUNCTION (SQL scalar)	Changes the description of an SQL scalar function	408
ALTER INDEX	Changes the description of an index	414
ALTER PROCEDURE (external)	Changes the description of a stored procedure	427
ALTER PROCEDURE (SQL))	Changes the description of an SQL procedure	438
ALTER STOGROUP	Changes the description of a storage group	444
ALTER TABLE	Changes the description of a table	447
ALTER TABLESPACE	Changes the description of a table space	467
ASSOCIATE LOCATORS	Gets the result set locator value for each result set returned by a stored procedure	479
BEGIN DECLARE SECTION	Marks the beginning of a host variable declaration section	482
CALL	Calls a stored procedure	483
CLOSE	Closes a cursor	491
COMMENT	Replaces or adds a comment to the description of an object	493
COMMIT	Ends a unit of recovery and commits the database changes made by that unit of recovery	499
CONNECT (Type 1)	Connects the process to a server	504
CONNECT (Type 2)	Connects the process to a server	510
CREATE ALIAS	Defines an alias	514
CREATE AUXILIARY TABLE	Defines an auxiliary table for storing LOB data	514
CREATE DATABASE	Defines a database	519
CREATE DISTINCT TYPE	Defines a distinct type (user-defined data type)	522
CREATE FUNCTION (external scalar)	Defines a user-defined external scalar function	530
CREATE FUNCTION (external table)	Defines a user-defined external table function	553
CREATE FUNCTION (sourced)	Defines a user-defined function that is based on an existing scalar or column function	571
CREATE FUNCTION (SQL scalar)	Defines a user-defined SQL scalar function	585
CREATE GLOBAL TEMPORARY TABLE	Defines a created temporary table at the current server	595
CREATE INDEX	Defines an index on a table	601

Statements

Table 39. SQL statements (continued)

SQL statement	Function	Page
CREATE PROCEDURE (external)	Defines an external stored procedure	617
CREATE PROCEDURE (SQL)	Defines an SQL procedure	636
CREATE STOGROUP	Defines a storage group	648
CREATE SYNONYM	Defines an alternate name for a table or view	651
CREATE TABLE	Defines a table	653
CREATE TABLESPACE	Defines a table space, which includes allocating and formatting the table space	681
CREATE TRIGGER	Defines a trigger	699
CREATE VIEW	Defines a view of one or more tables or views	711
DECLARE CURSOR	Defines an SQL cursor	718
DECLARE GLOBAL TEMPORARY TABLE	Defines a declared temporary table at the current server	725
DECLARE STATEMENT	Declares names used to identify prepared SQL statements	735
DECLARE TABLE	Provides the programmer and the precompiler with a description of a table or view	736
I DECLARE VARIABLE	Defines a CCSID for a host variable	739
DELETE	Deletes one or more rows from a table	742
DESCRIBE (prepared statement or TABLE)	Describes the result columns of a prepared statement or the columns of a table or view	749
DESCRIBE CURSOR	Puts information about the result set associated with a cursor into a descriptor	756
DESCRIBE INPUT	Puts information about the input parameters (markers) of a prepared statement into a descriptor	758
DESCRIBE PROCEDURE	Puts information about the result sets returned by a stored procedure into a descriptor	760
DROP	Deletes objects	763
END DECLARE SECTION	Marks the end of a host variable declaration section	775
EXECUTE	Executes a prepared SQL statement	776
EXECUTE IMMEDIATE	Prepares and executes an SQL statement	779
EXPLAIN	Obtains information about how an SQL statement would be executed	781
I FETCH	Positions the cursor (BEFORE/AFTER), returns data (CURRENT), or both positions the cursor and returns data	793
FREE LOCATOR	Removes the association between a LOB locator variable and its value	802
GRANT (collection privileges)	Grants authority to create a package in a collection	806
GRANT (database privileges)	Grants privileges on a database	807
I GRANT (distinct type or JAR privileges)	Grants the usage privilege on a distinct type (user-defined data type) or JARs	809
GRANT (function or procedure privileges)	Grants privileges on a user-defined function or a stored procedure	811
GRANT (package privileges)	Grants authority to bind, execute, or copy a package	816

Table 39. SQL statements (continued)

SQL statement	Function	Page
GRANT (plan privileges)	Grants authority to bind or execute an application plan	818
GRANT (schema privileges)	Grants privileges on a schema	819
GRANT (system privileges)	Grants system privileges	819
GRANT (table or view privileges)	Grants privileges on a table or view	824
GRANT (use privileges)	Grants authority to use specified buffer pools, storage groups, or table spaces	827
HOLD LOCATOR	Allows a LOB locator variable to retain its association with its value beyond a unit of work	829
INCLUDE	Inserts declarations into a source program	830
INSERT	Inserts one or more rows into a table	832
LABEL ON	Replaces or adds a label on the description of a table, view, alias, or column	838
LOCK TABLE	Locks a table or table space partition in shared or exclusive mode	840
OPEN	Opens a cursor	842
PREPARE	Prepares an SQL statement (with optional parameters) for execution	846
RELEASE (connection)	Places one or more connections in the release pending status	859
RELEASE SAVEPOINT	Releases a savepoint and any subsequently set savepoints within a unit of recovery	862
RENAME TABLE	Renames an existing table	863
REVOKE (collection privileges)	Revokes authority to create a package in a collection	870
REVOKE (database privileges)	Revokes privileges on a database	871
REVOKE (distinct type or JAR privileges)	Revokes the usage privilege on a distinct type (user-defined data type) or JARs	874
REVOKE (Function or Procedure privileges)	Revokes privileges on a user-defined function or a stored procedure	876
REVOKE (package privileges)	Revokes authority to bind, execute, or copy a package	881
REVOKE (plan privileges)	Revokes authority to bind or execute an application plan	883
REVOKE (schema privileges)	Revokes privileges on a schema	884
REVOKE (system privileges)	Revokes system privileges	886
REVOKE (table or view privileges)	Revokes privileges on a table or view	889
REVOKE (use privileges)	Revokes authority to use specified buffer pools, storage groups, or table spaces	892
ROLLBACK	Ends a unit of recovery and backs out the changes to the database made by that unit of recovery, or partially rolls back the changes to a savepoint within the unit of recovery	894
SAVEPOINT	Sets a savepoint within a unit of recovery	897
SELECT	Specifies the SELECT statement of the cursor	899
SELECT INTO	Specifies a result table of no more than one row and assigns the values to host variables	900

Statements

Table 39. SQL statements (continued)

SQL statement	Function	Page
SET CONNECTION	Establishes the database server of the process by identifying one of its existing connections	904
SET CURRENT APPLICATION ENCODING SCHEME	Assigns a value to the CURRENT APPLICATION ENCODING SCHEME special register	906
SCHEME		
SET CURRENT DEGREE	Assigns a value to the CURRENT DEGREE special register	907
SET CURRENT LOCALE LC_CTYPE	Assigns a value to the CURRENT LOCALE LC_CTYPE special register	909
SET CURRENT OPTIMIZATION HINT	Assigns a value to the CURRENT OPTIMIZATION HINT special register	911
SET CURRENT PACKAGESET	Assigns a value to the CURRENT PACKAGESET special register	912
SET CURRENT PRECISION	Assigns a value to the CURRENT PRECISION special register	914
SET CURRENT RULES	Assigns a value to the CURRENT RULES special register	915
SET CURRENT SQLID	Assigns a value to the CURRENT SQLID special register	916
SET host-variable Assignment	Assigns values to host variables	918
SET PATH	Assigns a value to the CURRENT PATH special register	921
SET transition-variable Assignment	Assigns values to transition variables	924
SIGNAL SQLSTATE	Signals an error with a user-specified SQLSTATE and description	927
UPDATE	Updates the values of one or more columns in one or more rows of a table	928
VALUES	Provides a method to invoke a user-defined function from a trigger	938
VALUES INTO	Assigns values to host variables	939
WHENEVER	Defines actions to be taken on the basis of SQL return codes	941

How SQL statements are invoked

The SQL statements described in this chapter are classified as *executable* or *nonexecutable*. The section on invocation in the description of each statement indicates whether or not the statement is executable.

Executable statements can be invoked in the following ways:

- Embedded in an application program
- Dynamically prepared and executed
- Dynamically prepared and executed using DB2 ODBC function calls
- Issued interactively

Depending on the statement, you can use some or all of these methods. The section on invocation in the description of each statement tells you which methods can be used. See Appendix B, “Characteristics of SQL statements in DB2 for OS/390 and z/OS,” on page 969 for a list of executable statements.

A *nonexecutable statement* can only be embedded in an application program.

In addition to the statements described in this chapter, there is one more SQL statement construct: the *select-statement*. (See “select-statement” on page 369.) It is not included in this chapter because it is used in a different way from other statements.

A select-statement can be invoked in the following ways:

- Included in DECLARE CURSOR and implicitly executed by OPEN
- Dynamically prepared, referred to in DECLARE CURSOR, and implicitly executed by OPEN
- Dynamically executed (no PREPARE required) using a DB2 ODBC function call
- Issued interactively

The first two methods are called, respectively, the *static* and the *dynamic* invocation of *select-statement*.

Embedding a statement in an application program

You can include SQL statements in a source program that will be submitted to the precompiler. Such statements are said to be *embedded* in the application program. An embedded statement can be placed anywhere in the application program where a host language statement is allowed. You must precede each embedded statement with EXEC SQL.

Executable statements: An executable statement embedded in an application program is executed every time a statement of the host language would be executed if specified in the same place. (Thus, for example, a statement within a loop is executed every time the loop is executed, and a statement within a conditional construct is executed only when the condition is satisfied.)

An embedded statement can contain references to host variables. A host variable referred to in this way can be used in one of two ways:

As input

The current value of the host variable is used in the execution of the statement.

As output

The variable is assigned a new value as a result of executing the statement.

In particular, all references to host variables in expressions and predicates are effectively replaced by current values of the variables; that is, the variables are used as input. The treatment of other references is described individually for each statement.

The successful or unsuccessful execution of the statement is indicated by setting the SQLCODE and SQLSTATE fields in the SQLCA.²⁶ You must therefore follow all executable statements by a test of SQLCODE or SQLSTATE. Alternatively, you can use the WHENEVER statement (which is itself nonexecutable) to change the flow of control immediately after the execution of an embedded statement.

Nonexecutable statements: An embedded nonexecutable statement is processed only by the precompiler. The statement is *never* executed, and acts as a

26. SQLCODE and SQLSTATE cannot be in the SQLCA when the precompiler option STDSQL(YES) is in effect. See “SQL standard language” on page 151.

Statements

“no-operation” if placed among executable statements of the application program. Therefore, you must not follow such statements by a test of the SQLCODE or SQLSTATE field in the SQLCA.

Dynamic preparation and execution

Your application program can dynamically build an SQL statement in the form of a character string placed in a host variable. In general, the statement is built from some data available to the application program (for example, input from a terminal). The statement so constructed can be prepared for execution by means of the (embedded) statement PREPARE and executed by means of the (embedded) statement EXECUTE, as described in Part 6 of *DB2 Application Programming and SQL Guide*. Alternatively, you can use the (embedded) statement EXECUTE IMMEDIATE to prepare and execute a statement in one step.

The statement may also be prepared by calling the DB2 ODBC SQLPrepare function and then executed by calling the DB2 ODBC SQLExecute function. In both cases, the application does not contain an embedded PREPARE or EXECUTE statement. You can execute the statement, without preparation, by passing the statement to the DB2 ODBC SQLExecDirect function.

DB2 ODBC Guide and Reference describes the APIs supported with this interface.

A statement that is going to be prepared must not contain references to host variables. It can instead contain parameter markers. (See “Parameter markers” on page 852 in the description of the PREPARE statement for rules concerning parameter markers.) When the prepared statement is executed, the parameter markers are effectively replaced by current values of the host variables specified in the EXECUTE statement. (See “EXECUTE” on page 776 for rules concerning this replacement.) Once prepared, a statement can be executed several times with different values of host variables.

Parameter markers are not allowed in the SQL statement prepared and executed using EXECUTE IMMEDIATE.

The successful or unsuccessful execution of the statement is indicated by the values returned in the SQLCODE and SQLSTATE fields in the SQLCA after the EXECUTE (or EXECUTE IMMEDIATE) statement. You should check the fields as described above for embedded statements.

As explained in “Authorization IDs and dynamic SQL” on page 43, the DYNAMICRULES behavior in effect determines the privilege set that is used for authorization checking when dynamic SQL statements are processed. Table 40 summarizes those privilege sets. (See Table 2 on page 44 for a list of the DYNAMICRULES bind option values that determine which behavior is in effect).

Table 40. DYNAMICRULES behaviors and authorization checking

DYNAMICRULES behavior	Privilege set
Run behavior	The union of the set of privileges held by each authorization ID of the process if the dynamically prepared statement is other than an ALTER, CREATE, DROP, GRANT, RENAME, or REVOKE statement.
	The privileges that are held by the SQL authorization ID of the process if the dynamic SQL statement is a CREATE, GRANT, or REVOKE statement.

Table 40. DYNAMICRULES behaviors and authorization checking (continued)

DYNAMICRULES behavior	Privilege set
Bind behavior	The privileges that are held by the primary authorization ID of the owner of the package or plan.
Define behavior	The privileges that are held by the authorization ID of the stored procedure or user-defined function owner (definer).
Invoke behavior	The privileges that are held by the authorization ID of the stored procedure or user-defined function invoker. However, if the invoker is the primary authorization ID of the process or the CURRENT SQLID value, secondary authorization IDs are also checked if they are needed for the required authorization. Therefore, in that case, the privilege set is the union of the set of privileges that are held by each authorization ID.

Static invocation of a SELECT statement

You can include a SELECT statement as a part of the (nonexecutable) statement DECLARE CURSOR. Such a statement is executed every time you open the cursor by means of the (embedded) statement OPEN. After the cursor is open, you can retrieve the result table a row at a time by successive executions of the (embedded) SQL FETCH statement.

If the application is using DB2 ODBC, the SELECT statement is first prepared with the SQLPrepare function call. It is then executed with the SQLExecute function call. Data is then fetched with the SQLFetch function call. The application does not explicitly open the cursor.

The SELECT statement used in this way can contain references to host variables. These references are effectively replaced by the values that the variables have at the moment of executing OPEN.

The successful or unsuccessful execution of the SELECT statement is indicated by the values returned in the SQLCODE and SQLSTATE fields in the SQLCA after the OPEN. You should check the fields as described above for embedded statements.

If the application is using DB2 ODBC, the successful execution of the SELECT statement is indicated by the return code from the SQLExecute function call. If necessary, the application may retrieve the SQLCA by calling the SQLGetSQLCA function.

Dynamic invocation of a SELECT statement

Your application program can dynamically build a SELECT statement in the form of a character string placed in a host variable. In general, the statement is built from some data available to the application program (for example, a query obtained from a terminal). The statement so constructed can be prepared for execution by means of the (embedded) statement PREPARE, and referred to by a (nonexecutable) statement DECLARE CURSOR. The statement is then executed every time you open the cursor by means of the (embedded) statement OPEN. After the cursor is open, you can retrieve the result table a row at a time by successive executions of the (embedded) SQL FETCH statement.

The SELECT statement used in that way must not contain references to host variables. It can instead contain parameter markers. (See "Notes" in "PREPARE" on page 846

Statements

page 843 for rules concerning parameter markers.) The parameter markers are effectively replaced by the values of the host variables specified in the OPEN statement. (See “OPEN” on page 842 for rules concerning this replacement.)

The successful or unsuccessful execution of the SELECT statement is indicated by the values returned in the SQLCODE and SQLSTATE fields in the SQLCA after the OPEN. You should check the fields as described above for embedded statements.

Interactive invocation

IBM relational database management systems allow you to enter SQL statements from a terminal. DB2 for OS/390 and z/OS provides SPUFI to prepare and execute these statements. Other products are also available. A statement entered in this way is said to be issued interactively.

A statement issued interactively must not contain parameter markers or references to host variables, because these make sense only in the context of an application program. For the same reason, there is no SQLCA involved.

Checking the execution of SQL statements

An application program that contains executable SQL statements must include one or both of the following stand-alone host variables:

- SQLCODE (SQLCOD in Fortran)
- SQLSTATE (SQLSTT in Fortran)

Or,

- An SQLCA, which can be provided by using the INCLUDE SQLCA statement

Whether you define stand-alone SQLCODE and SQLSTATE host variables or an SQLCA in your program depends on the DB2 precompiler option you choose.

If the application is using DB2 ODBC and it calls the SQLGetSQLCA function, it need only include an SQLCA. Otherwise, all notification of success or errors is specified with return codes for the various function calls.

When you specify STDSQL(YES), which indicates conformance to the SQL standard, you should not define an SQLCA. The stand-alone variable for SQLCODE must be a valid host variable in the DECLARE SECTION of a program. It can also be declared outside of the DECLARE SECTION when no variable is defined for SQLSTATE. The stand-alone variable for SQLSTATE must be declared in the DECLARE SECTION; it must not be declared as an element of a structure.

When you specify STDSQL(NO), which indicates conformance to DB2 rules, you must include an SQLCA explicitly.

SQLCODE

Regardless of whether the application program provides an SQLCA or a stand-alone variable for SQLCODE, DB2 sets SQLCODE after each SQL statement is executed. DB2 conforms to the SQL standard as follows:

- If SQLCODE = 0, execution was successful.
- If SQLCODE > 0, execution was successful with a warning.
- If SQLCODE < 0, execution was not successful.

SQLCODE +100 indicates "no data". For example, a FETCH statement returned no data because the cursor was positioned after the last row of the result table. The

SQL standard does not define the meaning of any other specific positive or negative values of SQLCODE and the meaning of these values is not the same in all implementations of SQL.

If the application is using DB2 ODBC, an SQLCODE is only returned if the application issues the SQLGetSQLCA function.

SQLSTATE

Regardless of whether the application program provides an SQLCA or a stand-alone variable for SQLSTATE, DB2 sets SQLSTATE after each SQL statement is executed. DB2 returns values that conform to the error specification in the SQL standard.

If the application is using DB2 ODBC, the SQLSTATE returned conforms to the ODBC Version 2.0 specification.

SQLSTATE provides application programs with common codes for common error conditions (the values of SQLSTATE are product-specific if the error or warning is product-specific). Furthermore, SQLSTATE is designed so that application programs can test for specific errors or classes of errors. The coding scheme is the same for all IBM implementations of SQL. The SQLSTATE values are based on the SQLSTATE specifications contained in the SQL standard.

Error messages and the tokens that are substituted for variables in error messages are associated with SQLCODE values, not SQLSTATE values.

ALLOCATE CURSOR

ALLOCATE CURSOR

The ALLOCATE CURSOR statement specifies a cursor and associates it with a result set locator variable.

Invocation

This statement can be embedded in an application program. It is an executable statement that can be dynamically prepared. It cannot be issued interactively.

Authorization

None required.

Syntax

```
►►ALLOCATE—cursor-name—CURSOR FOR RESULT SET—rs-locator-variable►►
```

Description

cursor-name

Identifies the cursor. The name must not identify a cursor that has already been declared in the source program.

CURSOR FOR RESULT SET *rs-locator-variable*

Specifies a result set locator variable that has been declared in the application program according to the rules for declaring result set locator variables.

The result set locator variable must contain a valid result set locator value, as returned by the ASSOCIATE LOCATORS or DESCRIBE PROCEDURE SQL statement.

Notes

Dynamically prepared ALLOCATE CURSOR statements: The EXECUTE statement with the USING clause must be used to execute a dynamically prepared ALLOCATE CURSOR statement. In a dynamically prepared statement, references to host variables are represented by parameter markers (question marks). In the ALLOCATE CURSOR statement, *rs-locator-variable* is always a host variable. Thus, for a dynamically prepared ALLOCATE CURSOR statement, the USING clause of the EXECUTE statement must identify the host variable whose value is to be substituted for the parameter marker that represents *rs-locator-variable*.

You cannot prepare an ALLOCATE CURSOR statement with a statement identifier that has already been used in a DECLARE CURSOR statement. For example, the following SQL statements are invalid because the PREPARE statement uses STMT1 as an identifier for the ALLOCATE CURSOR statement and STMT1 has already been used for a DECLARE CURSOR statement.

```
DECLARE CURSOR C1 FOR STMT1;
```

```
PREPARE STMT1 FROM      INVALID  
'ALLOCATE C2 CURSOR FOR RESULT SET ?';
```

Rules for using an allocated cursor: The following rules apply when you use an allocated cursor:

- You cannot open an allocated cursor with the OPEN statement.

- You can close an allocated cursor with the CLOSE statement. Closing an allocated cursor closes the associated cursor defined in the stored procedure.
- You can allocate only one cursor to each result set.

The life of an allocated cursor: A rollback operation, an implicit close, or an explicit close destroy allocated cursors. A commit operation destroys allocated cursors that are not defined WITH HOLD by the stored procedure. Destroying an allocated cursor closes the associated cursor defined in the stored procedure.

Example

The statement in the following example is assumed to be in a PL/I program.

Define and associate cursor C1 with the result set locator variable LOC1 and the related result set returned by the stored procedure:

```
EXEC SQL ALLOCATE C1 CURSOR FOR RESULT SET :LOC1;
```

ALTER DATABASE

ALTER DATABASE

The ALTER DATABASE statement changes the description of a database at the current server.

Invocation

This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is implicitly or explicitly specified.

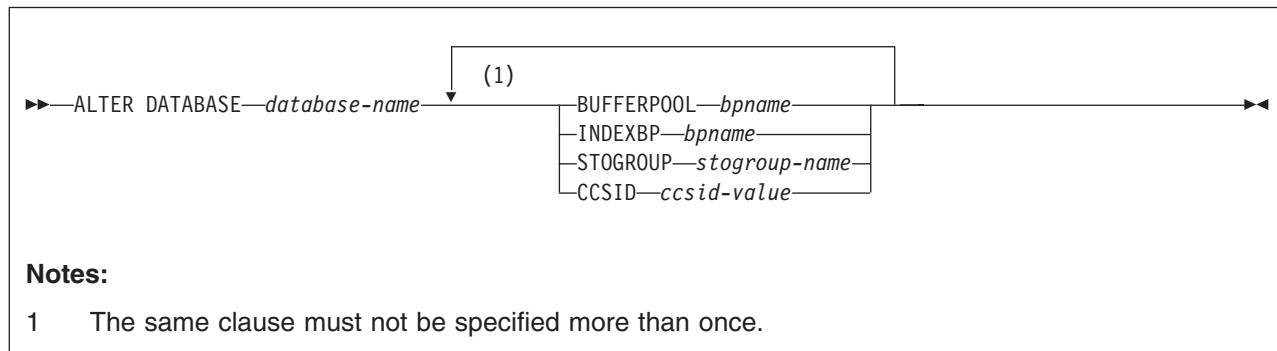
Authorization

The privilege set that is defined below must include at least one of the following:

- The DROP privilege on the database
- Ownership of the database
- DBADM or DBCTRL authority for the database
- SYSADM or SYSCTRL authority

Privilege set: If the statement is embedded in an application program, the privilege set is the privileges that are held by the authorization ID of the owner of the plan or package. If the statement is dynamically prepared, the privilege set is the union of the privilege sets that are held by each authorization ID of the process.

Syntax



Description

DATABASE *database-name*

Identifies the database to be altered. The name must identify a database that exists at the current server.

BUFFERPOOL *bpname*

Identifies the default buffer pool for the table spaces within the database. It does not apply to table spaces that already exist within the database.

If the database is a work file database, 8KB and 16KB buffer pools cannot be specified.

See “Naming conventions” on page 34 for more details about *bpname*.

INDEXBP *bpname*

Identifies the default buffer pool for the indexes within the database. It does not apply to indexes that already exist within the database. The name must identify a 4KB buffer pool. See “Naming conventions” on page 34 for more details about *bpname*.

INDEXBP cannot be specified for a work file database.

STOGROUP *stogroup-name*

Identifies the storage group to be used, as required, as a default storage group to support DASD space requirements for table spaces and indexes within the database. It does not apply to table spaces and indexes that already exist within the database.

STOGROUP cannot be specified for a work file database.

CCSID *ccsid-value*

Identifies the default CCSID for table spaces within the database. It does not apply to existing table spaces in the database. *ccsid-value* must identify a CCSID value that is compatible with the current value of the CCSID for the database. "Notes" contains a list that shows the CCSID to which a given CCSID can be altered.

CCSID cannot be specified for a work file database or a TEMP database.

Notes

Altering the CCSID: The ability to alter the default CCSID enables you to change to a CCSID that supports the Euro symbol. You can only convert between specific CCSIDs that do and do not define the Euro symbol. In most cases, the codepoint that supports the Euro symbol replaces an existing codepoint, such as the International Currency Symbol (ICS).

Changing a CCSID can be disruptive to the system and requires several steps. For each encoding scheme of a system (ASCII, EBCDIC, and Unicode), DB2 supports SBCS, DBCS, and mixed CCSIDs. Therefore, the CCSIDs for all databases and all table spaces within an encoding scheme should be altered at the same time. Otherwise, unpredictable results might occur.

The recommended method for changing the CCSID requires that the data be unloaded and reloaded. See Appendix A of *DB2 Installation Guide* for the steps needed to change the CCSID, such as running an installation CLIST to modify the CCSID data in DSNHDECP, when to drop and recreate views, and when to rebind invalidated plans and packages.

The following lists show the CCSIDs that can be converted. The second CCSID in each pair is the CCSID with the Euro symbol. The CCSID can be changed from the CCSID that does not support the Euro symbol to the CCSID that does, and vice versa. For example, if the current CCSID is 500, it can be changed to 1148.

EBCDIC CCSIDs

<hr/>	
37	1140
273	1141
277	1142
278	1143
280	1144
284	1145
285	1146
297	1147
500	1148
871	1149

ALTER DATABASE

ASCII CCSIDs	
850	858
874	4970
1250	5346
1251	5347
1252	5348
1253	5349
1254	5350
1255	5351
1256	5352
1257	5353

Example

Change the default buffer pool for both table spaces and indexes within database ABCDE to BP2.

```
ALTER DATABASE ABCDE
  BUFFERPOOL BP2
  INDEXBP BP2;
```

ALTER FUNCTION (external)

The ALTER FUNCTION statement changes the description of a user-defined external scalar or external table function at the current server.

Invocation

This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is implicitly or explicitly specified.

Authorization

The privilege set defined below must include at least one of the following:

- Ownership of the function
 - The ALTERIN privilege for the schema or all schemas
 - SYSADM or SYSCTRL authority

The authorization ID that matches the schema name implicitly has the **ALTERIN** privilege on the schema.

Privilege set: If the statement is embedded in an application program, the privilege set is the privileges that are held by the authorization ID of the owner of the plan or package.

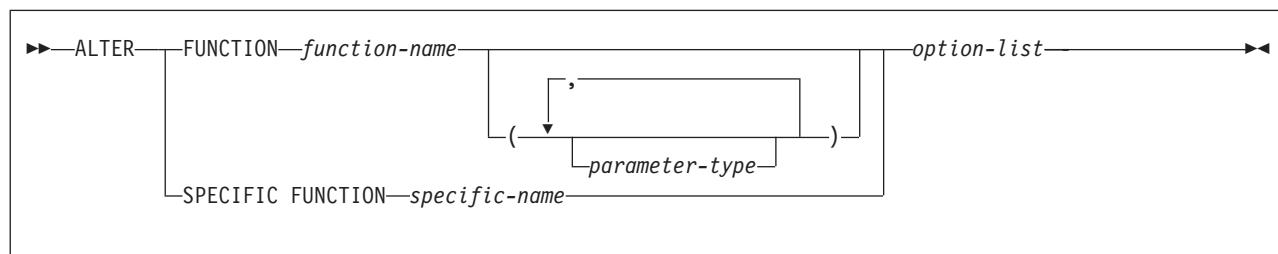
If the statement is dynamically prepared, the privilege set is the privileges that are held by the authorization IDs of the process. The specified function name can include a schema name (a qualifier). However, if the schema name is not the same as one of these authorization IDs, one of the following conditions must be met:

- The privilege set includes SYSADM or SYSCTRL authority.
 - An authorization ID of the process has the ALTERIN privilege on the schema.

If the environment in which the function is to be run is being changed, the authorization ID must have authority to use the WLM environment specified. The required authorization is obtained from an external security product, such as RACF.

For *external scalar functions*, when LANGUAGE is JAVA and a *jar-name* is specified in the EXTERNAL NAME clause, the privilege set must include USAGE on the JAR, the Java ARchive file.

Syntax



ALTER FUNCTION (external)

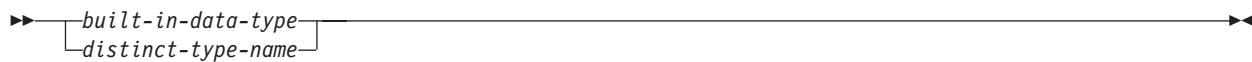
parameter-type:

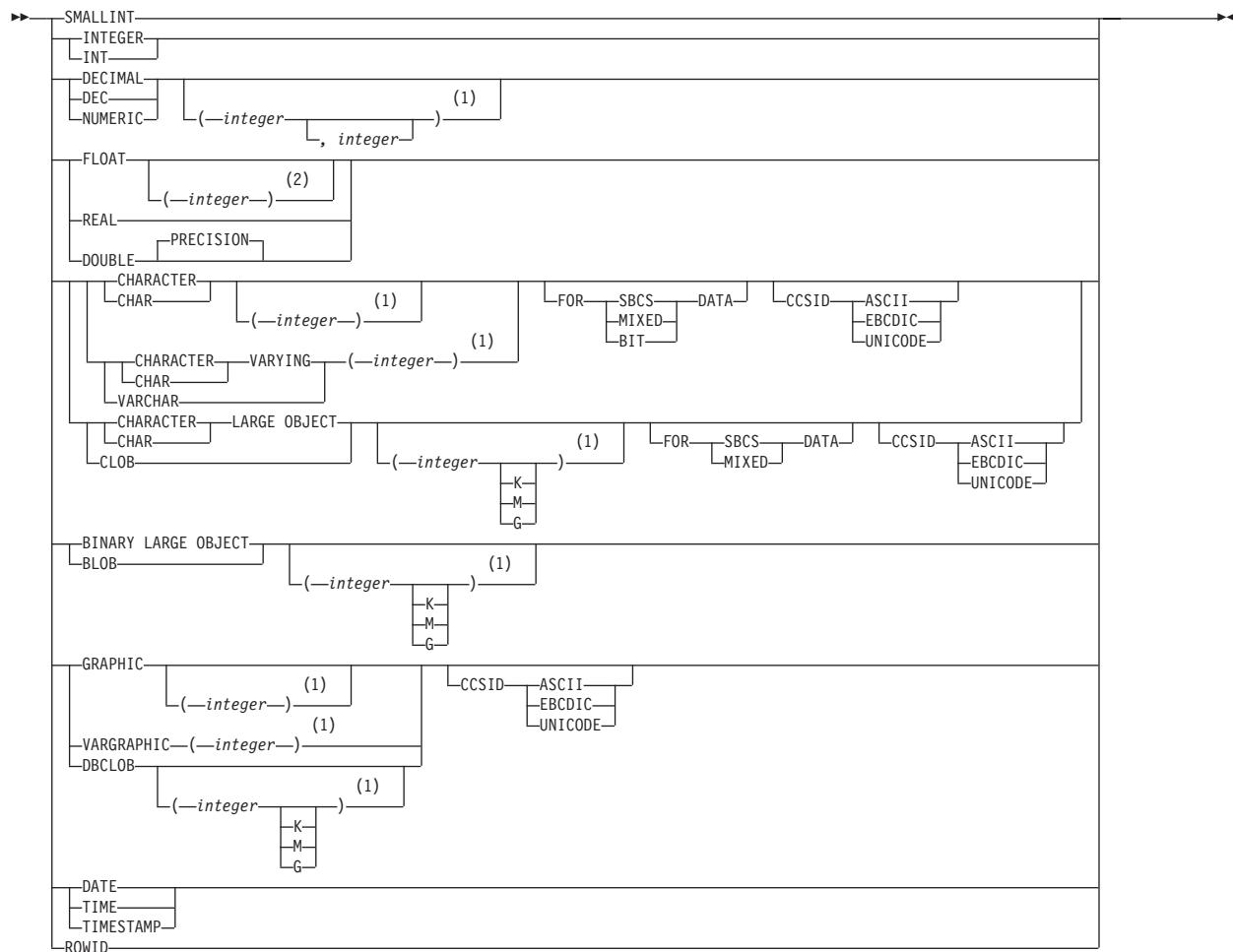


Notes:

- 1 AS LOCATOR can be specified only for a LOB data type or a distinct type based on a LOB data type.

data-type:

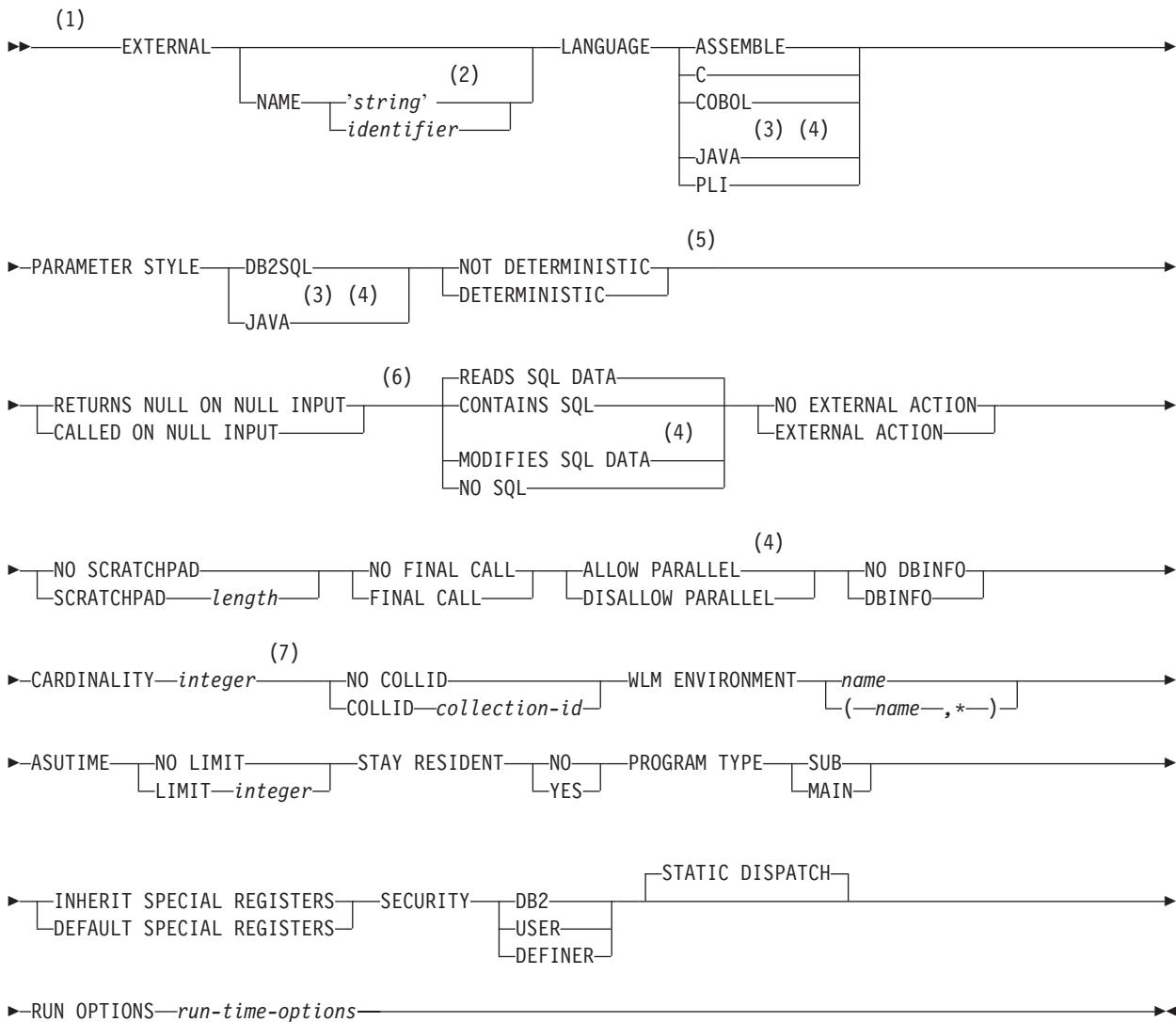


built-in-data-type:**Notes:**

- 1 The values that are specified for length, precision, or scale attributes must match the values that were specified when the function was created. Coding specific values is optional. Empty parentheses, (), can be used instead to indicate that DB2 is to ignore the attributes when determining whether data types match.
- 2 The value that is specified does not have to match the value that was specified when the function was created because matching is based on data type (REAL or DOUBLE). $1 \leq integer \leq 21$ indicates REAL and $22 \leq integer \leq 53$ indicates DOUBLE. Coding a specific value is optional. Empty parentheses cannot be used.

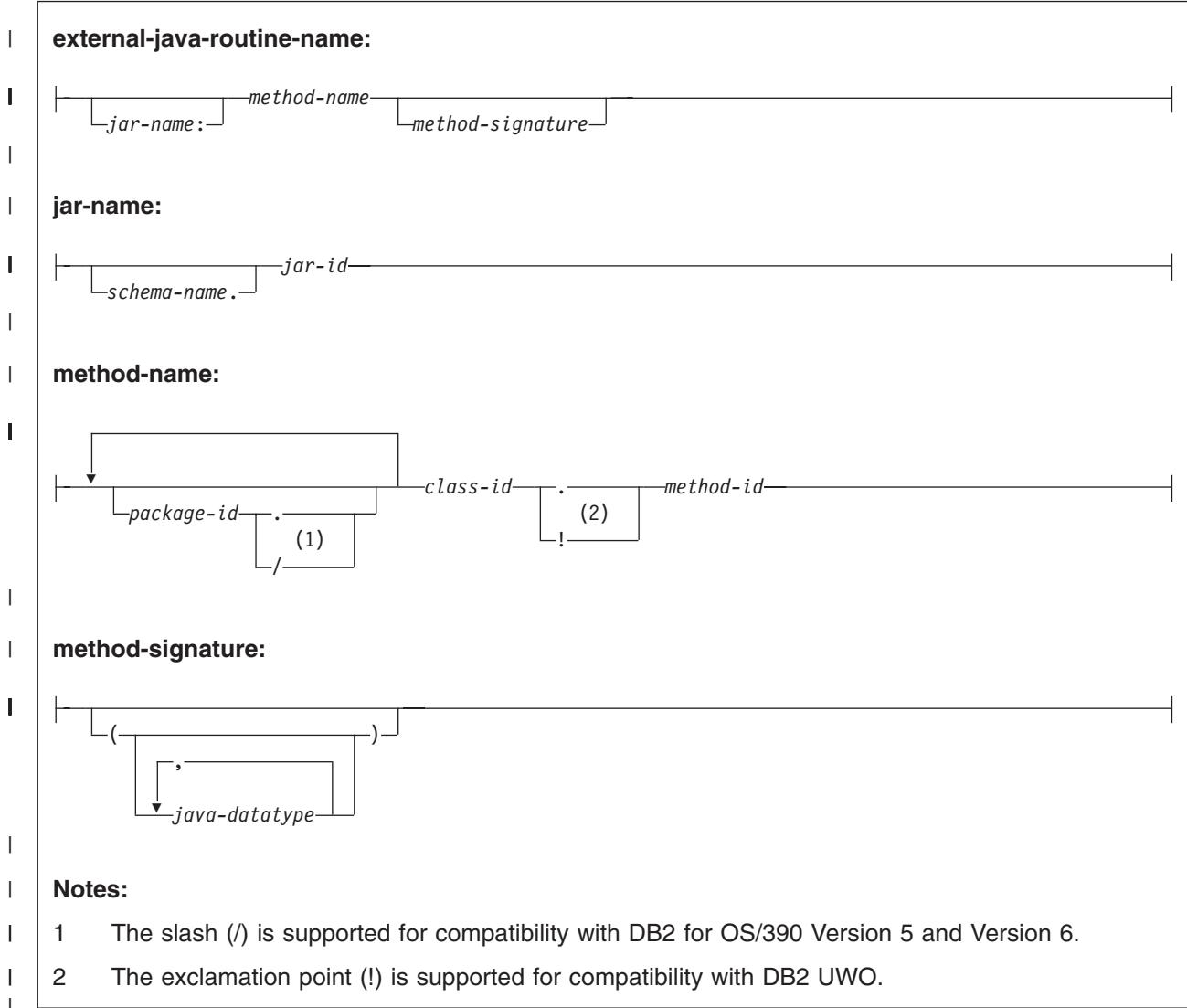
ALTER FUNCTION (external)

option-list:



Notes:

- 1 The clauses in the *option-list* can be specified in any order, but only once.
- 2 For LANGUAGE JAVA, use a valid *external-java-routine-name* for EXTERNAL NAME.
- 3 When LANGUAGE JAVA is specified, PARAMETER STYLE JAVA must also be specified. When PARAMETER STYLE JAVA is specified, LANGUAGE JAVA must be also be specified.
- 4 LANGUAGE JAVA, PARAMETER STYLE JAVA, MODIFIES SQL DATA, and ALLOW PARALLEL are not supported for *external table functions*.
- 5 Synonyms include VARIANT for NOT DETERMINISTIC, and NOT VARIANT for DETERMINISTIC.
- 6 Synonyms include NOT NULL CALL and NULL CALL.
- 7 CARDINALITY is not supported for *external scalar functions*.



Description

One of the following three clauses identifies the function to be changed.

FUNCTION *function-name*

Identifies the external function by its function name. The name is implicitly or explicitly qualified with a schema name. If the name is not explicitly qualified, it is implicitly qualified with a schema name according to the following rules:

- If the statement is embedded in a program, the schema name is the authorization ID in the QUALIFIER bind option when the plan or package was created or last rebound. If QUALIFIER was not specified, the schema name is the owner of the plan or package.
- If the statement is prepared dynamically, the schema name is the SQL authorization ID in the CURRENT SQLID special register.

The identified function must be an external function. There must be exactly one function with *function-name* in the schema. The function can have any number of input parameters. If the schema does not contain a function with *function-name* or contains more than one function with this name, an error occurs.

ALTER FUNCTION (external)

FUNCTION *function-name* (*parameter-type*,...)

Identifies the external function by its function signature, which uniquely identifies the function. If *function-name()* is specified, the function that is identified must have zero parameters.

function-name

Gives the function name of the external function. If the function name is not qualified, it is implicitly qualified with a schema name as described in the preceding description for FUNCTION *function-name*.

(*parameter-type*,...)

Identifies the number of input parameters of the function and their data types.

The data type of each parameter must match the data type that was specified in the CREATE FUNCTION statement for the parameter in the corresponding position. The number of data types and the logical concatenation of the data types are used to uniquely identify the function. Therefore, you cannot change the number of parameters or the data types of the parameters.

For data types that have a length, precision, or scale attribute, you can specify a value or use a set of empty parentheses:

- Empty parentheses indicate that DB2 is to ignore the attribute when determining whether the data types match.
FLOAT cannot be specified with empty parentheses because its parameter value indicates different data types (REAL or DOUBLE).
- If you use a specific value for a length, precision, or scale attribute, the value must exactly match the value that was specified (implicitly or explicitly) in the CREATE FUNCTION statement.

The specific value for FLOAT(*n*) does not have to exactly match the defined value of the source function because 1<=n<= 21 indicates REAL and 22<=n<=53 indicates DOUBLE. Matching is based on whether the data type is REAL or DOUBLE.

- If length, precision, or scale is not explicitly specified, and empty parentheses are not specified, the default length of the data type is implied. For example:

CHAR	CHAR(1)
GRAPHIC	GRAPHIC(1)
DECIMAL	DECIMAL(5,0)
FLOAT	DOUBLE (length of 8)

The implicit length must exactly match the value that was specified (implicitly or explicitly) in the CREATE FUNCTION statement. For a complete list of the default lengths of data types, see "CREATE TABLE" on page 653.

For data types with a subtype or encoding scheme attribute, specifying the FOR DATA clause or CCSID clause is optional. Omission of either clause indicates that DB2 is to ignore the attribute when determining whether the data types match. If you specify either clause, it must match the value that was implicitly or explicitly specified in the CREATE FUNCTION statement.

See "CREATE FUNCTION" on page 529 for more information on the specification of the parameter list.

A function with the function signature must exist in the explicitly or implicitly specified schema; otherwise, an error occurs.

SPECIFIC FUNCTION *specific-name*

Identifies the external function by its specific name. The name is implicitly or explicitly qualified with a schema name. A function with the specific name must exist in the schema; otherwise, an error occurs.

If the specific name is not qualified, it is implicitly qualified with a schema name as described in the preceding description for FUNCTION *function-name*.

The following clauses change the description of the function that has been identified to be changed.

EXTERNAL

Identifies the program that runs when the function is invoked.

DB2 loads the load module when the function is invoked. The load module is created when the program that contains the function body is compiled and link-edited. The load module does not need to exist when the ALTER FUNCTION statement is executed. However, it must exist and be accessible by the current server when the function is invoked.

You can specify the EXTERNAL clause in one of the following ways:

```
EXTERNAL  
EXTERNAL NAME PKJVSP1  
EXTERNAL NAME 'PKJVSP1'
```

If you specify an external program name, you must use the NAME keyword. For example, this syntax is not valid:

```
EXTERNAL PKJVSP1
```

NAME '*string*' or *identifier*

Identifies the user-written code that implements the user-defined function.

If LANGUAGE is JAVA, '*string*' must be specified and enclosed in single quotation marks, with no extraneous blanks within the single quotation marks. It must specify a valid *external-java-routine-name*. If multiple '*string*'s are specified, the total length of all of them must not be greater than 1305 bytes and they must be separated by a space or a line break. Do not specify a JAR for a JAVA function for which NO SQL is in effect.

An *external-java-routine-name* contains the following parts:

jar-name

Identifies the name given to the JAR when it was installed in the database. The name contains *jar-id*, which can optionally be qualified with a schema. Examples are "myJar" and "mySchema.myJar." The unqualified *jar-id* is implicitly qualified with a schema name according to the following rules:

- If the statement is embedded in a program, the schema name is the authorization ID in the QUALIFIER bind option when the package or plan was created or last rebound. If the QUALIFIER was not specified, the schema name is the owner of the package or plan.
- If the statement is dynamically prepared, the schema name is the SQL authorization ID in the CURRENT SQLID special register.

If *jar-name* is specified, it must exist when the ALTER FUNCTION statement is processed.

ALTER FUNCTION (external)

If *jar-name* is not specified, the function is loaded from the class file directly instead of being loaded from a JAR file. DB2 for DB2 for OS/390 and z/OS searches the directories in the CLASSPATH associated with the WLM Environment. Environmental variables for Java routines are specified in a dataset identified in a JAVAENV DD card on the JCL used to start the address space for a WLM-managed function.

method-name

Identifies the name of the method and must not be longer than 254 bytes. Its package, class, and method ID's are specific to Java and as such are not limited to 18 bytes. In addition, the rules for what these can contain are not necessarily the same as the rules for an SQL ordinary identifier.

package-id

Identifies the package list that the class identifier is part of. If the class is part of a package, the method name must include the complete package prefix, such as "myPacks.UserFuncs." The Java virtual machine looks in the directory "/myPacks/UserFuncs/" for the classes.

class-id

Identifies the class identifier of the Java object.

method-id

Identifies the method identifier with the Java class to be invoked.

method-signature

Identifies a list of zero or more Java data types for the parameter list and must not be longer than 1024 bytes. Specify the *method-signature* if the user-defined function involves any input or output parameters that can be NULL. When the function being created is called, DB2 searches for a Java method with the exact *method-signature*. The number of *java-datatype* elements specified indicates how many parameters that the Java method must have.

A Java procedure can have no parameters. In this case, you code an empty set of parentheses for *method-signature*. If a Java *method-signature* is not specified, DB2 searches for a Java method with a signature derived from the default JDBC types associated with the SQL types specified in the parameter list of the ALTER FUNCTION statement.

For other values of LANGUAGE, the name can be a string constant that is no longer than 8 characters or a short identifier. It must conform to the naming conventions for MVS load modules. Alphabetical extenders for national languages can be used as the first character and as subsequent characters in the load module name.

If you do not specify the NAME clause, 'NAME *function-name*' is implicit. In this case, *function-name* must not be longer than 8 characters.

LANGUAGE

Specifies the application programming language in which the function is written. All programs must be designed to run in IBM's Language Environment® environment .

ASSEMBLE

The function is written in Assembler.

C The function is written in C or C++.

COBOL

The function is written in COBOL, including the object-oriented language extensions.

JAVA

The user-defined function is written in Java byte code and is executed in the OS/390 Java Virtual Machine. If the ALTER FUNCTION statement results in changing LANGUAGE to JAVA, PARAMETER STYLE JAVA and an EXTERNAL NAME clause must be specified to provide the appropriate values. When LANGUAGE JAVA is specified, the EXTERNAL NAME clause must also be specified with a valid *external-java-routine-name* and PARAMETER STYLE must be specified with JAVA.

Do not specify LANGUAGE JAVA when SCRATCHPAD, FINAL CALL, DBINFO, PROGRAM TYPE MAIN, or RUN OPTIONS is in effect. Do not specify LANGUAGE JAVA for a table function.

PLI

The function is written in PL/I.

PARAMETER STYLE

Specifies the linkage convention that the function program uses to receive input parameters from and pass return values to the invoking SQL statement.

DB2SQL

Indicates that parameters for indicator variables are associated with each input and return value to allow for null values. The parameters that are passed between the invoking SQL statement and the function include:

- Input parameters. The first n parameters are the input parameters that are specified for the function.
- Result parameters. For an external scalar function, a parameter for the result of the function that is specified on the RETURNS clause. For an external table function, the next m parameters are the result columns of the function that are specified on the RETURNS TABLE clause.
- Input parameter indicator variables. n parameters for the indicator variables for the input parameters.
- Result parameter indicator variables. For an external scalar function, a parameter for the indicator variable for the result of the function that is specified on the RETURNS clause. For an external table function, m parameters for the indicator variables of the result columns of the function that are specified on the RETURNS TABLE clause.
- The SQLSTATE to be returned to DB2.
- The qualified name of the function.
- The specific name of the function.
- The SQL diagnostic string to be returned to DB2.
- The scratchpad, if SCRATCHPAD is specified.
- The call type. For an external scalar function, the call type is passed only if FINAL CALL is specified.
- The DBINFO structure, if DBINFO is specified.

JAVA

Indicates that the user-defined function uses a convention for passing parameters that conforms to the Java and SQLJ specifications. If the ALTER FUNCTION statement results in changing LANGUAGE to JAVA, PARAMETER STYLE JAVA and an EXTERNAL NAME clause must be specified to provide the appropriate values. PARAMETER STYLE JAVA can

ALTER FUNCTION (external)

| be specified only if LANGUAGE is JAVA. JAVA must be specified for
| PARAMETER STYLE when LANGUAGE is JAVA.

| Do not specify PARAMETER STYLE JAVA for a table function.

NOT DETERMINISTIC or DETERMINISTIC

Specifies whether the function returns the same results for identical input arguments.

NOT DETERMINISTIC

The function might not return the same result for identical input arguments. The function depends on some state values that affect the results. DB2 uses this information to disable the merging of views and table expressions when processing SELECT, UPDATE, DELETE, or INSERT statements that reference this function. An example of a function that is not deterministic is one that generates random numbers, or any function that contains SQL statements.

Some SQL functions that invoke functions that are not deterministic can receive incorrect results if the function is executed by parallel tasks. Specify the DISALLOW PARALLEL clause for these functions.

If a view refers to the function, the function cannot be changed to NOT DETERMINISTIC. To change the function, drop any views that refer to the function first.

DETERMINISTIC

The function always returns the same result for identical input arguments. An example of a deterministic function is a function that calculates the square root of the input. DB2 uses this information to enable the merging of views and table expressions for SELECT, UPDATE, DELETE, or INSERT statements that reference this function. If applicable, specify DETERMINISTIC to prevent non-optimal access paths from being chosen for SQL statements that reference this function.

DB2 does not verify that the function program is consistent with the specification of DETERMINISTIC or NOT DETERMINISTIC.

RETURNS NULL ON NULL INPUT or CALLED ON NULL INPUT

Specifies whether the function is called if any of the input arguments is null at execution time.

RETURNS NULL ON NULL INPUT

The function is not called if any of the input arguments is null. For an external scalar function, the result is the null value. For an external table function, the result is an empty table, which is a table with no rows.

CALLED ON NULL INPUT

The function is called regardless of whether any of the input arguments is null, making the function responsible for testing for null argument values. For an external scalar function, the function can return a null or nonnull value. For an external table function, the function can return an empty table, depending on its logic.

NO SQL, MODIFIES SQL DATA, READS SQL DATA, or CONTAINS SQL

Indicates whether the function issues any SQL statements and, if so, what type. DB2 verifies that the SQL issued by the function is consistent with this specification. See Table 72 on page 972 for a detailed list of the SQL statements that can be executed under each data access indication.

NO SQL

The function does not execute SQL statements. Do not specify NO SQL for a JAVA function that uses a JAR.

MODIFIES SQL DATA

The function might execute any SQL statement except those statements that are not supported in any function. Do not specify MODIFIES SQL DATA for external table functions or when ALLOW PARALLEL is in effect.

READS SQL DATA

The function does not execute SQL statements that modify data. SQL statements that are not supported in any function return a different error.

READS SQL DATA is the default.

CONTAINS SQL

The function does not execute SQL statements that read or modify data. SQL statements that are not supported in any function return a different error.

NO EXTERNAL ACTION or EXTERNAL ACTION

Specifies whether the function takes an action that changes the state of an object that DB2 does not manage. An example of an external action is sending a message or writing a record to a file.

Because DB2 uses the RRS attachment for external functions, DB2 can participate in two-phase commit with any other resource manager that uses RRS. For resource managers that do not use RRS, there is no coordination of commit or rollback operations on non-DB2 resources.

NO EXTERNAL ACTION

The function does not take any action that changes the state of an object that DB2 does not manage. DB2 uses this information to enable the merging of views and table expressions for SELECT, UPDATE, DELETE, or INSERT statements that reference this function. If applicable, specify NO EXTERNAL ACTION to prevent non-optimal access paths from being chosen for SQL statements that reference this function.

EXTERNAL ACTION

The function can take an action that changes the state of an object that DB2 does not manage.

Some SQL statements that invoke functions with external actions can result in incorrect results if parallel tasks execute the function. For example, if the function sends a note for each initial call to it, one note is sent for each parallel task instead of once for the function. Specify the DISALLOW PARALLEL clause for functions that do not work correctly with parallelism.

If you specify EXTERNAL ACTION, DB2:

- Materializes the views and table expressions in SELECT, UPDATE, DELETE or INSERT statements that refer to the function. This materialization can adversely affect the access paths that are chosen for the SQL statements that reference this function. Do not specify EXTERNAL ACTION if the function does not have an external action.
- Does not move the function from one task control block (TCB) to another between FETCH operations.
- Does not allow another function or stored procedure to use the TCB until the cursor is closed. This is also applicable for cursors declared WITH HOLD.

```
#  
#  
#  
#  
#
```

ALTER FUNCTION (external)

The only changes to resources made outside of DB2 that are under the control of commit and rollback operations are those changes made under RRS control.

If a view refers to the function, the function cannot be changed to EXTERNAL ACTION. To change the function, drop any views that refer to the function first.

DB2 does not verify that the function program is consistent with the specification of EXTERNAL ACTION or NO EXTERNAL ACTION.

NO SCRATCHPAD or **SCRATCHPAD**

Specifies whether DB2 is to provide a scratchpad for the function. It is strongly recommended that external functions be reentrant, and a scratchpad provides an area for the function to save information from one invocation to the next.

NO SCRATCHPAD

A scratchpad is not allocated and passed to the function.

SCRATCHPAD *length*

When the function is invoked for the first time, DB2 allocates memory for a scratchpad. A scratchpad has the following characteristics:

- *length* must be between 1 and 32767. The default value is 100 bytes.
- DB2 initializes the scratchpad to all binary zeros (X'00's).
- The scope of a scratchpad is the SQL statement. For each reference to the function in an SQL statement, there is one scratchpad. For example, assuming that function UDFX was defined with the SCRATCHPAD keyword, three scratchpads are allocated for the three references to UDFX in the following SQL statement:

```
SELECT A, UDFX(A) FROM TABLEB  
      WHERE UDFX(A) > 103 OR UDFX(A) < 19;
```

If the function is run under parallel tasks, one scratchpad is allocated for each parallel task of each reference to the function in the SQL statement. This can lead to unpredictable results. For example, if a function uses the scratchpad to count the number of times that it is invoked, the count reflects the number of invocations done by the parallel task and not the SQL statement. Specify the DISALLOW PARALLEL clause for functions that do not work correctly with parallelism.

- The scratchpad is persistent. DB2 preserves its content from one invocation of the function to the next. Any changes that the function makes to the scratchpad on one call are still there on the next call. DB2 initializes the scratchpads when it begins to execute an SQL statement. DB2 does not reset scratchpads when a correlated subquery begins to execute.
- The scratchpad can be a central point for the system resources that the function acquires. If the function acquires system resources, specify FINAL CALL to ensure that DB2 calls the function one more time so that the function can free those system resources.

Each time that the function is invoked, DB2 passes an additional argument to the function that contains the address of the scratchpad.

If you specify SCRATCHPAD, DB2:

- Does not move the function from one TCB or address space to another between FETCH operations.

- Does not allow another function or stored procedure to use the TCB until the cursor is closed. This is also applicable for cursors declared WITH HOLD.

| Do not specify SCRATCHPAD when LANGUAGE JAVA is in effect.

NO FINAL CALL or FINAL CALL

Specifies whether a *final call* is made to the function. A final call enables the function to free any system resources that it has acquired. A final call is useful when the function has been defined with the SCRATCHPAD keyword and the function acquires system resource and anchors them in the scratchpad.

The effect of NO FINAL CALL or FINAL call depends on whether the external function is a scalar function or a table function.

For an external scalar function:**NO FINAL CALL**

A final call is not made to the external scalar function. The function does not receive an additional argument that specifies the type of call.

FINAL CALL

A final call is made to the external scalar function. See the following description of call types for the characteristics of a final call. When FINAL CALL is specified, the function receives an additional argument that specifies the type of call to enable the function to differentiate between a final call and another type of call. Do not specify FINAL CALL when LANGUAGE JAVA is in effect.

| For more information on NO FINAL CALL and FINAL CALL for external scalar functions, including the types of calls, see the description of the option for “CREATE FUNCTION (external scalar)” on page 530.

For an external table function:**NO FINAL CALL**

A first and final call are not made to the external table function.

FINAL CALL

A first call and final call are made to the external table function in addition to one or more other types of calls.

For both NO FINAL CALL and FINAL CALL, the function receives an additional argument that specifies the type of call. For more information on NO FINAL CALL and FINAL CALL for external table functions, including the types of calls, see the description of the option for “CREATE FUNCTION (external table)” on page 553.

ALLOW or DISALLOW PARALLEL

Specifies whether, for a single reference to the function, the function can be executed in parallel. If the function is defined with MODIFIES SQL DATA, specify DISALLOW PARALLEL, not ALLOW PARALLEL.

ALLOW PARALLEL

Specifies that DB2 can consider parallelism for the function. Parallelism is not forced on the SQL statement that invokes the function or on any SQL statement in the function. Existing restrictions on parallelism apply.

See SCRATCHPAD, EXTERNAL ACTION, and FINAL CALL for considerations when specifying ALLOW PARALLEL.

ALTER FUNCTION (external)

DISALLOW PARALLEL

Specifies that DB2 does not consider parallelism for the function.

NO DBINFO or DBINFO

Specifies whether specific information that DB2 knows is passed to the function when it is invoked.

NO DBINFO

Additional information is not passed.

DBINFO

An additional argument is passed when the function is invoked. The argument is a structure that contains information such as the application run-time authorization ID, the schema name, the name of a table or column that the function might be inserting into or updating, and identification of the database server that invoked the function. For details about the argument and its structure, see *DB2 Application Programming and SQL Guide*.

| Do not specify DBINFO when LANGUAGE JAVA is in effect.

CARDINALITY *integer*

Specifies an estimate of the expected number of rows that the function returns. The number is used for optimization purposes. The value of *integer* must range from 0 to 2147483647.

If a function has an infinite cardinality—the function never returns the “end-of-table” condition and always returns a row, then a query that requires the “end-of-table” to work correctly, will need to be interrupted. Thus, avoid using such functions in queries that involve GROUP BY and ORDER BY.

| Do not specify CARDINALITY for external scalar functions.

NO COLLID or COLLID *collection-id*

Identifies the package collection that is to be used when the function is executed. This is the package collection into which the DBRM that is associated with the function program is bound.

NO COLLID

The package collection for the function is the same as the package collection of the program that invokes the function. If a trigger invokes the function, the collection of the trigger package is used. If the invoking program does not use a package, the package collection is the value of the CURRENT PACKAGESET special register.

COLLID *collection-id*

The name of the package collection that is to be used when the function is executed.

WLM ENVIRONMENT

A long SQL identifier that identifies the *name* of the WLM (MVS workload manager) application environment in which the function is to run.

name

The WLM environment in which the function must run. If the user-defined function is nested and if the calling stored procedure or invoking user-defined function is not running in an address space associated with the specified WLM environment, DB2 routes the function request to a different MVS address space.

(*name,**)

When an SQL application program calls the function, *name* specifies the WLM environment in which the function runs.

If another user-defined function or a stored procedure calls the function, the function runs in the same environment that the calling routine uses. In this case, authorization to run the function in the WLM environment is not checked because the authorization of the calling routine suffices.

The *name* of the WLM environment is a long identifier.

To change the environment in which the function is to run, you must have appropriate authority for the WLM environment. For an example of a RACF command that provides this authorization, see “Running stored procedures” on page 634.

ASUTIME

Specifies the total amount of processor time, in CPU service units, that a single invocation of the function can run. The value is unrelated to the ASUTIME column of the resource limit specification table.

When you are debugging a function, setting a limit can be helpful if the function gets caught in a loop. For information on service units, see *OS/390 MVS Initialization and Tuning Guide*.

NO LIMIT

There is no limit on the service units.

LIMIT *integer*

The limit on the service units is a positive integer in the range of 1 to 2G. If the function uses more service units than the specified value, DB2 cancels the function.

STAY RESIDENT

Specifies whether the load module for the function is to remain resident in memory when the function ends.

NO

The load module is deleted from memory after the function ends. Use NO for non-reentrant functions.

YES

The load module remains resident in memory after the function ends. Use YES for reentrant functions.

PROGRAM TYPE

Specifies whether the function program runs as a main routine or a subroutine.

SUB

The function runs as a subroutine.

MAIN

The function runs as a main routine.

| Do not specify PROGRAM TYPE MAIN when LANGUAGE JAVA is in effect.

INHERIT SPECIAL REGISTERS

| Indicates that special registers should be inherited according to the rules listed in the table for characteristics of special registers in a user-defined function in Table 19 on page 93.

DEFAULT SPECIAL REGISTERS

| Indicates that special registers should be initialized to the default values, as indicated by the rules in the table for characteristics of special registers in a user-defined function in Table 19 on page 93.

ALTER FUNCTION (external)

SECURITY

Specifies how the function interacts with an external security product, such as RACF, to control access to non-SQL resources.

DB2

The function does not require an external security environment. If the function accesses resources that an external security product protects, the access is performed using the authorization ID associated with the WLM-established stored procedure address space.

USER

An external security environment should be established for the function. If the function accesses resources that the external security product protects, the access is performed using the primary authorization ID of the process that invoked the function.

DEFINER

An external security environment should be established for the function. If the function accesses resources that the external security product protects, the access is performed using the authorization ID of the owner of the function.

#

STATIC DISPATCH

At function resolution time, DB2 chooses a function based on the static (or declared) types of the function parameters. STATIC DISPATCH is the default.

RUN OPTIONS *run-time-options*

Specifies the Language Environment run-time options to be used for the function. You must specify *run-time-options* as a character string that is no longer than 254 bytes. To replace any existing run-time options with no options, specify an empty string with RUN OPTIONS. When you specify an empty string, DB2 does not pass any run-time options to Language Environment, and Language Environment uses its installation defaults.

For a description of the Language Environment run-time options, see *OS/390 Language Environment for OS/390 & VM Programming Reference*.

Do not specify RUN OPTIONS when LANGUAGE JAVA is in effect.

Notes

Changes are immediate: Any changes that the ALTER FUNCTION statement causes to the definition of an external function take effect immediately. The changed definition is used the next time that the function is invoked.

Invalidation of plans and packages: When an external function is altered, all the plans and packages that refer to that function are marked invalid.

LANGUAGE C and the PARAMETER VARCHAR clause: The ALTER FUNCTION statement does not allow you to alter a function's PARAMETER VARCHAR setting or its PARAMETER CCSID setting. However, you can alter a function's LANGUAGE. If the PARAMETER VARCHAR clause is specified during creation of a LANGUAGE C function, the catalog information that is recorded for that option is not removed by the ALTER FUNCTION statement if the LANGUAGE of the function is changed. If the function later reverts to LANGUAGE C, the setting of the PARAMETER VARCHAR option that is recorded in the catalog will be the setting that is specified during the CREATE FUNCTION statement.

Examples

Example 1: Assume that there are two functions CENTER in the PELLOW schema. The first function has two input parameters with INTEGER and FLOAT data types, respectively. The specific name for the first function is FOCUS1. The second function has three parameters with CHAR(25), DEC(5,2), and INTEGER data types.

Using the specific name to identify the function, change the WLM environment in which the first function runs from WLMENVNAME1 to WLMENVNAME2.

```
ALTER SPECIFIC FUNCTION ENGLS.FOCUS1 WLM ENVIRONMENT WLMENVNAME2;
```

Example 2: Change the second function that is described in *Example 1* so that it is not invoked when any of the arguments are null. Use the function signature to identify the function,

```
ALTER FUNCTION ENGLS.CENTER (CHAR(25), DEC(5,2), INTEGER)
    RETURNS NULL ON NULL INPUT;
```

You can also code the ALTER FUNCTION statement without the exact values for the CHAR and DEC data types:

```
ALTER FUNCTION ENGLS.CENTER (CHAR(), DEC(), INTEGER)
    RETURNS NULL ON NULL INPUT;
```

If you use empty parentheses, DB2 is to ignore the length, precision, and scale attributes when looking for matching data types to find the function.

ALTER FUNCTION (SQL scalar)

ALTER FUNCTION (SQL scalar)

The ALTER FUNCTION (SQL) statement changes the description of a user-defined SQL scalar function at the current server.

Invocation

This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is implicitly or explicitly specified.

Authorization

The privilege set defined below must include at least one of the following:

- Ownership of the function
- The ALTERIN privilege for the schema or all schemas
- SYSADM or SYSCTRL authority

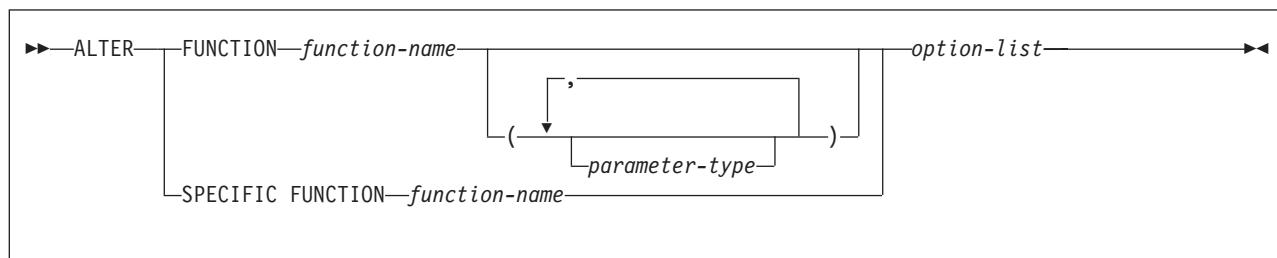
The authorization ID that matches the schema name implicitly has the ALTERIN privilege on the schema.

Privilege set: If the statement is embedded in an application program, the privilege set is the privileges that are held by the authorization ID of the owner of the plan or package.

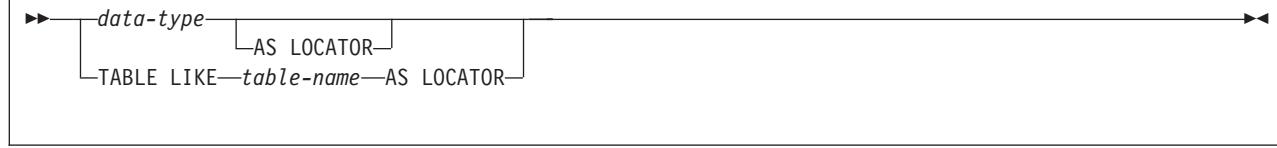
If the statement is dynamically prepared, the privilege set is the privileges that are held by the authorization IDs of the process. The specified function name can include a schema name (a qualifier). However, if the schema name is not the same as one of these authorization IDs, one of the following conditions must be met:

- The privilege set includes SYSADM or SYSCTRL authority.
- An authorization ID of the process has the ALTERIN privilege on the schema.

Syntax



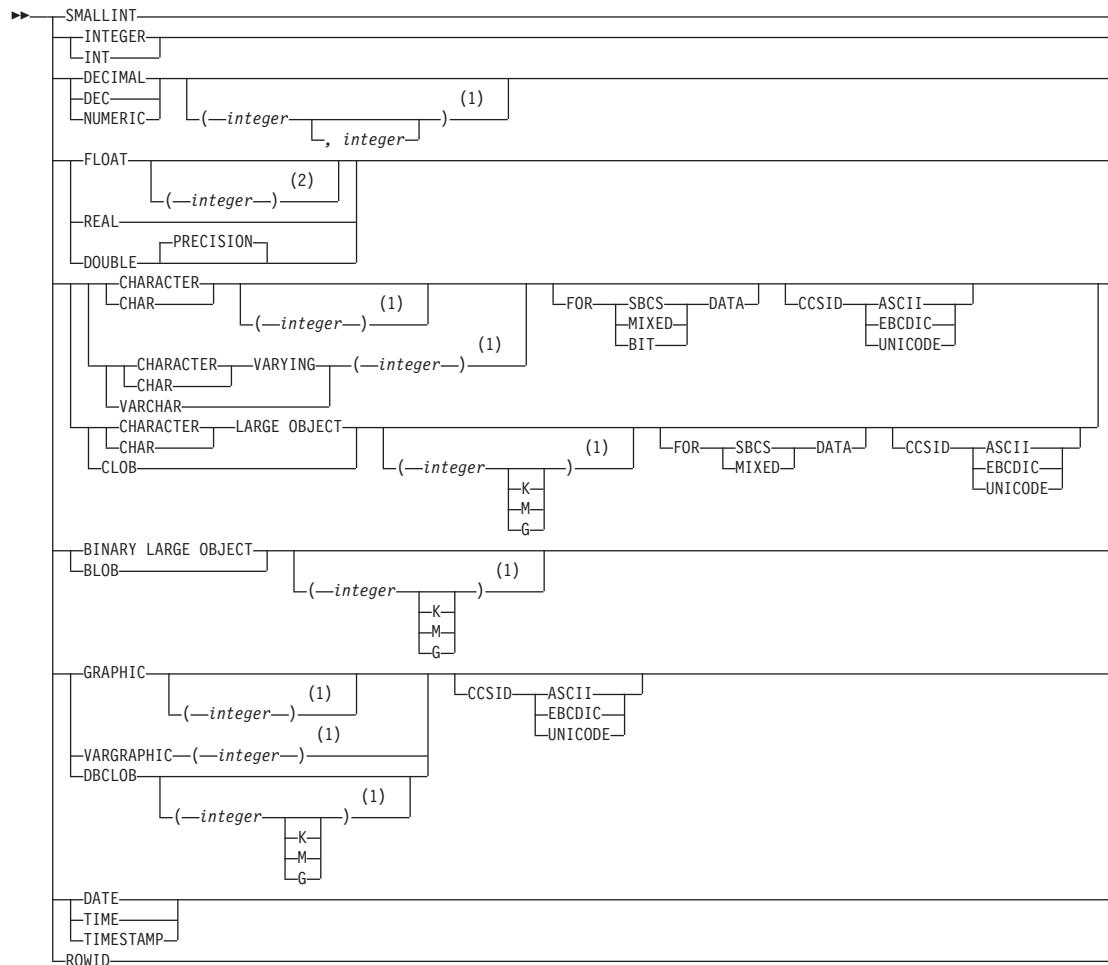
parameter-type:



data-type:

```

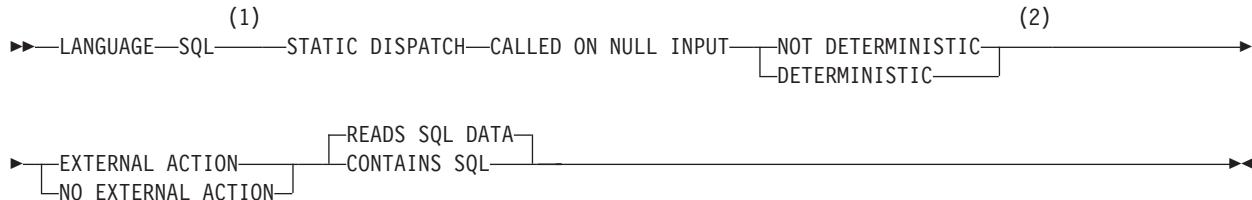
    >>> built-in-data-type
        | distinct-type-name
    
```

built-in-data-type:**Notes:**

- 1 The values that are specified for length, precision, or scale attributes must match the values that were specified when the function was created. Coding specific values is optional. Empty parentheses, (), can be used instead to indicate that DB2 is to ignore the attributes when determining whether data types match.
- 2 The value that is specified does not have to match the value that was specified when the function was created because matching is based on data type (REAL or DOUBLE). $1 \leq \text{integer} \leq 21$ indicates REAL and $22 \leq \text{integer} \leq 53$ indicates DOUBLE. Coding a specific value is optional. Empty parentheses cannot be used.

ALTER FUNCTION (SQL scalar)

option-list:



Notes:

- 1 This clause and the other clauses in the *option-list* can be specified in any order. However, the same clause cannot be specified more than once.
- 2 Synonyms for this clause include VARIANT for NOT DETERMINISTIC, and NOT VARIANT for DETERMINISTIC.

Description

One of the following three clauses identifies the function to be changed.

FUNCTION *function-name*

Identifies the SQL function by its function name. The name is implicitly or explicitly qualified with a schema name. If the name is not explicitly qualified, it is implicitly qualified with a schema name according to the following rules:

- If the statement is embedded in a program, the schema name is the authorization ID in the QUALIFIER bind option when the plan or package was created or last rebound. If QUALIFIER was not specified, the schema name is the owner of the plan or package.
- If the statement is prepared dynamically, the schema name is the SQL authorization ID in the CURRENT SQLID special register.

The identified function must be an SQL function. There must be exactly one function with *function-name* in the schema. The function can have any number of input parameters. If the schema does not contain a function with *function-name* or contains more than one function with this name, an error occurs.

FUNCTION *function-name* (*parameter-type*,...)

Identifies the SQL function by its function signature, which uniquely identifies the function. If *function-name()* is specified, the function that is identified must have zero parameters.

function-name

Gives the function name of the SQL function. If the function name is not qualified, it is implicitly qualified with a schema name as described in the preceding description for FUNCTION *function-name*.

(*parameter-type*,...)

Identifies the number of input parameters of the function and the data type of each parameter. The data type of each parameter must match the data type that was specified in the CREATE FUNCTION statement for the parameter in the corresponding position. The number of data types and the

logical concatenation of the data types are used to uniquely identify the function. Therefore, you cannot change the number of parameters or the data types of the parameters.

For data types that have a length, precision, or scale attribute, you can specify a value or use a set of empty parentheses:

- Empty parentheses indicate that DB2 is to ignore the attribute when determining whether the data types match.
- FLOAT cannot be specified with empty parentheses because its parameter value indicates different data types (REAL or DOUBLE).
- If you use a specific value for a length, precision, or scale attribute, the value must exactly match the value that was specified (implicitly or explicitly) in the CREATE FUNCTION statement.

The specific value for FLOAT(*n*) does not have to exactly match the defined value of the source function because 1<=n<= 21 indicates REAL and 22<=n<=53 indicates DOUBLE. Matching is based on whether the data type is REAL or DOUBLE.

- If length, precision, or scale is not explicitly specified, and empty parentheses are not specified, the default length of the data type is implied. For example:

CHAR	CHAR(1)
GRAPHIC	GRAPHIC(1)
DECIMAL	DECIMAL(5,0)
FLOAT	DOUBLE (length of 8)

The implicit length must exactly match the value that was specified (implicitly or explicitly) in the CREATE FUNCTION statement. For a complete list of the default lengths of data types, see “CREATE TABLE” on page 653.

For data types with a subtype or encoding scheme attribute, specifying the FOR DATA clause or CCSID clause is optional. Omission of either clause indicates that DB2 is to ignore the attribute when determining whether the data types match. If you specify either clause, it must match the value that was implicitly or explicitly specified in the CREATE FUNCTION statement.

See “CREATE FUNCTION” on page 529 for more information on the specification of the parameter list.

A function with the function signature must exist in the explicitly or implicitly specified schema; otherwise, an error occurs.

SPECIFIC FUNCTION *function-name*

Identifies the SQL function by its specific name. The name is implicitly or explicitly qualified with a schema name. A function with the specific name must exist in the schema; otherwise, an error occurs.

If the specific name is not qualified, it is implicitly qualified with a schema name as described in the preceding description for FUNCTION *function-name*.

The following clauses change the description of the function that has been identified to be changed.

LANGUAGE SQL

Specifies the application programming language in which the stored function is

ALTER FUNCTION (SQL scalar)

written. The value of the function is written as DB2 SQL in the *expression* of the RETURN clause in the CREATE FUNCTION statement. LANGUAGE SQL is the default.

NOT DETERMINISTIC or DETERMINISTIC

Specifies whether the function returns the same results for identical input arguments.

NOT DETERMINISTIC

The function might not return the same result for identical input arguments. The function depends on some state values that affect the results. DB2 uses this information to disable the merging of views and table expressions when processing SELECT, UPDATE, DELETE, or INSERT statements that reference this function. An example of a function that is not deterministic is one that generates random numbers.

NOT DETERMINISTIC must be specified explicitly or implicitly if the function program accesses a special register or invokes another non-deterministic function. NOT DETERMINISTIC is the default.

DETERMINISTIC

The function always returns the same result for identical input arguments. An example of a deterministic function is a function that calculates the square root of the input. DB2 uses this information to enable the merging of views and table expressions for SELECT, UPDATE, DELETE, or INSERT statements that reference this function. If applicable, specify DETERMINISTIC to prevent non-optimal access paths from being chosen for SQL statements that reference this function.

DB2 does not verify that the function program is consistent with the specification of DETERMINISTIC or NOT DETERMINISTIC.

EXTERNAL ACTION or NO EXTERNAL ACTION

Specifies whether the function takes an action that changes the state of an object that DB2 does not manage. An example of an external action is sending a message or writing a record to a file.

EXTERNAL ACTION

The function can take an action that changes the state of an object that DB2 does not manage.

Some SQL statements that invoke functions with external actions can result in incorrect results if parallel tasks execute the function. For example, if the function sends a note for each initial call to it, one note is sent for each parallel task instead of once for the function. Specify the DISALLOW PARALLEL clause for functions that do not work correctly with parallelism.

If you specify EXTERNAL ACTION, then DB2:

- Materializes the views and table expressions in SELECT, UPDATE, DELETE or INSERT statements that refer to the function. This materialization can adversely affect the access paths that are chosen for the SQL statements that reference this function. Do not specify EXTERNAL ACTION if the function does not have an external action.
- Does not move the function from one task control block (TCB) to another between FETCH operations.
- Does not allow another function or stored procedure to use the TCB until the cursor is closed. This is also applicable for cursors declared WITH HOLD.

The only changes to resources made outside of DB2 that are under the control of commit and rollback operations are those changes made under RRS control.

EXTERNAL ACTION must be specified implicitly or explicitly specified if the SQL routine body invokes a function that is defined with EXTERNAL ACTION.. EXTERNAL ACTION is the default.

NO EXTERNAL ACTION

The function does not take any action that changes the state of an object that DB2 does not manage. DB2 uses this information to enable the merging of views and table expressions for SELECT, UPDATE, DELETE, or INSERT statements that reference this function. If applicable, specify NO EXTERNAL ACTION to prevent non-optimal access paths from being chosen for SQL statements that reference this function.

DB2 does not verify that the function program is consistent with the specification of EXTERNAL ACTION or NO EXTERNAL ACTION.

READS SQL DATA or CONTAINS SQL

Indicates whether the function can execute any SQL statements and, if so, what type. DB2 verifies that the SQL issued by the function is consistent with this specification. See Table 72 on page 972 for a detailed list of the SQL statements that can be executed under each data access indication.

READS SQL DATA

The function does not execute SQL statements that modify data. SQL statements that are not supported in any function return a different error.

READS SQL DATA is the default.

CONTAINS SQL

The function does not execute SQL statements that read or modify data. SQL statements that are not supported in any function return a different error.

STATIC DISPATCH

At function resolution time, DB2 chooses a function based on the static (or declared) types of the function parameters. STATIC DISPATCH is the default.

CALLED ON NULL INPUT

The function is called regardless of whether any of the input arguments is null, making the function responsible for testing for null arguments. The function can return null. CALLED ON NULL INPUT is the default.

Notes

Changes are immediate: Any changes that the ALTER FUNCTION statement causes to the definition of an SQL function take effect immediately. The changed definition is used the next time that the function is invoked.

Invalidation of plans and packages: When an SQL function is altered, all the plans and packages that refer to that function are marked invalid.

Examples

Example 1: Modify the definition for an SQL function to indicate that the function is deterministic.

```
ALTER FUNCTION MY_UDF1 DETERMINISTIC;
```

ALTER INDEX

ALTER INDEX

The ALTER INDEX statement changes the description of an index at the current server.

Invocation

This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is implicitly or explicitly specified.

Authorization

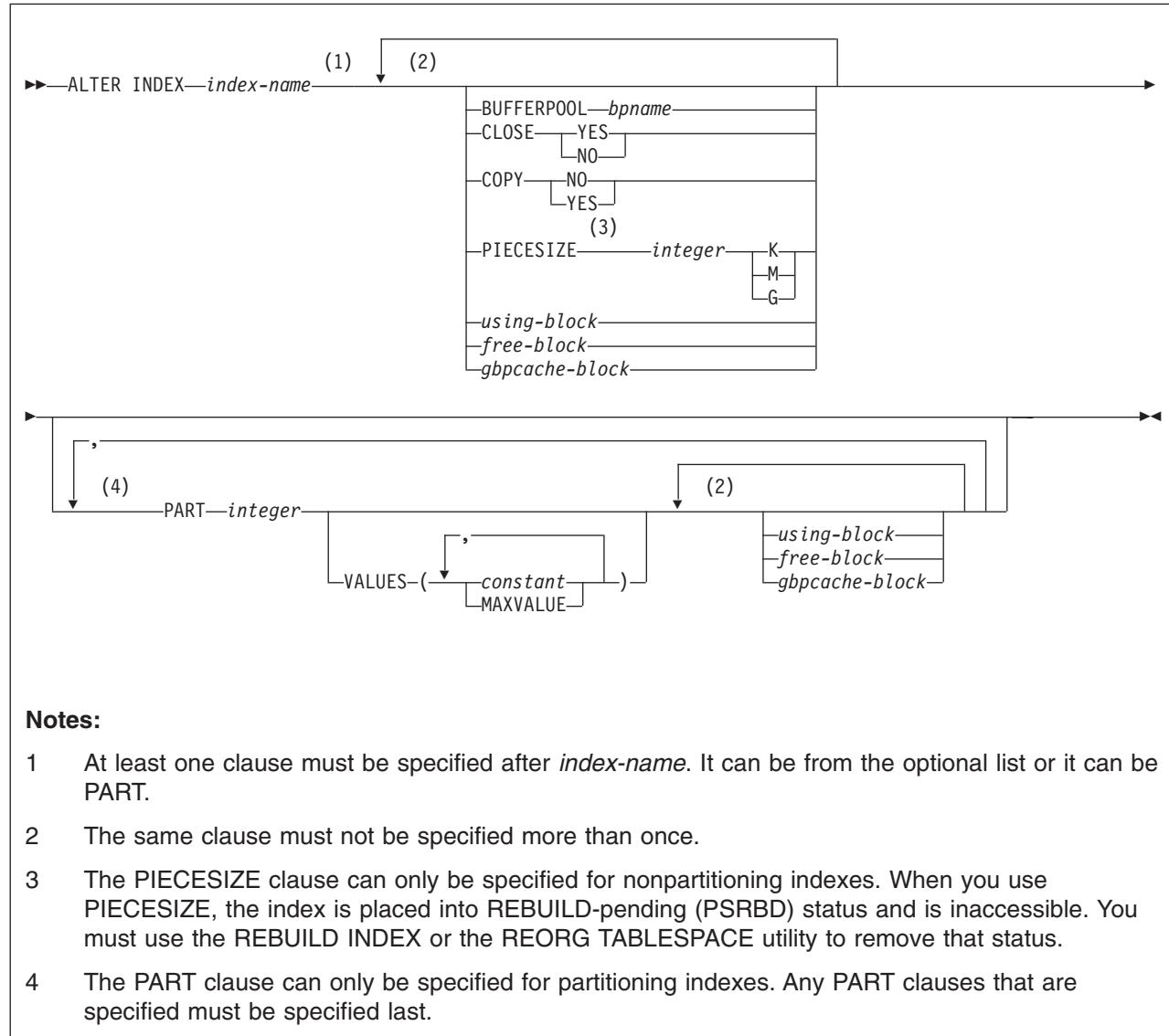
The privilege set that is defined below must include one of the following:

- Ownership of the index
- Ownership of the table on which the index is defined
- DBADM authority for the database that contains the table
- SYSADM or SYSCTRL authority

If BUFFERPOOL or USING STOGROUP is specified, additional privileges could be needed, as explained in the description of those clauses.

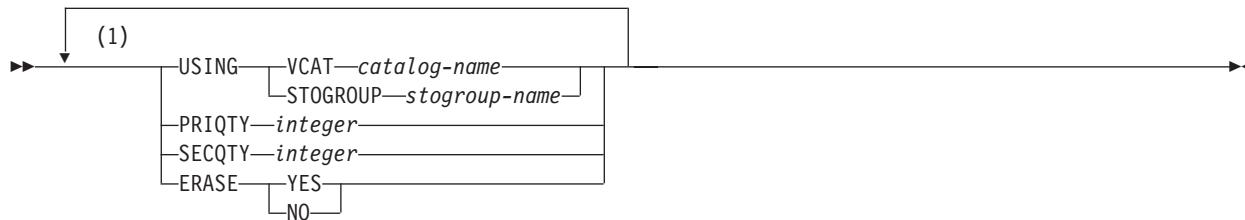
Privilege set: If the statement is embedded in an application program, the privilege set is the privileges that are held by the authorization ID of the owner of the plan or package. If the statement is dynamically prepared, the privilege set is the union of the privilege sets that are held by each authorization ID of the process.

Syntax



ALTER INDEX

using-block:



Notes:

- 1 The same clause must not be specified more than once.

free-block:



Notes:

- 1 The same clause must not be specified more than once.

gbpcache-block:



Description

index-name

Identifies the index to be altered. The name must identify a user-created index that exists at the current server. The name must not identify an index that is defined on a declared temporary table.

BUFFERPOOL *bpname*

Identifies the buffer pool to be used for the index. The *bpname* must identify an activated 4KB buffer pool, and the privilege set must include SYSADM authority, SYSCTRL authority, or the USE privilege for the buffer pool. See “Naming conventions” on page 34 for more details about *bpname*.

The change to the description of the index takes effect the next time the data sets of the index space are opened. The data sets can be closed and reopened

by a STOP DATABASE command to stop the index followed by a START DATABASE command to start the index.

In a data sharing environment, if you specify BUFFERPOOL, the index space must be in the stopped state when the ALTER INDEX statement is executed.

CLOSE

Specifies whether the data set is eligible to be closed when the index is not being used and the limit on the number of open data sets is reached. The change to the close rule takes effect the next time the data sets of the index space are opened.

YES

Eligible for closing.

NO

Not eligible for closing.

If DSMAX is reached and there are no CLOSE YES page sets to close, CLOSE NO page sets will be closed.

COPY

Indicates whether the COPY utility is allowed for the index.

NO

Does not allow full image or concurrent copies or the use of the RECOVER utility on the index.

YES

Allows full image or concurrent copies and the use the RECOVER utility on the index. For data sharing, changing COPY to YES causes additional SCA (Shared Communications Area) storage to be used until the next full image copy is taken or until COPY is set back to NO.

PIECESIZE *integer*

Specifies the maximum addressability of each piece (data set) for a nonpartitioning index.

Be very aware that when you alter the PIECESIZE value, the index is placed into page set REBUILD-pending (PSRBD) status. The entire index space becomes inaccessible. You must run the REBUILD INDEX or the REORG TABLESPACE utility to remove that status.

The subsequent keyword K, M, or G, indicates the units of the value specified in *integer*.

- K** Indicates that the *integer* value is to be multiplied by 1 024 to specify the maximum piece size in bytes. The integer must be a power of two between 256 and 67 108 864.
- M** Indicates that the *integer* value is to be multiplied by 1 048 576 to specify the maximum piece size in bytes. The integer must be a power of two between 1 and 65 536.
- G** Indicates that the *integer* value is to be multiplied by 1 073 741 824 to specify the maximum piece size in bytes. The integer must be a power of two between 1 and 64.

Table 41 on page 418 shows the valid values for piece size, which depend on the size of the table space.

#

|
|
|
|

ALTER INDEX

Table 41. Valid values of PIECESIZE clause

K units	M units	G units	Size attribute of table space
254 K	-	-	-
512 K	-	-	-
1024 K	1 M	-	-
2048 K	2 M	-	-
4096 K	4 M	-	-
8192 K	8 M	-	-
16384 K	16 M	-	-
32768 K	32 M	-	-
65536 K	64 M	-	-
131072 K	128 M	-	-
262144 K	256 M	-	-
524288 K	512 M	-	-
1048576 K	1024 M	1 G	-
2097152 K	2048 M	2 G	-
4194304 K	4096 M	4 G	LARGE, DSSIZE 4 G (or greater)
8388608 K	8192 M	8 G	DSSIZE 8 G (or greater)
16777216 K	16384 M	16 G	DSSIZE 16 G (or greater)
33554432 K	32768 M	32 G	DSSIZE 32 G (or greater)
67108864 K	65536 M	64 G	DSSIZE 64 G

using-block

The components of the *using-block* are discussed below, first for nonpartitioning indexes and then for partitioning indexes.

Using Block for Nonpartitioning Indexes

For nonpartitioning indexes, the USING clause specifies whether the data sets for the index are to be managed by the user or managed by DB2. The USING clause applies to every data set that can be used for the index.

If you specify USING, the index must be in the stopped state when the ALTER INDEX statement is executed. See “Altering storage attributes” on page 425 to determine how and when changes take effect.

VCAT catalog-name

Specifies a user-managed data set with a name that starts with the specified catalog name. You must specify the catalog name in the form of a short identifier. Thus, you must specify an alias if the name of the integrated catalog facility catalog is longer than eight characters. When the new description of the index is applied, the integrated catalog facility catalog must contain an entry for the data set that conforms to the DB2 naming conventions set forth in Part 2 (Volume 1) of *DB2 Administration Guide*.

One or more DB2 subsystems could share integrated catalog facility catalogs with the current server. To avoid the chance of having one of those subsystems attempt to assign the same name to different data sets, select a value for *catalog-name* that is not used by the other DB2 subsystems.

STOGROUP stogroup-name

Specifies using a DB2-managed data set that resides on a volume of the specified storage group. The stogroup name must identify a storage group that exists at the current server and the privilege set must include SYSADM authority, SYSCTRL authority, or the USE privilege for the storage group. When the new description of the index is applied, the description of the storage group must include at least one volume serial number. Each volume serial number must identify a volume that is accessible to MVS for dynamic allocation of the data set, and all identified volumes must be of the same device type. Furthermore, the integrated catalog facility catalog used for the storage group must not contain an entry for the data set.

If you specify USING STOGROUP and omit the PRIQTY, SECQTY, or ERASE clause, the implicit value of the omitted clause is its current value when the current data set is DB2-managed. If the current data set is being changed from being user-managed to DB2-managed, the implicit value of the omitted clause is its default value, which are:

- PRIQTY 12 when PRIQTY is omitted
- SECQTY 12 when PRIQTY and SECQTY are omitted. When SECQTY is omitted but PRIQTY is specified, SECQTY is either 10% of PRIQTY or 3 times the index page size (4K), whichever is larger.
- ERASE NO when ERASE is omitted

PRIQTY integer

Specifies the minimum primary space allocation for a DB2-managed data set. This clause can be specified only if the data set is currently managed by DB2 and USING VCAT is not specified.

If PRIQTY is specified, the primary space allocation is at least *n* kilobytes, where *n* is:

12 If *integer* is less than 12

ALTER INDEX

integer If *integer* is between 12 and 4194304
4194304 If *integer* is greater than 4194304

If USING STOGROUP is specified and PRIQTY is omitted, the value of PRIQTY is its current value. (However, if the current data set is being changed from being user-managed to DB2-managed, the value is its default value. See the description of USING STOGROUP.)

DB2 specifies the primary space allocation to access method services using the smallest multiple of 4KB not less than *n*. The allocated space can be greater than the amount of space requested by DB2. For example, it could be the smallest number of tracks that will accommodate the space requested. To more closely estimate the actual amount of storage, see the description of the DEFINE CLUSTER command in *DFSMS/MVS: Access Method Services for the Integrated Catalog*.

When determining a suitable value for PRIQTY, be aware that two of the pages of the primary space are used by DB2 for purposes other than storing index entries.

SECQTY *integer*

Specifies the minimum secondary space allocation for a DB2-managed data set. This clause can be specified only if the data set is currently managed by DB2 and USING VCAT is not specified.

If SECQTY is specified, the secondary space allocation is at least *n* kilobytes, where *n* is:

integer If *integer* is not greater than 4194304
4194304 If *integer* is greater than 4194304

If *integer* is 0, no data set for the index can be extended.

If USING STOGROUP is specified and SECQTY is omitted, the value of SECQTY is its current value. (However, if the current data set is being changed from being user-managed to DB2-managed, the value is its default value. See the description of USING STOGROUP.)

DB2 specifies the secondary space allocation to access method services using the smallest multiple of 4KB not less than *n*. The allocated space can be greater than the amount of space requested by DB2. For example, it could be the smallest number of tracks that will accommodate the space requested. To more closely estimate the actual amount of storage, see the description of the DEFINE CLUSTER command in *DFSMS/MVS: Access Method Services for the Integrated Catalog*.

ERASE

Indicates whether the DB2-managed data sets are to be erased when they are deleted during the execution of a utility or an SQL statement that drops the index. Refer to *DFSMS/MVS: Access Method Services for the Integrated Catalog* for more information.

NO

Does not erase the data sets. Operations involving data set deletion will perform better than ERASE YES. However, the data is still accessible, though not through DB2.

YES

Erases the data sets. As a security measure, DB2 overwrites all data in the data sets with zeros before they are deleted.

This clause can be specified only if the data set is currently managed by DB2 and USING VCAT is not specified. If you specify ERASE, the index must be in the stopped state when the ALTER INDEX statement is executed. See “Altering storage attributes” on page 425 to determine how and when changes take effect.

USING Block for Partitioning Indexes:

For a partitioning index, there is an optional PART clause for each partition. A *using-block* can be specified at the global level or at the partition level. A *using-block* within a PART clause applies only to that partition. A *using-block* specified before any PART clauses applies to every partition except those with a PART clause with a *using-block*.

For DB2-managed data sets, the values of PRIQTY, SECQTY, and ERASE for each partition are given by the first of these choices that applies:

- The values of PRIQTY, SECQTY, and ERASE given in the *using-block* within the PART clause for the partition. Do not use more than one *using-block* in any PART clause.
- The values of PRIQTY, SECQTY, and ERASE given in a *using-block* before any PART clauses
- The current values of PRIQTY, SECQTY, and ERASE

For data sets that are being changed from user-managed to DB2-managed, the values of PRIQTY, SECQTY, and ERASE for each partition are given by the first of these choices that applies:

- The values of PRIQTY, SECQTY, and ERASE given in the *using-block* within the PART clause for the partition. Do not use more than one *using-block* in any PART clause.
- The values of PRIQTY, SECQTY, and ERASE given in a *using-block* before any PART clauses
- The default values of PRIQTY, SECQTY, and ERASE, which are:
 - PRIQTY 12
 - SECQTY 12, if PRIQTY is not specified in either *using-block*, or 10% of PRIQTY or 3 times the index page size (whichever is larger) when PRIQTY is specified
 - ERASE NO

Any partition for which USING or ERASE is specified (either explicitly at the partition level or implicitly at the global level) must be in the stopped state when the ALTER INDEX statement is executed. See “Altering storage attributes” on page 425 to determine how and when changes take effect.

VCAT catalog-name

Specifies a user-managed data set with a name that starts with the specified catalog name. You must specify the catalog name in the form of a short identifier. Thus, you must specify an alias if the name of the integrated catalog facility catalog is longer than eight characters.

If *n* is the number of the partition, the identified integrated catalog facility catalog must already contain an entry for the *n*th data set of the index, conforming to the DB2 naming convention for data sets set forth in Part 2 (Volume 1) of *DB2 Administration Guide*.

ALTER INDEX

One or more DB2 subsystems could share integrated catalog facility catalogs with the current server. To avoid the chance of having one of those subsystems attempt to assign the same name to different data sets, select a value for *catalog-name* that is not used by the other DB2 subsystems.

DB2 assumes one and only one data set for each partition.

STOGROUP *stogroup-name*

If USING STOGROUP is used, *stogroup-name* must identify a storage group that exists at the current server and the privilege set must include SYSADM authority, SYSCTRL authority, or the USE privilege for the storage group.

DB2 assumes one and only one data set for each partition.

For information on the PRIQTY, SECQTY, and ERASE clauses, see the description of those clauses for nonpartitioning indexes.

End of using-block

free-block

FREEPAGE *integer*

Specifies how often to leave a page of free space when index entries are created as the result of executing a DB2 utility. One free page is left for every *integer* pages. The value of *integer* can range from 0 to 255. The change to the description of the index or partition has no effect until it is loaded or reorganized using a DB2 utility.

PCTFREE *integer*

Determines the percentage of free space to leave in each nonleaf page and leaf page when entries are added to the index or partition as the result of executing a DB2 utility. The first entry in a page is loaded without restriction. When additional entries are placed in a nonleaf or leaf page, the percentage of free space is at least as great as *integer*.

The value of *integer* can range from 0 to 99, however, if a value greater than 10 is specified, only 10 percent of free space will be left in nonleaf pages. The change to the description of the index or partition has no effect until it is loaded or reorganized using a DB2 utility.

If the index is partitioning, the values of FREEPAGE and PCTFREE for a particular partition are given by the first of these choices that applies:

- The values of FREEPAGE and PCTFREE given in the PART clause for that partition. Do not use more than one *free-block* in any PART clause.
- The values given in a *free-block* before any PART clauses.
- The current values of FREEPAGE and PCTFREE for that partition.

End of free-block

gbpcache-block

GBPCACHE

Specifies what index pages are written to the group buffer pool in a data sharing environment. In a non-data-sharing environment, you can specify this option, but it is ignored.

CHANGED

When there is inter-DB2 R/W interest on the index or partition, updated pages are written to the group buffer pool. When there is no inter-DB2 R/W interest, the group buffer pool is not used. Inter-DB2 R/W interest exists when more than one member in the data sharing group has the index or partition open, and at least one member has it open for update.

If the index is in a group buffer pool that is defined as GBPCACHE(NO), CHANGED is ignored and no pages are cached to the group buffer pool.

ALL

Indicates that pages are to be cached to the group buffer pool as they are read in from DASD, with one exception. When the page set is not GBP-dependent and one DB2 data sharing member has exclusive R/W interest in that page set (no other group members have any interest in the page set), no pages are cached in the group buffer pool.

Hiperpools are not used for indexes or partitions that are defined with GBPCACHE ALL.

If the index is in a group buffer pool that is defined as GBPCACHE(NO), ALL is ignored and no pages are cached to the group buffer pool.

NONE

Indicates that no pages are to be cached to the group buffer pool. DB2 uses the group buffer pool only for cross-validation.

If you specify NONE, the index or partition must not be in group buffer pool recover pending (GRECP) status.

If the index is partitioning, the value of GBPCACHE for a particular partition is given by the first of these choices that applies:

1. The value of GBPCACHE given in the PART clause for that partition. Do not use more than one *gbpcache-block* in any PART clause.
2. The value given in a *gbpcache-block* before any PART clauses.
3. The current value of GBPCACHE for that partition.

If you specify GBPCACHE in a data sharing environment, the index or partition must be in the stopped state when the ALTER INDEX statement is executed. You cannot alter the GBPCACHE value for certain indexes on DB2 catalog tables; for more information, see “SQL statements allowed on the catalog” on page 1011.

End of *gbpcache-block*

PART *integer*

Identifies a partition of the index. For an index that has *n* partitions, you must specify an integer in the range 1 to *n*. You must not use this clause if the index is nonpartitioned. You must use this clause if the index is partitioned and you specify the VALUES clause.

VALUES(*constant or MAXVALUE,...*)

Specifies the highest value of the index key for the identified partition of the partitioning index. In this context, highest means highest in the sorting sequences of the index columns. In a column defined as ascending (ASC), highest and lowest have their usual meanings. In a column defined as descending (DESC), the lowest actual value is highest in the sorting sequence.

```
#  
#  
#  
#  
#  
#  
#  
#
```

ALTER INDEX

You must use at least one value (*constant* or MAXVALUE) after VALUES in each PART clause. You can use as many values as there are columns in the key. The concatenation of all the values is the highest value of the key in the corresponding partition of the index.

constant
Specifies a constant value with a data type that is the same as its
corresponding column. If a string constant is longer or shorter than what
is required by the length attribute of its column, the constant is either
truncated or padded on the right to the required length. If the column is
ascending, the padding character is X'FF'. If the column is descending,
the padding character is X'00'. The precision and scale of a decimal
constant must not be greater than the precision and scale of its
corresponding column.

MAXVALUE
Specifies a value greater than the greatest possible value for the data
type of the column to which it corresponds (the value is all X'FF',
regardless of whether the column is ascending or descending). If all of
the columns in the partitioning key are ascending, a constant cannot be
specified following MAXVALUE; all subsequent columns must be
specified as MAXVALUE.

The key values are subject to the following rules:

- The first value corresponds to the first column of the key, the second value to the second column, and so on.
- If a key includes a ROWID column (or a column with a distinct type that is sourced on a ROWID data type), the values of the ROWID column are assumed to be in the range of X'000...00' to X'FFF...FF'. Only the first 17 bytes of the value that is specified for the corresponding ROWID column are considered.
- Using fewer values than there are columns in the key has the same effect as using the highest possible values for all omitted columns.
- The highest value of the key in any partition must be lower than the highest value of the key in the next partition.
- The highest value of the key in the last partition depends on how the table space is defined.

For table spaces that are created without the LARGE or DSSIZE option, the values that you specify after VALUES are not enforced. The highest value of the key that can be placed in the table is the highest possible value of the key.

For table spaces that are created with the LARGE or DSSIZE options, the values that you specify after VALUES are enforced. The value that is specified is the highest value of the key that can be placed in the table. Any keys that are made invalid after the ALTER statement is executed are placed in a discard data set when you run REORG. If the last partition is in REORG pending, regardless of whether you changed its limiting key values, you must specify a discard data set when you run REORG.

Notes

Running utilities: You cannot execute the ALTER INDEX statement while a DB2 utility has control of the index or its associated table space, unless the options being altered are PRIQTY or SECQTY.

Altering storage attributes: The USING, PRIQTY, SECQTY, and ERASE clauses define the storage attributes of the index or partition. However, if you specify the USING or ERASE clause when altering storage attributes, the index or partition must be in the stopped state when the ALTER INDEX statement is executed. A STOP DATABASE...SPACENAM... command can be used to stop the index or partition.

If the catalog name changes, the changes take effect after you move the data and start the index or partition using the START DATABASE...SPACENAM... command. The catalog name can be implicitly or explicitly changed by the ALTER INDEX statement. The catalog name also changes when you move the data to a different device. See the procedures for moving data in Part 2 (Volume 1) of *DB2 Administration Guide*.

Changes to the secondary space allocation (SECQTY) take effect the next time DB2 extends the data set; however, the new value is not reflected in the integrated catalog until you use the REORG, RECOVER, or LOAD REPLACE utility on the index or partition. Changes to the other storage attributes take effect the next time you use the REORG, RECOVER, or LOAD REPLACE utility on the index or partition. If you change the primary space allocation parameters or erase rule, you can have the changes take effect earlier if you move the data before you start the index or partition.

```
#  
#  
#  
#  
# Altering limit keys: If you specify PART and VALUES to change the limit key  
# values of a partitioning index, the plans and packages that are dependent on that  
# index are marked INVALID and go through automatic rebind the next time they are  
# run.
```

Altering indexes on DB2 catalog tables: For details on altering options on catalog tables, see “SQL statements allowed on the catalog” on page 1011.

Examples

Example 1: Alter the index DSN8710.XEMP1. Indicate that DB2 is not to close the data sets that support the index when there are no current users of the index.

```
ALTER INDEX DSN8710.XEMP1  
CLOSE NO;
```

Example 2: Alter the index DSN8710.XPROJ1. Use BP1 as the buffer pool that is to be associated with the index, indicate that full image or concurrent copies on the index are allowed, and change the maximum size of each data set to 8 megabytes.

```
ALTER INDEX DSN8710.XPROJ1  
BUFFERPOOL BP1  
COPY YES  
PIECESIZE 8M;
```

Example 3: Alter partitioned index DSN8710.DEPT1. For partition 3, leave one page of free space for every 13 pages and 13 percent of free space per page. For partition 5, leave one page for every 25 pages and 25 percent of free space. For all the other partitions, leave one page of free space for every 6 pages and 11 percent of free space. Ensure that index pages are cached to the group buffer pool for all partitions except partition 4. For partition 4, write pages only when there is inter-DB2 R/W interest on the partition.

```
ALTER INDEX DSN8710.XDEPT1  
BUFFERPOOL BP1  
CLOSE YES  
COPY YES
```

ALTER INDEX

```
USING VCAT CATLGG
FREEPAGE 6
PCTFREE 11
GBPCACHE ALL
PART 3
    USING VCAT CATLGG
    FREEPAGE 13
    PCTFREE 13,
PART 4
    USING VCAT CATLGG
    GBPCACHE CHANGED,
PART 5
    USING VCAT CATLGG
    FREEPAGE 25
    PCTFREE 25;
```

ALTER PROCEDURE (external)

The ALTER PROCEDURE statement changes the description of an external stored procedure at the current server.

Invocation

This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is implicitly or explicitly specified.

Authorization

The privilege set that is defined below must include at least one of the following:

- Ownership of the stored procedure
- The ALTERIN privilege for the schema or all schemas
- SYSADM or SYSCTRL authority

The authorization ID that matches the schema name implicitly has the ALTERIN privilege on the schema.

Privilege set: If the statement is embedded in an application program, the privilege set is the privileges that are held by the authorization ID of the owner of the plan or package.

If the statement is dynamically prepared, the privilege set is the privileges that are held by the authorization IDs of the process. The specified procedure name can include a schema name (a qualifier). However, if the schema name is not the same as one of these authorization IDs, one of the following conditions must be met:

- The privilege set includes SYSADM or SYSCTRL authority.
- An authorization ID of the process has the ALTERIN privilege on the schema.

If the environment in which the stored procedure is to run is being changed, the authorization ID must have authority to use the WLM environment or DB2-established stored procedure address space. This authorization is obtained from an external security product, such as RACF.

When LANGUAGE is JAVA and a *jar-name* is specified in the EXTERNAL NAME clause, the privilege set must include USAGE on the JAR, the Java ARchive file.

Syntax

```
►—ALTER PROCEDURE—procedure-name—option-list—►
```

ALTER PROCEDURE (external)

option-list:



Notes:

- 1 The clauses in the *option list* can be specified in any order. The same clause must not be specified more than once.
- 2 Synonyms include RESULT SET for DYNAMIC RESULT SET and RESULT SETS for DYNAMIC RESULT SETS.
- 3 If LANGUAGE is COMPJAVA or JAVA, EXTERNAL NAME must be specified with a '*string*' that is a valid *external-java-routine-name*. See the following figure.
- 4 Synonyms for the clause include STANDARD CALL for DB2SQL, SIMPLE CALL for GENERAL, and SIMPLE CALL WITH NULLS for GENERAL WITH NULLS.
- 5 Synonyms for the clause include VARIANT for NOT DETERMINISTIC, and NOT VARIANT is a synonym for DETERMINISTIC.
- 6 Synonyms include NULL CALL for CALLED ON NULL INPUT.

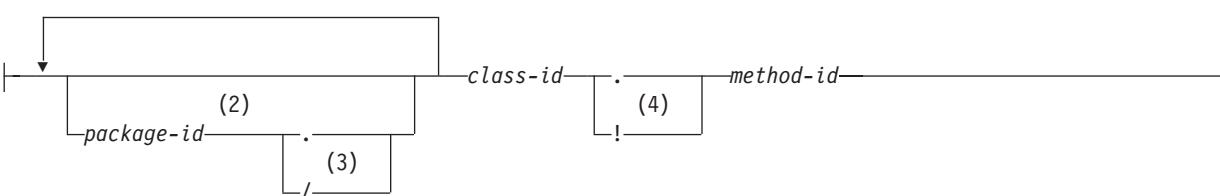
external-java-routine-name:



jar-name:



method-name:



method-signature:



Notes:

- 1 With LANGUAGE COMPJAVA, *jar-name* must not be specified.
- 2 With LANGUAGE COMPJAVA, at least one *package-id* must be specified.
- 3 The slash (/) is supported for compatibility with DB2 for OS/390 Version 5 and Version 6.
- 4 The exclamation point (!) is supported for compatibility with DB2 UWO.

Description

procedure-name

Identifies the stored procedure to be altered. The name is implicitly or explicitly qualified by a schema name. If the name is not explicitly qualified, it is implicitly qualified with a schema name according to the following rules:

- If the statement is embedded in a program, the schema name is the authorization ID in the QUALIFIER option when the plan or package was created or last rebound. If QUALIFIER was not specified, the schema name is the owner of the plan or package.
- If the statement is dynamically prepared, the schema name is the SQL authorization ID in the CURRENT SQLID special register.

ALTER PROCEDURE (external)

DYNAMIC RESULT SET *integer* or DYNAMIC RESULT SETS *integer*

Specifies the maximum number of query result sets that the stored procedure can return. The value must be between 0 and 32767.

EXTERNAL

Identifies the program that runs when the procedure name is specified in a CALL statement.

The program does not need to exist when the ALTER PROCEDURE statement is executed. However, it must exist and be accessible by the current server when a CALL statement to the stored procedure is issued.

You can specify the EXTERNAL clause in one of the following ways:

EXTERNAL

EXTERNAL NAME PKJVSP1

EXTERNAL NAME 'PKJVSP1'

If you specify an external program name, you must use the NAME keyword. For example, this syntax is not valid:

EXTERNAL PKJVSP1

NAME '*string*' or *identifier*

Identifies the user-written code that implements the stored procedure.

If LANGUAGE is COMPJAVA or JAVA, '*string*' must be specified and enclosed in single quotation marks, with no extraneous blanks within the single quotation marks. It must specify a valid *external-java-routine-name*. If multiple '*string*'s are specified, the total length of all of them must not be greater than 1305 bytes, and they must be separated by a space or a line break. Do not specify a JAR for a JAVA procedure for which NO SQL is in effect.

An *external-java-routine-name* contains the following parts:

jar-name

Identifies the name given to the JAR when it was installed in the database. The name contains *jar-id*, which can optionally be qualified with a schema. Examples are "myJar" and "mySchema.myJar." The unqualified *jar-id* is implicitly qualified with a schema name according to the following rules:

- If the statement is embedded in a program, the schema name is the authorization ID in the QUALIFIER bind option when the package or plan was created or last rebound. If the QUALIFIER was not specified, the schema name is the owner of the package or plan.
- If the statement is dynamically prepared, the schema name is the SQL authorization ID in the CURRENT SQLID special register.

If *jar-name* is specified, it must exist when the ALTER PROCEDURE statement is processed. With LANGUAGE COMPJAVA, a *jar-name* must not be specified.

If *jar-name* is not specified, the procedure is loaded from the class file directly instead of being loaded from a JAR file. DB2 for DB2 for OS/390 and z/OS searches the directories in the CLASSPATH associated with the WLM Environment. Environmental variables for Java routines are specified in a data set identified in a JAVAENV DD card on the JCL used to start the address space for a WLM-managed stored procedure.

method-name

Identifies the name of the method and must not be longer than 254 bytes. Its package, class, and method ID's are specific to Java and as such are not limited to 18 bytes. In addition, the rules for what these can contain are not necessarily the same as the rules for an SQL ordinary identifier.

package-id

Identifies the package list that the class identifier is part of. If the class is part of a package, the method name must include the complete package prefix, such as "myPacks.StoredProcs." The Java virtual machine looks in the directory "/myPacks/StoredProcs/" for the classes. With LANGUAGE COMPJAVA, at least one *package-id* must be specified.

class-id

Identifies the class identifier of the Java object.

method-id

Identifies the method identifier with the Java class to be invoked.

method-signature

Identifies a list of zero or more Java data types for the parameter list and must not be longer than 1024 bytes. Specify the *method-signature* if the procedure involves any input or output parameters that can be NULL. When the stored procedure being created is called, DB2 searches for a Java method with the exact *method-signature*. The number of *java-datatype* elements specified indicates how many parameters that the Java method must have.

A Java procedure can have no parameters. In this case, you code an empty set of parentheses for *method-signature*. If a Java *method-signature* is not specified, DB2 searches for a Java method with a signature derived from the default JDBC types associated with the SQL types specified in the parameter list of the ALTER PROCEDURE statement.

For other values of LANGUAGE, the name can be a string constant that is no longer than 8 characters or a short identifier. It must conform to the naming conventions for MVS load modules. Alphabetical extenders for national languages can be used as the first character and as subsequent characters in the load module name.

If you do not specify the NAME clause, *NAME procedure-name* is implicit. In this case procedure-name must not be longer than 8 characters, and LANGUAGE must not be COMPJAVA or JAVA.

LANGUAGE

Specifies the application programming language in which the stored procedure is written. Assembler, C, COBOL, and PL/I programs must be designed to run in IBM's Language Environment.

ASSEMBLE

The stored procedure is written in Assembler.

C The stored procedure is written in C or C++.

COBOL

The stored procedure is written in COBOL, including the OO-COBOL language extensions.

ALTER PROCEDURE (external)

COMPJAVA

The stored procedure is written in Java and the Java byte code has been bound into a PDSE member using the Visual Age for Java ET/390 byte code binder. When LANGUAGE COMPJAVA is specified, the EXTERNAL NAME clause must also be specified with a valid *external-java-routine-name*.

JAVA

The stored procedure is written in Java byte code and is executed in the OS/390 Java Virtual Machine. When LANGUAGE JAVA is specified, the EXTERNAL NAME clause must also be specified with a valid *external-java-routine-name* and PARAMETER STYLE must be specified with JAVA.

Do not specify LANGUAGE JAVA when DBINFO, NO WLM ENVIRONMENT, PROGRAM TYPE MAIN, or RUN OPTIONS is in effect.

PLI

The stored procedure is written in PL/I.

REXX

The stored procedure is written in REXX. Do not specify LANGUAGE REXX when PARAMETER STYLE DB2SQL or NO WLM ENVIRONMENT is in effect.

PARAMETER STYLE

Identifies the linkage convention used to pass parameters to the stored procedure. All of the linkage conventions provide arguments to the stored procedure that contain the parameters specified on the CALL statement. Some of the linkage conventions pass additional arguments to the stored procedure that provide more information to the stored procedure. For more information on linkage conventions, see *DB2 Application Programming and SQL Guide*.

DB2SQL

In addition to the parameters on the CALL statement, the following arguments are also passed to the stored procedure:

- A null indicator for each parameter on the CALL statement
- The SQLSTATE to be returned to DB2
- The qualified name of the stored procedure
- The specific name of the stored procedure
- The SQL diagnostic string to be returned to DB2

If DBINFO is specified, an additional parameter, the DB2INFO structure, might also be passed.

Do not specify DB2 SQL when LANGUAGE REXX is in effect.

GENERAL

Only the parameters on the CALL statement are passed to the stored procedure. The parameters cannot be null.

GENERAL WITH NULLS

In addition to the parameters on the CALL statement, another argument is also passed to the stored procedure. The additional argument contains a vector of null indicators for each of the parameters on the CALL statement that enables the stored procedure to accept or return null parameter values.

JAVA

The stored procedure uses a convention for passing parameters that conforms to the Java and SQLJ specifications. PARAMETER STYLE JAVA can be specified only if LANGUAGE is COMPJAVA or JAVA. If the ALTER

PROCEDURE statement results in changing LANGUAGE to JAVA, PARAMETER STYLE JAVA and an EXTERNAL NAME clause might need to be specified to provide appropriate values. If LANGUAGE is COMPJAVA, PARAMETER STYLE defaults to JAVA. JAVA must be specified for PARAMETER STYLE when LANGUAGE is JAVA.

INOUT and OUT parameters are passed as single-entry arrays. The INOUT and OUT parameters are declared in the Java method as single-element arrays of the Java type.

For REXX stored procedures (LANGUAGE REXX), GENERAL and GENERAL WITH NULLS are the only valid values for PARAMETER STYLE; therefore, specify one of these values and do not allow PARAMETER STYLE to default to DB2SQL.

NOT DETERMINISTIC or DETERMINISTIC

Specifies whether the stored procedure returns the same result from successive calls with identical input arguments.

NOT DETERMINISTIC

The stored procedure might not return the same result from successive calls with identical input arguments.

DETERMINISTIC

The stored procedure returns the same result from successive calls with identical input arguments.

DB2 does not verify that the stored procedure code is consistent with the specification of DETERMINISTIC or NOT DETERMINISTIC.

NO SQL, MODIFIES SQL DATA, READS SQL DATA, or CONTAINS SQL DATA

Indicates whether the stored procedure issues any SQL statements and, if so, what type. DB2 verifies that the SQL issued by the function is consistent with this specification. See Table 72 on page 972 for a detailed list of the SQL statements that can be executed under each data access indication.

NO SQL

The stored procedure does not execute SQL statements. Do not specify NO SQL for a JAVA procedure that uses a JAR.

MODIFIES SQL DATA

The stored procedure can execute any SQL statement except those statements that are not supported in any stored procedure.

READS SQL DATA

The stored procedure cannot execute SQL statements that modify data. SQL statements that are not supported in any stored procedure return a different error.

CONTAINS SQL

The stored procedure cannot execute any SQL statements that read or modify data. SQL statements that are not supported in any stored procedure return a different error.

NO DBINFO or DBINFO

Specifies whether specific information known by DB2 is passed to the stored procedure when it is invoked.

NO DBINFO

Additional information is not passed.

ALTER PROCEDURE (external)

DBINFO

An additional argument is passed when the stored procedure is invoked. The argument is a structure that contains information such as the application run-time authorization ID, the schema name, the name of a table or column that the procedure might be inserting into or updating, and identification of the database server that invoked the procedure. For details about the argument and its structure, see *DB2 Application Programming and SQL Guide*.

| DBINFO can be specified only if PARAMETER STYLE DB2SQL is in effect.

NO COLLID or COLLID *collection-id*

Identifies the package collection that is to be used when the stored procedure is executed. This is the package collection into which the DBRM that is associated with the stored procedure is bound.

NO COLLID

Specifies that the package collection for the stored procedure is the same as the package collection of the calling program. If the calling program does not use a package, the package collection is set to the value of the CURRENT PACKAGESET special register.

COLLID *collection-id*

Identifies the package collection that is to be used when the stored procedure is executed. It is the name of the package collection into which the DBRM associated with the stored procedure is bound.

For REXX stored procedures, *collection-id* can be DSNREXRR, DSNREXRS, DSNREXCR, or DSNREXCS.

WLM ENVIRONMENT

Identifies the MVS workload manager (WLM) environment in which the stored procedure is to run when the DB2 stored procedure address space is WLM-established. The *name* of the WLM environment is a long identifier.

name

The WLM environment in which the stored procedure must run. If another stored procedure or a user-defined function calls the stored procedure and that calling routine is running in an address space that is not associated with the specified WLM environment, DB2 routes the stored procedure request to a different MVS address space.

(*name*,*)

When the stored procedure is called directly by an SQL application program, the WLM environment in which the stored procedure runs.

If another stored procedure or a user-defined function calls the stored procedure, the stored procedure runs in the same WLM environment that the calling routine uses.

To change the environment in which the procedure is to run, you must have appropriate authority for the WLM environment. For an example of a RACF command that provides this authorization, see “Running stored procedures” on page 634.

NO WLM ENVIRONMENT

Indicates that the stored procedure is to run in the DB2-established stored procedure address space.

Do not specify NO WLM ENVIRONMENT if the definition of the stored procedure implicitly or explicitly includes the following clauses or parameters:

- The PROGRAM TYPE SUB clause
- The SECURITY USER or SECURITY DEFINER clause
- The LANGUAGE REXX, LANGUAGE COMPJAVA, or JAVA clause
- Parameters with a LOB data type or a distinct type based on a LOB data type

To change the procedure to run in the DB2-established stored procedure address space, you must have appropriate authority for the DB2-established stored procedure address space. For an example of a RACF command that provides this authorization, see “Running stored procedures” on page 634.

ASUTIME

Specifies the total amount of processor time, in CPU service units, that a single invocation of a stored procedure can run. The value is unrelated to the ASUTIME column in the resource limit specification table.

When you are debugging a stored procedure, setting a limit can be helpful in case the stored procedure gets caught in a loop. For information on CPU service units, see *OS/390 MVS Initialization and Tuning Guide*.

NO LIMIT

There is no limit on the service units.

LIMIT *integer*

The limit on the service units is a positive *integer* in the range of 1 to 2G. If the stored procedure uses more service units than the specified value, DB2 cancels the stored procedure.

STAY RESIDENT

Specifies whether the stored procedure load module is to remain resident in memory when the stored procedure ends.

NO

The load module is deleted from memory after the stored procedure ends. Use NO for non-reentrant stored procedures.

YES

The load module remains resident in memory after the stored procedure ends.

PROGRAM TYPE

Specifies whether the stored procedure runs as a main routine or a subroutine.

SUB

The stored procedure runs as a subroutine.

Do not specify PROGRAM TYPE SUB for stored procedures with a LANGUAGE value of REXX. Do not specify PROGRAM TYPE SUB when NO WLM ENVIRONMENT is in effect.

MAIN

The stored procedure runs as a main routine.

Do not specify PROGRAM TYPE MAIN when LANGUAGE JAVA is in effect.

SECURITY

Specifies how the stored procedure interacts with an external security product, such as RACF, to control access to non-SQL resources.

DB2

The stored procedure does not require a special external security environment. If the stored procedure accesses resources that an external

ALTER PROCEDURE (external)

security product protects, the access is performed using the authorization ID associated with the stored procedure address space.

SECURITY DB2 is the only valid choice when NO WLM ENVIRONMENT is in effect.

USER

An external security environment should be established for the stored procedure. If the stored procedure accesses resources that the external security product protects, the access is performed using the authorization ID of the user who invoked the stored procedure.

|
| Do not specify SECURITY USER when NO WLM ENVIRONMENT is in effect.

DEFINER

An external security environment should be established for the stored procedure. If the stored procedure accesses resources that the external security product protects, the access is performed using the authorization ID of the owner of the stored procedure.

|
| Do not specify SECURITY DEFINER when NO WLM ENVIRONMENT is in effect.

RUN OPTIONS *run-time-options*

Specifies the Language Environment run-time options to be used for the stored procedure. For a REXX stored procedure, specifies the Language Environment run-time options to be passed to the REXX language interface to DB2. You must specify *run-time-options* as a character string that is no longer than 254 bytes. To replace any existing run-time options with no options, specify an empty string with RUN OPTIONS. When you specify an empty string, DB2 does not pass any run-time options to Language Environment, and Language Environment uses its installation defaults. For a description of the Language Environment run-time options, see *OS/390 Language Environment for OS/390 & VM Programming Reference*.

|
| Do not specify RUN OPTIONS when LANGUAGE COMPJAVA or LANGUAGE JAVA is in effect.

COMMIT ON RETURN

Indicates whether DB2 is to commit the transaction immediately on return from the stored procedure.

NO

DB2 does not issue a commit when the stored procedure returns.

YES

DB2 issues a commit when the stored procedure returns if the following statements are true:

- The SQLCODE that is returned by the CALL statement is not negative.
- The stored procedure is not in a must abort state.

The commit operation includes the work that is performed by the calling application process and the stored procedure.

If the stored procedure returns result sets, the cursors that are associated with the result sets must have been defined WITH HOLD to be usable after the commit.

INHERIT SPECIAL REGISTERS

Indicates that values of special registers are inherited according to the rules listed in the table for characteristics of special registers in a stored procedure in Table 19 on page 93.

DEFAULT SPECIAL REGISTERS

Indicates that special registers are initialized to the default values, as indicated by the rules in the table for characteristics of special registers in a stored procedure in Table 19 on page 93.

CALLED ON NULL INPUT

Specifies that the stored procedure will be called even if any of the input arguments is null, making the procedure responsible for testing for null argument values. The result is the null value.

Notes

Changes are immediate: Any changes that the ALTER PROCEDURE statement cause to the definition of a procedure take effect immediately. The changed definition is used the next time that the procedure is called.

Invalidation of plans and packages: When an external procedure is altered, all the plans and packages that refer to that procedure are marked invalid.

Restrictions for nested stored procedures: A stored procedure, user-defined function, or trigger cannot call a stored procedure that is defined with the COMMIT ON RETURN clause.

LANGUAGE C and the PARAMETER VARCHAR clause: The ALTER PROCEDURE statement does not allow you to alter a procedure's PARAMETER VARCHAR setting or its PARAMETER CCSID setting. However, you can alter a procedure's LANGUAGE. If the PARAMETER VARCHAR clause is specified during creation of a LANGUAGE C procedure, the catalog information that is recorded for that option is not removed by the ALTER PROCEDURE statement if the LANGUAGE of the procedure is changed. If the procedure later reverts to LANGUAGE C, the setting of the PARAMETER VARCHAR option that is recorded in the catalog will be the setting that is specified during the CREATE PROCEDURE statement.

Example

Assume that stored procedure SYSPROC.MYPROC is currently defined to run in WLM environment PARTSA and that you have appropriate authority on that WLM environment and WLM environment PARTSEC. Change the definition of the stored procedure so that it runs in PARTSEC.

```
ALTER PROCEDURE SYSPROC.MYPROC WLM ENVIRONMENT PARTSEC;
```

ALTER PROCEDURE (SQL)

ALTER PROCEDURE (SQL)

The ALTER PROCEDURE statement changes the description of an SQL stored procedure at the current server.

Invocation

This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is implicitly or explicitly specified.

Authorization

The privilege set that is defined below must include at least one of the following:

- Ownership of the stored procedure
- The ALTERIN privilege for the schema or all schemas
- SYSADM or SYSCTRL authority

The authorization ID that matches the schema name implicitly has the ALTERIN privilege on the schema.

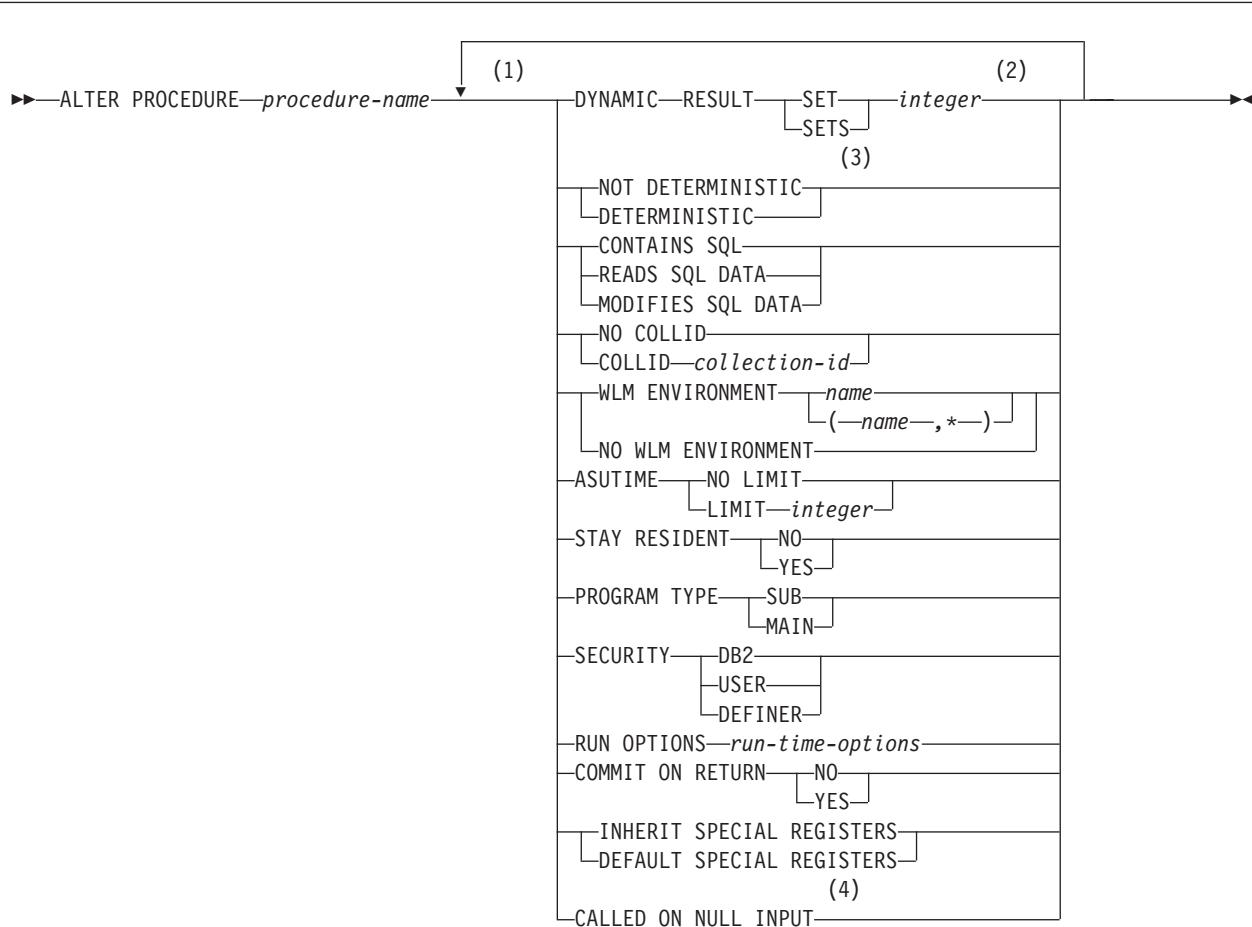
Privilege set: If the statement is embedded in an application program, the privilege set is the privileges that are held by the authorization ID of the owner of the plan or package.

If the statement is dynamically prepared, the privilege set is the privileges that are held by the authorization IDs of the process. The specified procedure name can include a schema name (a qualifier). However, if the schema name is not the same as one of these authorization IDs, one of the following conditions must be met:

- The privilege set includes SYSADM or SYSCTRL authority.
- An authorization ID of the process has the ALTERIN privilege on the schema.

The authorization ID used to alter the stored procedure definition must have appropriate authority for the specified environment (WLM environment or DB2-established stored procedure address space) in which the procedure is currently defined to run. This authorization is obtained from an external security product, such as RACF.

Syntax



Notes:

- 1 The same clause must not be specified more than once.
- 2 Synonyms include RESULT SET for DYNAMIC RESULT SET and RESULT SETS for DYNAMIC RESULT SETS.
- 3 Synonyms for the clause include VARIANT for NOT DETERMINISTIC, and NOT VARIANT is a synonym for DETERMINISTIC.
- 4 Synonyms include NULL CALL for CALLED ON NULL INPUT.

Description

procedure-name

Identifies the stored procedure to be altered. The name is implicitly or explicitly qualified by a schema. If the name is not explicitly qualified, it is implicitly qualified with a schema name according to the following rules:

- If the statement is embedded in a program, the schema name is the authorization ID in the `QUALIFIER` bind option when the plan or package was created or last rebound. If `QUALIFIER` was not specified, the schema name is the owner of the plan or package.

ALTER PROCEDURE (SQL)

- If the statement is dynamically prepared, the schema name is the SQL authorization ID in the CURRENT SQLID special register.

DYNAMIC RESULT SET *integer* or DYNAMIC RESULT SETS *integer*

Specifies the maximum number of query result sets that the stored procedure can return. The value must be between 0 and 32767.

NOT DETERMINISTIC or DETERMINISTIC

Specifies whether the stored procedure returns the same result from successive calls with identical input arguments.

NOT DETERMINISTIC

The stored procedure might not return the same result from successive calls with identical input arguments.

DETERMINISTIC

The stored procedure returns the same result from successive calls with identical input arguments.

DB2 does not verify that the stored procedure code is consistent with the specification of DETERMINISTIC or NOT DETERMINISTIC.

MODIFIES SQL DATA, READS SQL DATA, or CONTAINS SQL DATA

Indicates whether the stored procedure can execute any SQL statements and, if so, what type. DB2 verifies that the SQL issued by the function is consistent with this specification. See Table 72 on page 972 for a detailed list of the SQL statements that can be executed under each data access indication.

MODIFIES SQL DATA

The stored procedure can execute any SQL statement except those statements that are not supported in any stored procedure.

READS SQL DATA

The stored procedure does not execute SQL statements that modify data. SQL statements that are not supported in any stored procedure return a different error.

CONTAINS SQL

The stored procedure does not execute any SQL statements that read or modify data. SQL statements that are not supported in any stored procedure return a different error.

NO COLLID or COLLID *collection-id*

Identifies the package collection that is to be used when the stored procedure is executed. This is the package collection into which the DBRM that is associated with the stored procedure is bound.

NO COLLID

Indicates that the package collection for the stored procedure is the same as the package collection of the calling program. If the calling program does not use a package, the package collection is set to the value of special register CURRENT PACKAGESET.

COLLID *collection-id*

Indicates the package collection for the stored procedure is the one specified.

WLM ENVIRONMENT

Identifies the MVS workload manager (WLM) environment in which the stored procedure is to run when the DB2 stored procedure address space is WLM-established. The *name* of the WLM environment is a long identifier.

name

The WLM environment in which the stored procedure must run. If another stored procedure or a user-defined function calls the stored procedure and that calling routine is running in an address space that is not associated with the specified WLM environment, DB2 routes the stored procedure request to a different MVS address space.

*(name, *)*

When an SQL application program directly calls a stored procedure, the WLM environment in which the stored procedure runs.

If another stored procedure or a user-defined function calls the stored procedure, the stored procedure runs in the same WLM environment that the calling routine uses.

To change the environment in which the stored procedure is to run, you must have appropriate authority for the WLM environment. For an example of a RACF command that provides this authorization, see “Running stored procedures” on page 634.

NO WLM ENVIRONMENT

Indicates that the stored procedure is to run in the DB2-established stored procedure address space.

Do not specify NO WLM ENVIRONMENT if you implicitly or explicitly define the stored procedure with the SECURITY USER or SECURITY DEFINER clause.

To change the procedure to run in the DB2-established stored procedure address space, you must have appropriate authority for the DB2-established stored procedure address space. For an example of a RACF command that provides this authorization, see “Running stored procedures” on page 634.

ASUTIME

Specifies the total amount of processor time, in CPU service units, that a single invocation of a stored procedure can run. The value is unrelated to the ASUTIME column of the resource limit specification table.

When you are debugging a stored procedure, setting a limit can be helpful in case the stored procedure gets caught in a loop. For information on service units, see *OS/390 MVS Initialization and Tuning Guide*.

NO LIMIT

There is no limit on the service units.

LIMIT *integer*

The limit on the service units is a positive *integer* in the range of 1 to 2 G. If the stored procedure uses more service units than the specified value, DB2 cancels the stored procedure.

STAY RESIDENT

Specifies whether the stored procedure load module is to remain resident in memory when the stored procedure ends.

NO

The load module is deleted from memory after the stored procedure ends.

YES

The load module remains resident in memory after the stored procedure ends.

PROGRAM TYPE

Specifies whether the stored procedure runs as a main routine or a subroutine.

|
|

ALTER PROCEDURE (SQL)

SUB

The stored procedure runs as a subroutine.

MAIN

The stored procedure runs as a main routine.

SECURITY

Specifies how the stored procedure interacts with an external security product, such as RACF, to control access to non-SQL resources.

DB2

The stored procedure does not require a special external security environment. If the stored procedure accesses resources that an external security product protects, the access is performed using the authorization ID associated with the stored procedure address space.

USER

An external security environment should be established for the stored procedure. If the stored procedure accesses resources that the external security product protects, the access is performed using the authorization ID of the user who invoked the stored procedure.

DEFINER

An external security environment should be established for the stored procedure. If the stored procedure accesses resources that the external security product protects, the access is performed using the authorization ID of the owner of the stored procedure.

RUN OPTIONS *run-time-options*

Specifies the Language Environment run-time options to be used for the stored procedure. You must specify *run-time-options* as a character string that is no longer than 254 bytes. If you do not specify RUN OPTIONS or pass an empty string, DB2 does not pass any run-time options to Language Environment, and Language Environment uses its installation defaults.

For a description of the Language Environment run-time options, see *OS/390 Language Environment for OS/390 & VM Programming Reference*.

COMMIT ON RETURN

Indicates whether DB2 commits the transaction immediately on return from the stored procedure.

NO

DB2 does not issue a commit when the stored procedure returns.

YES

DB2 issues a commit when the stored procedure returns if the following statements are true:

- The SQLCODE that is returned by the CALL statement is not negative.
- The stored procedure is not in a must abort state.

The commit operation includes the work that is performed by the calling application process and the stored procedure.

If the stored procedure returns result sets, the cursors that are associated with the result sets must have been defined as WITH HOLD to be usable after the commit.

INHERIT SPECIAL REGISTERS

Indicates that special registers should be inherited according to the rules listed in the table for characteristics of special registers in a stored procedure function in Table 19 on page 93.

DEFAULT SPECIAL REGISTERS

Indicates that special registers should be initialized to the default values, as indicated by the rules in the table for characteristics of special registers in a stored procedure function in Table 19 on page 93.

CALLED ON NULL INPUT

Specifies that the stored procedure will be called even if any of the input arguments is null, making the procedure responsible for testing for null argument values. The result is the null value.

Notes

Changes are immediate: Any changes that the ALTER PROCEDURE statement cause to the definition of a procedure take effect immediately. The changed definition is used the next time that the procedure is called.

Invalidation of plans and packages: When an SQL procedure is altered, all the plans and packages that refer to that procedure are marked invalid.

Restrictions for nested stored procedures: A stored procedure, user-defined function, or trigger cannot call a stored procedure that is defined with the COMMIT ON RETURN clause.

Example

Modify the definition for an SQL procedure so that SQL changes are committed on return from the SQL procedure and the SQL procedure runs in the WLM environment named WLMSQLP.

```
ALTER PROCEDURE UPDATE_SALARY_1
  COMMIT ON RETURN YES
  WLM ENVIRONMENT WLMSQLP;
```

ALTER STOGROUP

ALTER STOGROUP

The ALTER STOGROUP statement changes the description of a storage group at the current server.

Invocation

This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is implicitly or explicitly specified.

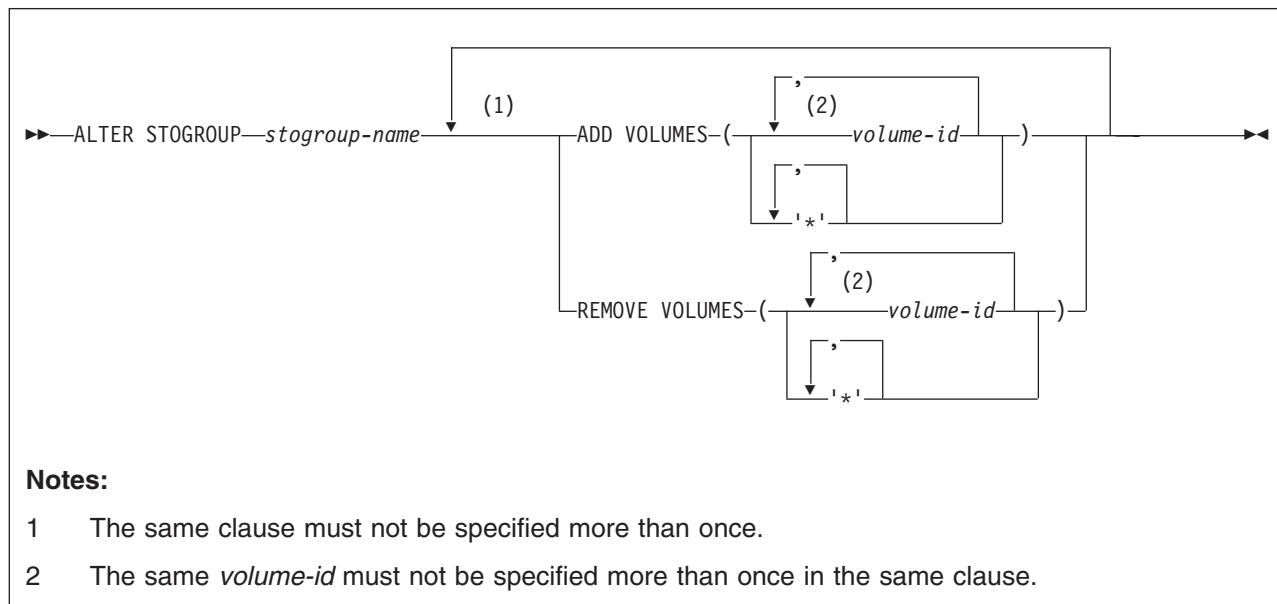
Authorization

The privilege set that is defined below must include one of the following:

- Ownership of the storage group
- SYSADM or SYSCTRL authority

Privilege set: If the statement is embedded in an application program, the privilege set is the privileges that are held by the authorization ID of the owner of the plan or package. If the statement is dynamically prepared, the privilege set is the union of the privilege sets that are held by each authorization ID of the process.

Syntax



Description

stogroup-name

Identifies the storage group to be altered. The name must identify a storage group that exists at the current server.

ADD VOLUMES(*volume-id*,...) or **ADD VOLUMES('*,...)**

Adds volumes to the storage group. Each *volume-id* is the volume serial number of a storage volume to be added. It can have a maximum of six characters and is specified as an identifier or a string constant.

A *volume-id* must not be specified if any volume of the storage group is designated by an asterisk (*). An asterisk must not be specified if any volume of the storage group is designated by a *volume-id*.

You cannot add a volume that is already in the storage group unless you first remove it with REMOVE VOLUMES.

Asterisks are recognized only by Storage Management Subsystem (SMS). To allow SMS control over volume selection, define DB2 STOGROUPs with VOLUMES('*,...'). SMS usage is recommended, rather than using DB2 to allocate data to specific volumes. Having DB2 select the volume requires non-SMS usage or assigning an SMS Storage Class with guaranteed space. However, because guaranteed space reduces the benefits of SMS allocation, it is not recommended.

If you do choose to use specific volume assignments, additional manual space management must be performed. Free space must be managed for each individual volume to prevent failures during the initial allocation and extension. This process generally requires more time for space management and results in more space shortages. Guaranteed space should be used only where the space needs are relatively small and do not change.

REMOVE VOLUMES(*volume-id*,...) or REMOVE VOLUMES('*,...')

Removes volumes from the storage group. Each *volume-id* is the volume serial number of a storage volume to be removed. Each *volume-id* must identify a volume that is in the storage group.

The REMOVE VOLUMES clause is applied to the current list of volumes before the ADD VOLUMES clause is applied. Removing a volume from a storage group does not affect existing data, but a volume that has been removed is not used again when the storage group is used to allocate storage for table spaces or index spaces.

Asterisks are recognized only by Storage Management Subsystem (SMS). For information about using asterisks, see the above description of the ADD VOLUMES clause.

Notes

Work file databases: If the storage group altered contains data sets in a work file database, the database must be stopped and restarted for the effects of the ALTER to be recognized. To stop and restart a database, issue the following commands:

```
-STOP DATABASE(database-name)
-START DATABASE(database-name)
```

Device types: When the storage group is used at run time, an error can occur if the volumes in the storage group are of different device types, or if a volume is not available to MVS for dynamic allocation of data sets.

When a storage group is used to extend a data set, all volumes in the storage group must be of the same device type as the volumes used when the data set was defined. Otherwise, an extend failure occurs if an attempt is made to extend the data set.

Number of volumes: There is no specific limit on the number of volumes that can be defined for a storage group. However, the maximum number of volumes that can be managed for a storage group is 133. Thus, there is no point in creating a storage group with more than 133 volumes.

ALTER STOGROUP

MVS imposes a limit on the number of volumes that can be allocated per data set: 59 at this writing. For the latest information on that restriction, see *DFSMS/MVS: Access Method Services for the Integrated Catalog*.

Verifying volume IDs: When processing the ADD VOLUMES or REMOVE VOLUMES clause, DB2 does not check the existence of the volumes or determine the types of devices that they identify. Later, when the storage group is used to allocate or deallocate data sets, the list of volumes is passed in the specified order to Data Facilities (DFSMSdfp), which does the actual work. See Part 2 (Volume 1) of *DB2 Administration Guide* for more information about creating DB2 storage groups.

SMS data set management: You can have Storage Management Subsystem (SMS) manage the storage needed for the objects that the storage group supports. To do so, specify ADD VOLUMES('*) and REMOVE VOLUMES(*current-vols*) in the ALTER statement, where *current-vols* is the list of the volumes currently assigned to the storage group. SMS manages every data set created later for the storage group. SMS does not manage data sets created before the execution of the statement.

You can also specify ADD VOLUMES(*volume-id*) and REMOVE VOLUMES('*) to make the opposite change.

See Part 2 (Volume 1) of *DB2 Administration Guide* for considerations for using SMS to manage data sets.

Examples

Example 1: Alter storage group DSN8G710. Add volumes DSNV04 and DSNV05.

```
ALTER STOGROUP DSN8G710  
    ADD VOLUMES (DSNV04,DSNV05);
```

Example 2: Alter storage group DSN8G710. Remove volumes DSNV04 and DSNV05.

```
ALTER STOGROUP DSN8G710  
    REMOVE VOLUMES (DSNV04,DSNV05);
```

ALTER TABLE

The ALTER TABLE statement changes the description of a table at the current server.

Invocation

This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is implicitly or explicitly specified.

Authorization

The privilege set that is defined below must include at least one of the following:

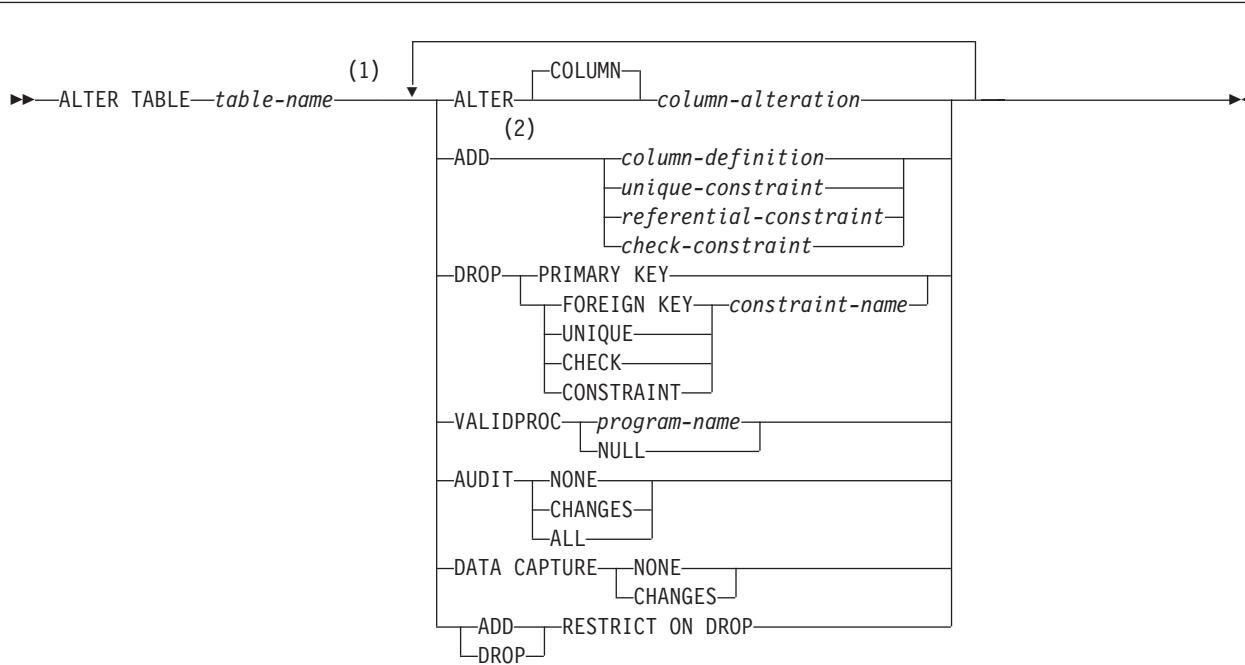
- The ALTER privilege on the table
- Ownership of the table
- DBADM authority for the database
- SYSADM or SYSCTRL authority

Additional privileges might be required if FOREIGN KEY, DROP PRIMARY KEY, DROP FOREIGN KEY, or DROP CONSTRAINT is specific or the data type of a column that is added to the table is a distinct type. See the description of the appropriate clauses for the details about these privileges.

Privilege set: If the statement is embedded in an application program, the privilege set is the privileges that are held by the authorization ID of the owner of the plan or package. If the statement is dynamically prepared, the privilege set is the union of the privilege sets that are held by each authorization ID of the process.

ALTER TABLE

Syntax

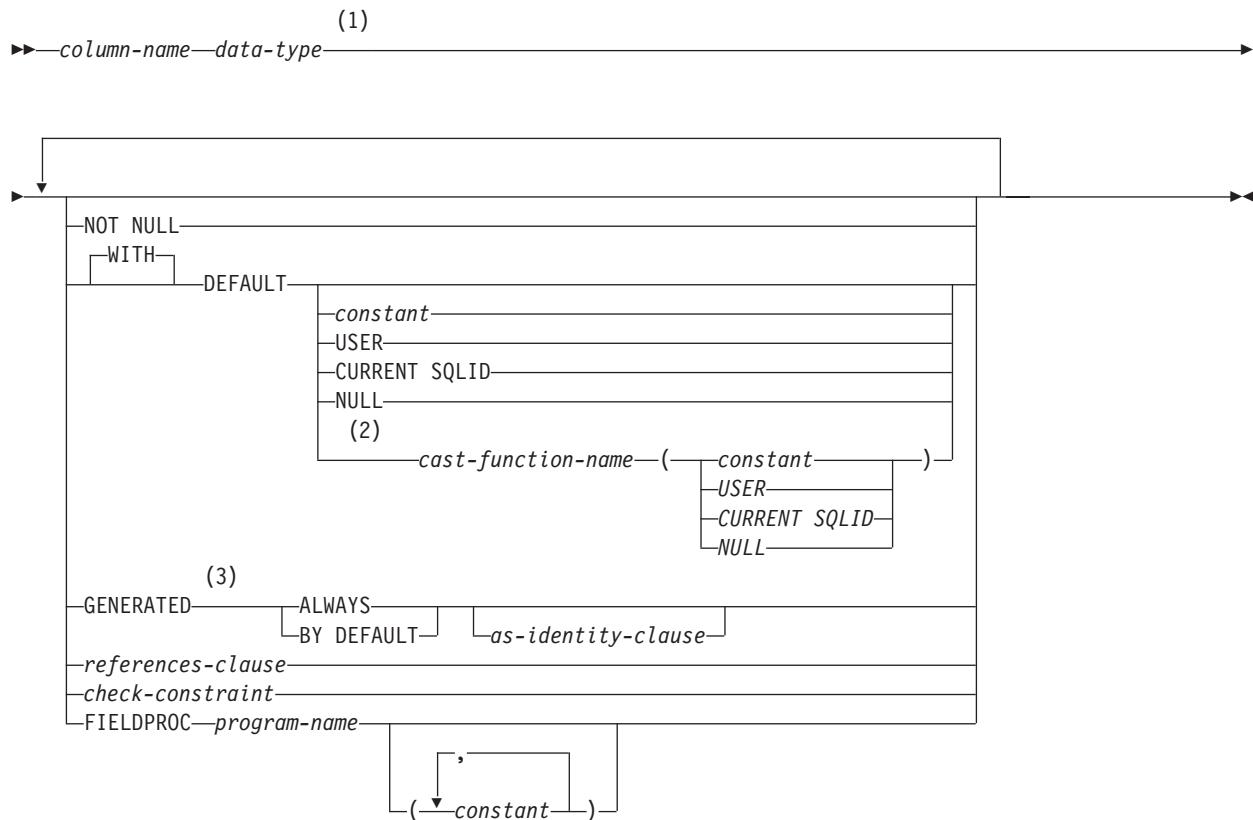


Notes:

- 1 The same clause must not be specified more than once, except for the `ALTER COLUMN` clause, which can be specified more than once. Do not specify `DROP CONSTRAINT` if `DROP FOREIGN KEY` or `DROP CHECK` is specified.
- 2 The `ADD` keyword is optional for the `unique-constraint` and `referential-constraint` clauses if it is the first clause specified in the statement. Otherwise, it is required.

column-alteration:



column-definition:**Notes:**

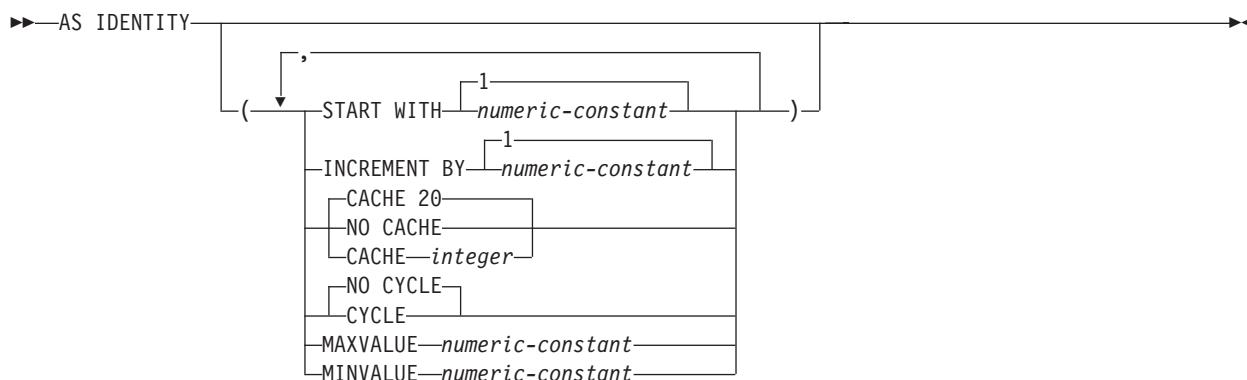
- 1 The same clause must not be specified more than once.
- 2 The cast-function-name form of the DEFAULT value can only be used with a column that is defined as a distinct type.
- 3 GENERATED can be specified only if the column has a ROWID data type (or a distinct type that is based on a ROWID data type), or the column is to be an identity column.

data-type:**Notes:**

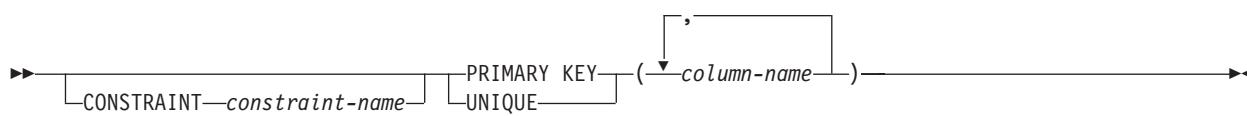
- 1 For the syntax, see “built-in-data-type” on page 656.

ALTER TABLE

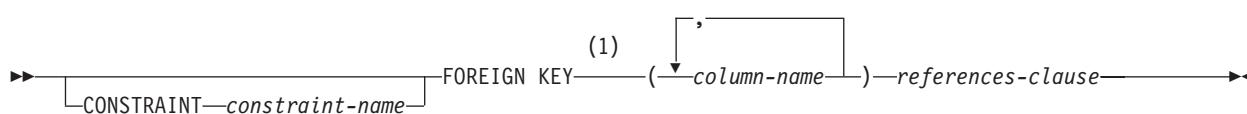
as-identity-clause:



unique-constraint:



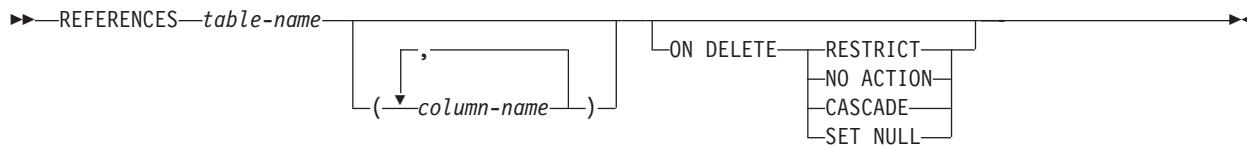
referential-constraint:



Notes:

- 1 For compatibility with prior releases, when the CONSTRAINT clause (shown above) is not specified, a *constraint-name* may be specified following FOREIGN KEY.

references-clause:



check-constraint:

```

>>> ┌─────────────────────────────────────────────────────────────────┐
>>> | CONSTRAINT—constraint-name ──────────────────────────────────|
>>> └─────────────────────────────────────────────────────────┘

```

Description*table-name*

Identifies the table to be altered. The name must identify a table that exists at the current server. The name must not identify an auxiliary table, declared temporary table, or view. If the name identifies a catalog table, DATA CAPTURE CHANGES is the only clause that can be specified.

column-alteration**ALTER COLUMN column-alteration**

Alters the definition of a column. Only the length attribute of an existing column with a VARCHAR data type can be changed. A column cannot be altered if it is used in a referential constraint or a view or has a field procedure routine. It also cannot be altered if it belongs to a table that has edit or validation routine, is defined with DATA CAPTURE CHANGES, or is a created temporary table.

column-name

Identifies the column to be altered. The name must not be qualified and must identify an existing column in the table that has a VARCHAR data type. The name must not identify a column that is being added in the same ALTER TABLE statement.

SET DATA TYPE VARCHAR (*integer*)

Specifies the new length for the column. The value of *integer* must be equal to or greater than the current maximum length of the column.

The new length must not make the total byte count of all columns in a row exceed the maximum row size. (For information on byte counts of columns, see “Byte counts” on page 676.) If the column is used in an index, the new length must not make the sum of the length attributes of the specified index columns greater than 255. If the column is used in a partitioning index, the limit is subject to the following restriction that is imposed by the combination of the number of table space partitions and the limit key length:

$$(\text{number of partitions}) * (106 + \text{limit key size in bytes}) < 65394.$$

```

#
#
#
#
```

ALTER TABLE

The length of more than one column can be changed in a single ALTER TABLE statement if each ALTER COLUMN clause identifies a unique column of the table. The ALTER COLUMN clause and ADD CHECK CONSTRAINT clause can identify the same column.

End of column-alteration

column-definition

ADD column-definition

Adds a column to the table. Except for a ROWID column and an identity column, all values of the column in existing rows are set to its default value. If the table has n columns, the ordinality of the new column is $n+1$. The value of n cannot be greater than 749. For a dependent table, n cannot be greater than 748.

The column cannot be added if the increase in the total byte count of the columns exceeds the maximum row size. The maximum row size for the table is eight less than the maximum record size as described in “Maximum record size” on page 676. A column also cannot be added to a table that has an edit procedure.

You can add a LOB column only if the table has a ROWID column. A table can have only one ROWID column. You cannot add a LOB or ROWID column to a created temporary table. For details about adding a LOB column, such as the other objects that might be implicitly created or need to be explicitly created, see “Creating a table with LOB columns” on page 675. For more information about adding a ROWID column, see “Adding a ROWID column” on page 464.

You cannot add an identity column to a table that has an identity column, or to a created temporary table. For more information about adding an identity column, see “Adding an identity column” on page 464.

column-name

Is the name of the column you want to add to the table. Do not use the name of an existing column of the table. Do not qualify *column-name*.

built-in-data-type

Specifies the data type of the column is one of the built-in data types. See “built-in-data-type” on page 658 for detail.

distinct-type-name

Specifies the distinct type (user-defined data type) of the column. The length and scale of the column are respectively the length and scale of the source type of the distinct type. The privilege set must implicitly or explicitly include the USAGE privilege on the distinct type.

The encoding scheme of the distinct type must be the same as the encoding scheme of the table.

If the column is to be used in the definition of the foreign key of a referential constraint, the data type of the corresponding column of the parent key must have the same distinct type.

NOT NULL

Prevents the column from containing null values. If NOT NULL is specified, the DEFAULT clause must be used to specify a nonnull default value for the column unless the column has a row ID data type or is an identity column. For a ROWID column, NOT NULL must be specified, and DEFAULT must

not be specified. For an identity column, although NOT NULL can be specified, DEFAULT must not be specified.

DEFAULT

The default value assigned to the column in the absence of a value specified on INSERT or LOAD. Do not specify DEFAULT for a ROWID column or an identity column (a column that is defined AS IDENTITY); DB2 generates default values. If a value is not specified after the DEFAULT keyword, the default value depends on the data type of the column as indicated in the following table:

Data Type	Default Value
Numeric	0
Fixed-length string	Blanks
Varying-length string	A string of length 0
Date	For existing rows, a date corresponding to 1 January 0001. For added rows, CURRENT DATE.
Time	For existing rows, a time corresponding to 0 hours, 0 minutes, and 0 seconds. For added rows, CURRENT TIME.
Timestamp	For existing rows, a date corresponding to 1 January 0001, and a time corresponding to 0 hours, 0 minutes, 0 seconds, and 0 microseconds. For added rows, CURRENT TIMESTAMP.

A default value other than the one that is listed above can be specified in one of the following forms, except for a LOB column. The only form that can be specified for a LOB column is DEFAULT NULL. Unlike other varying-length strings, a LOB column can only have the default value of a zero-length string as listed above or null.

constant

Specifies a constant as the default value for the column. The value of the constant must conform to the rules for assigning that value to the column. If the table has an encoding scheme of Unicode and the column is a graphic data type (or a distinct type that is sourced on a graphic data type), a character string constant can also be specified.

USER

Specifies the value of the USER special register at the time of INSERT or LOAD as the default for the column. If USER is specified, the data type of the column must be a character string with a length greater than or equal to the length attribute of the USER special register. For existing rows, the value is that of the USER special register at the time the ALTER TABLE statement is processed.

CURRENT SQLID

Specifies the value of the SQL authorization ID of the process at the time of INSERT or LOAD as the default for the column. If CURRENT SQLID is specified, the data type of the column must be a character string with a length greater than or equal to the length attribute of the

#

#

ALTER TABLE

CURRENT SQLID special register. For existing rows, the value is the SQL authorization ID of the process at the time the ALTER TABLE statement is processed.

NULL

Specifies null as the default value for the column.

cast-function-name

The name of the cast function that matches the name of the distinct type for the column. A cast function can be specified only if the data type of the column is a distinct type.

The schema name of the cast function, whether it is explicitly specified or implicitly resolved through function resolution, must be the same as the explicitly or implicitly specified schema name of the distinct type.

In a given column definition:

- DEFAULT and FIELDPROC cannot both be specified.
- NOT NULL and DEFAULT NULL cannot both be specified.
- DEFAULT cannot be specified for a ROWID column or an identity column.
- Omission of NOT NULL and DEFAULT for a column other than an identity column is an implicit specification of DEFAULT NULL. For an identity column, it is an implicit specification of NOT NULL, and DB2 generates default values.

GENERATED

Indicates that DB2 generates values for the column. You must specify GENERATED if the column is to be considered an identity column (defined with the AS IDENTITY clause) or the data type of the column is a ROWID (or a distinct type that is based on a ROWID).

ALWAYS

Indicates that DB2 will generate a value for the column when a row is inserted into the table. ALWAYS is the recommended value unless you are using data propagation.

BY DEFAULT

Indicates that DB2 will generate a value for the column when a row is inserted unless a value is specified into the table only if a value is not specified. Otherwise, DB2 uses the value that you specify.

For a ROWID column, DB2 uses a specified value only if it is a valid row ID value that was previously generated by DB2 and the column has a unique, single-column index. Until this index is created on the ROWID column, the SQL INSERT statement and the LOAD utility cannot be used to add rows to the table. If the value of special register CURRENT RULES is 'STD' when the ALTER TABLE statement is processed, DB2 implicitly creates the index on the ROWID column. The name of this index is 'I' followed by the first ten characters of the column name followed by seven randomly generated characters. If the column name is less than ten characters, DB2 adds underscore characters to the end of the name until it has ten characters. The implicitly created index has the COPY NO attribute.

For an identity column, DB2 inserts a specified value but does not verify that it is a unique value for the column unless the identity column has a

unique, single-column index; without a unique index, DB2 can guarantee unique values only among the set of system-generated values.

BY DEFAULT is the recommended value only when you are using data propagation.

AS IDENTITY

Specifies that the column is an identity column for the table. A table can have only one identity column. AS IDENTITY can be specified only if the data type for the column is an exact numeric type with a scale of zero (SMALLINT, INTEGER, DECIMAL with a scale of zero, or a distinct type based on one of these types).

An identity column is implicitly NOT NULL.

START WITH *numeric-constant*

Specifies the first value that is generated for the identity column. The value can be any positive or negative value that could be assigned to the column without non-zero digits existing to the right of the decimal point.

If a value is not explicitly specified when the identity column is defined, the default is the MINVALUE for an ascending sequence and the MAXVALUE for a descending sequence. This value is not necessarily the value that a sequence would cycle to after reaching the maximum or minimum value of the sequence. The START WITH clause can be used to start a sequence outside the range that is used for cycles. The range used for cycles is defined by MINVALUE and MAXVALUE.

INCREMENT BY *numeric-constant*

Specifies the interval between consecutive values of the identity column. The value can be any positive or negative value that is not 0, does not exceed the value of a large integer constant, and could be assigned to the column without any non-zero digits existing to the right of the decimal point. The default is 1.

If the value is positive, the sequence of values for the identity column ascends. If the value is negative, the sequence of values descends.

CACHE or NO CACHE

Specifies whether to keep some preallocated values in memory. Preallocating and storing values in the cache improves the performance of inserting rows into a table.

CACHE *integer*

Specifies the number of values of the identity column sequence that DB2 preallocates and keeps in memory. The minimum value that can be specified is 2, and the maximum is the largest value that can be represented as an integer. The default is 20.

During a system failure, all cached identity column values that are yet to be assigned are lost, and thus, will never be used. Therefore, the value specified for CACHE also represents the maximum number of values for the identity column that could be lost during a system failure.

In a data sharing environment, each member gets its own range of consecutive values to assign. For example, if CACHE

ALTER TABLE

20 is specified, DB2A might get values 1-20 for a particular sequence, and DB2B might get values 21-40. Therefore, if transactions from different members generate values for the same identity column, the values that are assigned might not be in the order in which they are requested.

The minimum value is 2. The maximum is the largest value that can be represented as an integer. The default is CACHE 20.

NO CACHE

Specifies that values for the identity column are not preallocated.

In a data sharing environment, use NO CACHE if you need to guarantee that the identity values are generated in the order in which they are requested.

CYCLE or NO CYCLE

Specifies whether this identity column should continue to generate values after reaching either the maximum or minimum value of the sequence.

CYCLE

Specifies that values continue to be generated for this column after the maximum or minimum value has been reached. If this option is used, after an ascending sequence reaches the maximum value of the sequence, it generates its minimum value. After a descending sequence reaches its minimum value of the sequence, it generates its maximum value. The maximum and minimum values for the column determine the range that is used for cycling.

When CYCLE is in effect, duplicate values can be generated by DB2 for an identity column. However, if a unique index exists on the identity column, and a non-unique value is generated for it, an error occurs.

NO CYCLE

Specifies that values will not be generated for the identity column once the maximum or minimum value for the sequence has been reached. This is the default.

MAXVALUE *numeric-constant*

Specifies the numeric constant that is the maximum value that is generated for this identity column. This value can be any positive or negative value that could be assigned to this column, but the value must be greater than the minimum value.

If a value is not explicitly specified when the identity column is defined, this is the maximum value of the datatype (and precision, if DECIMAL) for an ascending sequence; or the **START WITH** value, or -1 if **START WITH** was not specified, for a descending sequence.

MINVALUE *numeric-constant*

Specifies the numeric constant that is the minimum value that is generated for this identity column. This value can be any positive or negative value that could be assigned to this column, but the value must be less than the maximum value.

If a value is not explicitly specified when the identity column is defined, this is the **START WITH** value, or 1 if **START WITH** was

not specified, for an ascending sequence; or the minimum value of the data type (and precision, if DECIMAL) for a descending sequence.

references-clause

The *references-clause* of a *column-definition* provides a shorthand method of defining a foreign key composed of a single column. Thus, if *references-clause* is specified in the definition of column C, the effect is the same as if that *references-clause* were specified as part of a FOREIGN KEY clause in which C is the only identified column.

Do not specify *references-clause* in the definition of a LOB or ROWID column because a LOB or ROWID column cannot be a foreign key.

check-constraint

The *check-constraint* of a *column-definition* has the same effect as specifying a table check constraint in a separate ADD *check-constraint* clause. For conformance with the SQL standard, a table check constraint specified in the definition of column C should not reference any columns other than C.

Do not specify a table check constraint in the definition of a LOB or ROWID column.

FIELDPROC *program-name*

Designates *program-name* as the field procedure exit routine for the column. Writing a field procedure exit routine is described in Appendix B (Volume 2) of *DB2 Administration Guide*. Field procedures can only be specified for short string columns that do not have a nonnull default value.

The field procedure encodes and decodes column values. Before a value is inserted in the column, it is passed to the field procedure for encoding. Before a value from the column is used by a program, it is passed to the field procedure for decoding. A field procedure could be used, for example, to alter the sorting sequence of values entered in the column.

The field procedure is also invoked during the processing of the ALTER TABLE statement. When so invoked, the procedure provides DB2 with the column's *field description*. The field description defines the data characteristics of the encoded values. By contrast, the information you supply for the column in the ALTER TABLE statement defines the data characteristics of the decoded values.

constant

Is a parameter that is passed to the field procedure when it is invoked. A parameter list is optional. The *n*th parameter specified in the FIELDPROC clause on ALTER TABLE corresponds to the *n*th parameter of the specified field procedure. The maximum length of the parameter list is 255 bytes, including commas but excluding insignificant blanks and the delimiting parentheses.

If you omit FIELDPROC, the column has no field procedure.

End of column-definition

unique-constraint

CONSTRAINT *constraint-name*

Names the primary key or unique key constraint. If a constraint name is not

ALTER TABLE

specified, a unique constraint name is generated. If a name is specified, it must be different from the names of any referential, check, primary key, or unique key constraints previously specified on the table.

PRIMARY KEY(*column-name*,...)

Defines a primary key composed of the identified columns. Each column name must be an unqualified name that identifies a column of the table except a LOB or ROWID column, and the same column must not be identified more than once. The number of identified columns must not exceed 64 and the sum of their length attributes must not exceed 255. The table must not have a primary key and the identified columns must be defined as NOT NULL.

The set of columns in the primary key cannot be the same as the set of columns of another unique key.

The table must have a unique index with a key that is identical to the primary key. The keys are identical only if they have the same number of columns and the *n*th column name of one is the same as the *n*th column name of the other.

The identified columns are defined as the primary key of the table. The description of the index is changed to indicate that it is a primary index. If the table has more than one unique index with a key that is identical to the primary key, the selection of the primary index is arbitrary.

UNIQUE(*column-name*,...)

Defines a unique key composed of the identified columns with the specified *constraint-name*. If a *constraint-name* is not specified, a name is generated. Each column name must be an unqualified name that identifies a column of the table except a LOB or ROWID column, and the same column must not be identified more than once. Each identified column must be defined as NOT NULL. The number of identified columns must not exceed 64 and the sum of their length attributes must not exceed 255.

The set of columns in the unique key cannot be the same as the set of columns of the primary key or another unique key. A unique key is a duplicate if it is the same as the primary key or a previously defined unique key. The specification of a duplicate unique key is ignored with a warning.

The table must have a unique index with a key that is identical to the unique key. The keys are identical only if they have the same number of columns and the *n*th column name of one is the same as the *n*th column name of the other.

The identified columns are defined as a unique key of the table. The description of the index is changed to indicate that it is enforcing a unique key constraint. If the table has more than one unique index with a key that is identical to the unique key, the selection of the enforcing index is arbitrary.

End of unique-constraint

referential-constraint

CONSTRAINT *constraint-name*

Names the referential constraint. If a constraint name is not specified, a unique constraint name is generated. If a name is specified, it must be different from the names of any referential, check, primary key, or unique key constraints previously specified on the table.

FOREIGN KEY (*column-name*,...) references-clause

Specifies a referential constraint with the specified *constraint-name*.

Let T1 denote the object table of the ALTER TABLE statement.

The foreign key of the referential constraint is composed of the identified columns. Each *column-name* must be an unqualified name that identifies a column of T1 except a LOB or ROWID column, and the same column must not be identified more than once. The number of identified columns must not exceed 64 and the sum of their length attributes must not exceed 255 minus the number of columns that allow null values. The referential constraint is a duplicate if the FOREIGN KEY and the parent table are the same as the FOREIGN KEY and parent table of an existing referential constraint on T1. The specification of a duplicate referential constraint is ignored with a warning.

End of referential-constraint

references-clause

REFERENCES *table-name (column-name,...)*

The table name specified after REFERENCES must identify a table that exists at the current server, but it must not identify a catalog table. Let T2 denote the identified parent table and let T1 denote the table being altered (T1 and T2 can be the same table).

T2 must have a unique index and the privilege set on T2 must include the ALTER or REFERENCES privilege on the parent table, or the REFERENCES privilege on the columns of the nominated parent key.

The parent key of the referential constraint is composed of the identified columns. Each *column-name* must be an unqualified name that identifies a column of T2. The identified column cannot be a LOB or a ROWID column. The same column must not be identified more than once.

The list of column names in the parent key must be identical to the list of column names in a primary key or unique key in the parent table T2. The column names must be specified in the *same order* as in the primary key or unique key.

If a list of column names is not specified, then T2 must have a primary key. Omission of a list of column names is an implicit specification of the columns of the primary key for T2.

The specified foreign key must have the same number of columns as the parent key of T2 and, except for their names, default values, null attributes and check constraints, the description of the *n*th column of the foreign key must be identical to the description of the *n*th column of the nominated parent key. If the foreign key includes a column defined as a distinct type, the corresponding column of the nominated parent key must be the same distinct type. If a column of the foreign key has a field procedure, the corresponding column of the nominated parent key must have the same field procedure and an identical field description. A *field description* is a description of the encoded value as it is stored in the database for a column that has been defined to have an associated field procedure.

The table space that contains T1 must be available to DB2. If T1 is populated, its table space is placed in a check pending status.²⁷ A table in a segmented table space is populated if the table is not empty. A table in an nonsegmented table space is considered populated if the table space has ever contained any records.

27. The check pending status prevents further updating or reading by other SQL applications. It does not affect the application process that issues ALTER TABLE. However, we do not recommend that a process create or alter a permanent table and then access it.

ALTER TABLE

The referential constraint specified by the FOREIGN KEY clause defines a relationship in which T2 is the parent and T1 is the dependent. A description of the referential constraint is recorded in the catalog.

ON DELETE

The delete rule of the relationship is determined by the ON DELETE clause. For more on the concepts used here, see “Referential constraints” on page 7.

If T1 and T2 are the same table, CASCADE or NO ACTION must be specified. SET NULL must not be specified unless some column of the foreign key allows null values. Also, SET NULL must not be specified if any nullable column of the foreign key is a column of the key of a partitioning index. The default value for the rule depends on the value of the CURRENT RULES special register when the CREATE TABLE statement is processed. If the value of the register is 'DB2', the delete rule defaults to RESTRICT; if the value is 'SQL', the delete rule defaults to NO ACTION.

The delete rule applies when a row of T2 is the object of a DELETE or propagated delete operation and that row has dependents in T1. Let p denote such a row of T2.

- If RESTRICT or NO ACTION is specified, an error occurs and no rows are deleted.
- If CASCADE is specified, the delete operation is propagated to the dependents of p in T1.
- If SET NULL is specified, each nullable column of the foreign key of each dependent of p in T1 is set to null.

A cycle involving two or more tables must not cause a table to be delete-connected to itself. Thus, if the relationship would form a cycle:

- The referential constraint cannot be defined if each of the existing relationships that would be part of the cycle have a delete rule of CASCADE.
- CASCADE must not be specified if T2 is delete-connected to T1.

If T1 is delete-connected to T2 through multiple paths, those relationships in which T1 is a dependent and which form all or part of those paths must have the same delete rule and it must not be SET NULL. For example, assume that T1 is a dependent of T3 in a relationship with a delete rule of r and that one of the following is true:

- T2 and T3 are the same table.
- T2 is a descendent of T3 and the deletion of rows from T3 cascades to T2.
- T2 and T3 are both descendants of the same table and the deletion of rows from that table cascades to both T2 and T3.

In this case, the referential constraint cannot be defined when r is SET NULL. When r is other than SET NULL, the referential constraint can be defined, but the delete rule that is implicitly or explicitly specified in the FOREIGN KEY clause must be the same as r .

End of references-clause

check-constraint

CONSTRAINT *constraint-name*

Names the table check constraint. If constraint-name is not specified, a unique constraint name is derived from the name of the first column in the

check-condition specified in the definition of the table check constraint. If a name is specified, it must be different from the names of any referential, check, primary key, or unique key constraints previously specified on the table.

CHECK (*check-condition*)

Defines a table check constraint. A check-condition can evaluate to unknown if a column that is an operand of the predicate is null. A check-condition that evaluates to unknown does not violate the check constraint. A *check-condition* is a search condition, with the following restrictions:

- It can refer only to the columns of table *table-name*; however, the columns cannot be LOB or ROWID columns.
- It can be up to 3800 bytes long, not including redundant blanks.
- It must not contain any of the following:
 - Subselects
 - Built-in or user-defined functions
 - Cast functions other than those created when the distinct type was created
 - Host variables
 - Parameter markers
 - Special registers
 - Columns that include a field procedure
 - CASE expressions
 - Quantified predicates
 - EXISTS predicates
- If a check-condition refers to a long string column, the reference must occur within a LIKE predicate.
- The AND and OR logical operators can be used between predicates. The NOT logical operator cannot be used.
- The first operand of every predicate must be the column name of a column in the table.
- The second operand in the check-condition must be either a constant or a column name of a column in the table.
 - If the second operand of a predicate is a constant, and if the constant is:
 - A floating-point number, then the column data type must be floating point.
 - A decimal number, then the column data type must be either floating point or decimal.
 - An integer number, then the column data type must not be a small integer.
 - A small integer number, then the column data type must be small integer.
 - A decimal constant, then its precision must not be larger than the precision of the column.
 - If the second operand of a predicate is a column, then both columns of the predicate must have:
 - The same data type
 - Identical descriptions with the exception that the specification of the NOT NULL and DEFAULT clauses for the columns can be different, and that string columns with the same data type can have different length attributes

Effects of defining a check constraint on a populated table: When a check constraint is defined on a populated table and the value of the special register CURRENT RULES is 'DB2', the check constraint is not immediately enforced on the table. The check constraint is added to the description of the table, and the

ALTER TABLE

table space that contains the table is placed in a check pending status. For a description of the check pending status and the implications for utility operations, see Part 2 of *DB2 Utility Guide and Reference*.

When a check constraint is defined on a populated table and the value of the special register CURRENT RULES is 'STD', the check constraint is checked against all rows of the table. If no violations occur, the check constraint is added to the table. If any rows violate the new check constraint, an error occurs and the description of the table is unchanged.

End of check-constraint

DROP PRIMARY KEY

Drops the definition of the primary key and all referential constraints in which the primary key is a parent key. The table must have a primary key and the privilege set must include the ALTER or REFERENCES privilege on every dependent table of the table.

The description of the primary index is changed to indicate that it is not a primary index.

DROP FOREIGN KEY *constraint-name*

Drops the referential constraint *constraint-name*. The constraint-name must identify a referential constraint in which the table is the dependent table, and the privilege set must include the ALTER or REFERENCES privilege on the parent table of that relationship, or the REFERENCES privilege on the columns of the parent table of that relationship.

DROP UNIQUE *constraint-name*

Drops the definition of the unique key constraint and all referential constraints in which the unique key is a parent key. The table must have a unique key. The privilege set must include the ALTER or REFERENCES privilege on every dependent table of the table. The description of the enforcing index is changed to indicate that it is not enforcing a unique key constraint.

DROP CHECK *constraint-name*

Drops the check constraint *constraint-name*. The constraint-name must identify an existing check constraint defined on the table.

DROP CONSTRAINT *constraint-name*

Drops the constraint *constraint-name*. The constraint-name must identify an existing primary key, unique key, check, or referential constraint defined on the table.

DROP CONSTRAINT must not be used on the same ALTER TABLE statement as DROP PRIMARY KEY, DROP UNIQUE KEY, DROP FOREIGN KEY or DROP CHECK.

VALIDPROC

Names a validation procedure for the table or inhibits the execution of any existing validation procedure.

program-name

Designates *program-name* as the new validation exit routine for the table. Validation exit routines are described in Appendix B (Volume 2) of *DB2 Administration Guide*.

The validation procedure can inhibit a load, insert, update, or delete operation on any row of the table. Before the operation takes place, the row is passed to the procedure. The values represented by any LOB columns in the table are not passed. After examining the row, the procedure returns a

value that indicates whether the operation should proceed. A typical use is to impose restrictions on the values that can appear in various columns.

A table can have only one validation procedure at a time. When you name a new procedure, any existing procedure is no longer used. The new procedure is not used to validate existing table rows. It is used only to validate rows that are loaded, inserted, updated, or deleted after execution of the ALTER TABLE statement.

NULL

Discontinues the use of any validation routine for the table.

AUDIT

Alters the auditing attribute of the table. For information about audit trace classes, see Part 3 (Volume 1) of *DB2 Administration Guide*.

NONE

Specifies that no auditing is to be done when the table is accessed.

CHANGES

Specifies that auditing is to be done when the table is accessed during the first insert, update, or delete operation performed by each unit of recovery. However, the auditing is done only if the appropriate audit trace class is active.

ALL

Specifies that auditing is to be done when the table is accessed during the first operation of any kind performed by each unit of work of a utility or application process. However, the auditing is done only if the appropriate audit trace class is active and the access is not performed with COPY, RECOVER, REPAIR, or any stand-alone utility.

If the AUDIT attribute is changed to CHANGES or ALL, subsequent ALTER TABLE statements will be audited if the appropriate audit trace class is active.

DATA CAPTURE

Specifies whether the logging of SQL INSERT, UPDATE, and DELETE operations on the table is augmented by additional information. For guidance on intended uses of the expanded log records, see:

- The description of data propagation to IMS in *DataPropagator NonRelational MVS/ESA Administration Guide*
- The instructions for using Remote Recovery Data Facility (RRDF) in *Remote Recovery Data Facility Program Description and Operations*
- The instructions for reading log records in Appendix C (Volume 2) of *DB2 Administration Guide*

NONE

Do not record additional information to the log.

CHANGES

Write additional data about SQL updates to the log. Information about the values that are represented by any LOB columns is not available.

For details about the recording of additional data for logged updates to catalog tables, see “Notes” on page 464.

ADD RESTRICT ON DROP

Restricts dropping the table and the database and table space that contain the table.

ALTER TABLE

DROP RESTRICT ON DROP

Removes the restriction on dropping the table and the database and table space that contain the table.

Notes

Restrictions for adding columns: When using ALTER TABLE, you cannot add:

- A column to a table that has an edit procedure
- A LOB column unless the table has a ROWID column
- A ROWID column to a table that already has a ROWID column
- An identity column to a table that already has an identity column
- A LOB, ROWID, or identity column to a created temporary table
- A GRAPHIC, VARGRAPHIC, DBCLOB, or CHAR FOR MIXED DATA column, when the setting for installation option MIXED DATA is NO

Because a distinct type is subject to the same restrictions as its source type, all the syntactic rules that apply to LOB and ROWID columns apply to distinct type columns that are sourced on LOBs and row IDs. For example, if a table has ROWID column, you cannot add a column with a distinct type that is sourced on a row ID.

Adding a column to table T only changes the description of T. If the catalog description of T is used to create a table T' and a facility such as DSN1COPY is used to effectively copy T into T', queries that refer to the added column in T' will fail because the data does not match its description. To avoid this problem, run the REORG utility against the table space of T before making the copy.

Adding a ROWID column: When you add a ROWID column to an existing table, DB2 ensures that the same, unique row ID value is returned for a row whenever it is accessed. Reorganizing a table space has no effect on the values in a ROWID column.

Adding an identity column: When you add an identity column to a table that is not empty, DB2 places the table space that contains the table in the REORG pending state. When the REORG utility is subsequently run, DB2 generates the values for the identity column in all existing rows and then removes the REORG pending status. These values are guaranteed to be unique, and their order is system-determined.

Altering the length of a column: Only the length of VARCHAR columns can be changed. When changing the length of a column, be aware of the following information about indexes, limit keys, check constraints, and invalidation.

- *Restrictions.* The length of a VARCHAR column cannot be changed if any of the following conditions are true:
 - The column is referenced in a referential constraint or view.
 - The column has a field procedure routine.
 - The table has an edit or validation routine.
 - The table is defined with DATA CAPTURE CHANGES.
 - The table is a created temporary table.
- *Indexes.* After the ALTER TABLE statement is executed, each index on the table with a key that includes a column whose length was increased remains available. However, SQL operations against such an index are not allowed until the changes from the ALTER TABLE statement are committed.

The maximum number of *distinct* alters that increase the index key length is sixteen or less. If the maximum number of alters is exceeded, SQLCODE -148 is returned, and the index must be reorganized or rebuilt. An alter is considered

distinct when it occurs in a different unit of work than the previous alter. For example, changing an index column length, committing database changes, and changing the column length of that index column or another index column counts as two distinct alters. Whereas, changing an index column length twice before committing any changes counts as one distinct alter; the second changes replace the first because it was in the same commit scope. Changing the length of two different index columns before committing the changes also counts as one distinct alter.

- *Length of partitioned index keys.* When a table is altered and the length of a column in the index is changed, DB2 changes the length of the limit key (the highest key value) for a partition, too. The length of the limit key is increased by the same amount that the length of column is increased.
- *Check constraints.* If a table check constraint refers to the column being altered, the length of the column is also changed in the check constraint.
- *Invalidation.* When a table is altered to change the length of a VARCHAR column, all plans, packages, and dynamic cached statements that reference the table are invalidated.

Invalidation of plans and packages: When a table is altered, all the plans and packages that refer to the table are invalidated if any one of the following conditions is true:

- The AUDIT attribute of the table is changed.
- A DATE, TIME, or TIMESTAMP column is added and its default value for added rows is CURRENT DATE, CURRENT TIME, or CURRENT TIMESTAMP, respectively.
- The length attribute of a VARCHAR column is changed.
- The table is a created temporary table.

When a referential constraint is defined with a delete rule of CASCADE or SET NULL, all plans and packages that refer to the parent table of the constraint are invalidated. Furthermore, all plans and packages that refer to tables from which deletes cascade to this parent table are also invalidated.

Views: Adding a column to a table has no effect on existing views.

Order of processing of clauses: When there is more than one clause, they are processed in the following order: VALIDPROC, AUDIT, DATA CAPTURE, DROP clauses, ALTER COLUMN, and ADD clauses.

Running utilities: You cannot execute the ALTER TABLE statement while a utility has control of the table space that contains the table.

Dropping constraints and check pending status: If a table space or partition is in check pending status because it contains a table with rows that violate constraints, dropping the constraints removes the check pending status.

Capturing changes to the DB2 catalog: To have logged changes to a DB2 catalog table augmented with information for data capture, specify ALTER TABLE xxx DATA CAPTURE CHANGES where xxx is the name of a catalog table (SYSIBM.xxx). Data capture of catalog table changes provides the possibility of creating and managing a shadow of the catalog.

ALTER TABLE

Examples

Example 1: Column DEPTNAME in table DSN8710.DEPT was created as a VARCHAR(36). Increase its length to 50 bytes. Also, add the column BLDG to the table DSN8710.DEPT. Describe the new column as a character string column that holds SBCS data.

```
ALTER TABLE DSN8710.DEPT  
    ALTER COLUMN DEPTNAME SET DATA TYPE VARCHAR(50)  
    ADD BLDG CHAR(3) FOR SBCS DATA;
```

Example 2: Assign a validation procedure named DSN8EAEM to the table DSN8710.EMP.

```
ALTER TABLE DSN8710.EMP  
    VALIDPROC DSN8EAEM;
```

Example 3: Disassociate the current validation procedure from the table DSN8710.EMP. After the statement is executed, the table no longer has a validation procedure.

```
ALTER TABLE DSN8710.EMP  
    VALIDPROC NULL;
```

Example 4: Define ADMRDEPT as the foreign key of a self-referencing constraint on DSN8710.DEPT.

```
ALTER TABLE DSN8710.DEPT  
    FOREIGN KEY(ADMRDEPT) REFERENCES DSN8710.DEPT ON DELETE CASCADE;
```

Example 5: Add a check constraint to the table DSN8710.EMP which checks that the minimum salary an employee can have is \$10,000.

```
ALTER TABLE DSN8710.EMP  
    ADD CHECK (SALARY >= 10000);
```

Example 6: Alter the PRODINFO table to define a foreign key that references a non-primary unique key in the product version table (PRODVER_1). The columns of the unique key are VERNAME, RELNO.

```
ALTER TABLE PRODINFO  
    FOREIGN KEY (PRODNAME,PRODVERNO)  
        REFERENCES PRODVER_1 (VERNAME,RELNO) ON DELETE RESTRICT;
```

Example 7: Assume that table DEPT has a unique index defined on column DEPTNAME. Add a unique key constraint named KEY_DEPTNAME consisting of column DEPTNAME to the DEPT table:

```
ALTER TABLE DSN8710.DEPT  
    ADD CONSTRAINT KEY_DEPTNAME UNIQUE( DEPTNAME );
```

ALTER TABLESPACE

The ALTER TABLESPACE statement changes the description of a table space at the current server.

Invocation

This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is implicitly or explicitly specified.

Authorization

The privilege set that is defined below must include at least one of the following:

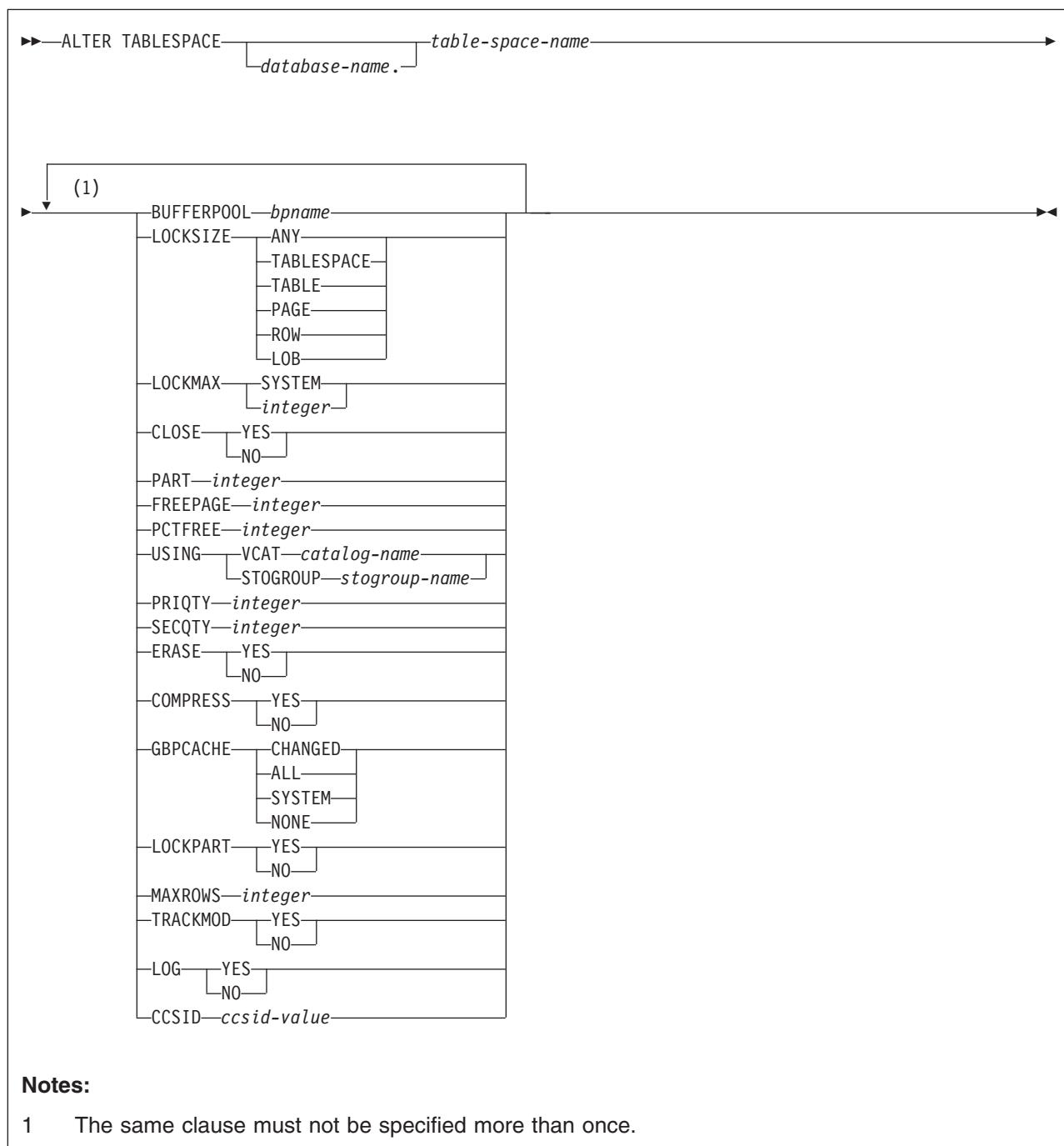
- Ownership of the table space
- DBADM authority for its database
- SYSADM or SYSCTRL authority

If BUFFERPOOL or USING STOGROUP is specified, additional privileges might be required, as explained in the description of those clauses.

Privilege set: If the statement is embedded in an application program, the privilege set is the privileges that are held by the authorization ID of the owner of the plan or package. If the statement is dynamically prepared, the privilege set is the union of the privilege sets that are held by each authorization ID of the process.

ALTER TABLESPACE

Syntax



Notes:

- 1 The same clause must not be specified more than once.

Description

database-name.table-space-name

Identifies the table space to be altered. The name must identify a table space that exists at the current server. Omission of *database-name* is an implicit specification of DSNDB04.

If you identify a table space of a work file database, the database must be in the stopped state. If you identify a partitioned table space, you can use the PART clause as explained below.

BUFFERPOOL *bpname*

Identifies the buffer pool to be used for the table space. The *bpname* must identify an activated buffer pool with the same page size as the table space. See “Naming conventions” on page 34 for more details about *bpname*.

The privilege set must include SYSADM or SYSCTRL authority or the USE privilege for the buffer pool.

The change to the description of the table space takes effect the next time the data sets of the table space are opened. The data sets can be closed and reopened by a STOP DATABASE command to stop the table space followed by a START DATABASE command to start the table space.

In a data sharing environment, if you specify BUFFERPOOL, the table space must be in the stopped state when the ALTER TABLESPACE statement is executed.

LOCKSIZE

Specifies the size of locks used within the table space and, in some cases, also the threshold at which lock escalation occurs. Do not specify **LOCKSIZE** for a table space in a work file database or a **TEMP** database.

ANY

Specifies that DB2 can use any lock size. Currently, DB2 never chooses row locks, but reserves the right to do so.

In most cases, DB2 uses **LOCKSIZE PAGE** **LOCKMAX SYSTEM** for non-LOB table spaces and **LOCKSIZE LOB** **LOCKMAX SYSTEM** for LOB table spaces. However, when the number of locks acquired for the table space exceeds the maximum number of locks allowed for a table space (an installation parameter), the page or LOB locks are released and locking is set at the next higher level. If the table space is segmented, the next higher level is the table. If the table space is nonsegmented, the next higher level is the table space.

TABLESPACE

Specifies table space locks.

TABLE

Specifies table locks. Use TABLE only for a segmented table space.

PAGE

Specifies page locks. Do not use PAGE for a LOB table space.

ROW

Specifies row locks. Do not use ROW for a LOB table space.

LOB

Specifies LOB locks. Use LOB only for a LOB table space.

The change in lock size is immediate and affects locks that are acquired on the
table space for subsequently executed SQL statements with these exceptions:

• Static statements in plans and packages that were bound with
ACQUIRE(ALLOCATE) and the lock size was changed from:
– Table space lock to table, page, or row lock
– Table lock to page or row lock

ALTER TABLESPACE

```
# To make the change in lock size take affect, you need to rebind the plan or
# package.
# • Static statements in plans and packages that were bound with
#   ACQUIRE(USE) and RELEASE(DEALLOC), depending on the timing of the
#   change to the lock size. If the lock size is changed after the statement first
#   accesses the table space, the change is not immediate.
#
# For example, assume that:
#   – The lock size for TABLE1 is table.
#   – Application 1 accesses TABLE1, commits, and then accesses TABLE1
#     again.
#   – Application 2 changes the lock size on TABLE1 when Application 1
#     commits.
#
# When Application 1 accesses TABLE1 the second time, the lock size is still
# table.
```

LOCKMAX

Specifies the maximum number of page, row, or LOB locks an application process can hold simultaneously in the table space. If a program requests more than that number, locks are escalated. The page, row, or LOB locks are released and the intent lock on the table space or segmented table is promoted to S or X mode. If you specify LOCKMAX a for table space in a TEMP database, DB2 ignores the value because these types of locks are not used.

For an application that uses Sysplex query parallelism, a lock count is maintained on each member.

integer

Specifies the number of locks allowed before escalating, in the range 0 to 2 147 483 647.

Zero (0) indicates that the number of locks on the table or table space are not counted and escalation does not occur.

SYSTEM

Indicates that the value of field LOCKS PER TABLE(SPACE) on installation panel DSNTIPJ specifies the maximum number of page, row, or LOB locks a program can hold simultaneously in the table or table space.

If you change LOCKSIZE and omit LOCKMAX, the following results occur:

LOCKSIZE	Resultant LOCKMAX
TABLESPACE or TABLE	0
PAGE, ROW, or LOB	Unchanged
ANY	SYSTEM

If the lock size is TABLESPACE or TABLE, LOCKMAX must be omitted, or its operand must be 0.

CLOSE

When the limit on the number of open data sets is reached, specifies the priority in which data sets are closed.

YES

Eligible for closing before CLOSE NO data sets. This is the default unless the table space is in a TEMP database.

NO

Eligible for closing after all eligible CLOSE YES data sets are closed.

For a table space in a TEMP database, DB2 uses CLOSE NO regardless of the value specified

PART *integer*

Identifies a partition of the table space. For a table space that has *n* partitions, you must specify an integer in the range 1 to *n*. You must not use this clause for a nonpartitioned table space or for a LOB table space. You must use this clause for a partitioned table space if you use any of the following clauses:

FREEPAGE
PCTFREE
USING
PRIQTY
SECQTY
COMPRESS
ERASE
GBPCACHE
TRACKMOD

In this case, the changes specified by these clauses apply only to the identified partition of the table space.

FREEPAGE *integer*

Specifies how often to leave a page of free space when the table space is loaded or reorganized. One free page is left after every *integer* pages; *integer* can range from 0 to 255. FREEPAGE 0 leaves no free pages. Do not specify FREEPAGE for a LOB table space, or a table space in a work file database or a TEMP database.

If the table space is segmented, the number of pages left free must be less than the SEGSIZE value. If the number of pages to be left free is greater than or equal to the SEGSIZE value, then the number of pages is adjusted downward to one less than the SEGSIZE value.

The change to the description of the table space or partition has no effect until it is loaded or reorganized.

PCTFREE *integer*

Specifies what percentage of each page to leave as free space when the table space is loaded or reorganized. The first record on each page is loaded without restriction. When additional records are loaded, at least *integer* percent of free space is left on each page. *integer* can range from 0 to 99. Do not specify PCTFREE for a LOB table space, or a table space in a work file database or a TEMP database.

This change to the description of the table space or partition has no effect until it is loaded or reorganized.

USING

Specifies whether a data set for the table space or partition is managed by the user or managed by DB2. If the table space is partitioned, USING applies to the data set for the partition identified in the PART clause. If the table space is not partitioned, USING applies to every data set that is eligible for the table space. (A nonpartitioned table space can have more than one data set if PRIQTY+118 × SECQTY is at least 2 gigabytes.)

ALTER TABLESPACE

If the USING clause is specified, the table space or partition must be in the stopped state when the ALTER TABLESPACE statement is executed. See “Altering storage attributes” on page 477 to determine how and when changes take effect.

VCAT *catalog-name*

Specifies a user-managed data set with a name that starts with *catalog-name*. You must specify the catalog name in the form of a short identifier. Thus, you must specify an alias if the name of the integrated catalog facility catalog is longer than eight characters. When the new description of the table space is applied, the integrated catalog facility catalog must contain an entry for the data set conforming to the DB2 naming conventions set forth in Part 2 (Volume 1) of *DB2 Administration Guide*.

One or more DB2 subsystems could share integrated catalog facility catalogs with the current server. To avoid the chance of having one of those subsystems attempt to assign the same name to different data sets, select a value for *catalog-name* that is not used by the other DB2 subsystems.

STOGROUP *stogroup-name*

Specifies a DB2-managed data set that resides on a volume of the identified storage group. The stogroup name must identify a storage group that exists at the current server and the privilege set must include SYSADM authority, SYSCTRL authority, or the USE privilege for the storage group. When the new description of the table space is applied, the description of the storage group must include at least one volume serial number, each volume serial number must identify a volume that is accessible to MVS for dynamic allocation of the data set, and all identified volumes must be of the same device type. Furthermore, the integrated catalog facility catalog used for the storage group must not contain an entry for the data set.

If you specify USING STOGROUP and the current data set for the table space or partition is DB2-managed:

- Omission of the PRIQTY clause is an implicit specification of the current PRIQTY value.
- Omission of the SECQTY clause is an implicit specification of the current SECQTY value.
- Omission of the ERASE clause is an implicit specification of the current ERASE rule.

If you specify USING STOGROUP to convert from user-managed data sets to DB2-managed data sets:

- Omission of the PRIQTY clause is an implicit specification of PRIQTY 12, 24, 48, or 96 for a table space with 4KB, 8KB, 16KB, or 32KB pages, respectively (For LOB table spaces, the respective PRIQTY values are 200, 400, 800, and 1600).
- Omission of the SECQTY and PRIQTY clauses is an implicit specification of SECQTY 12, 24, 48, or 96 for a table space with 4KB, 8KB, 16KB, or 32KB pages, respectively. (For LOB table spaces, the respective SECQTY values are 200, 400, 800, and 1600).

If SECQTY is omitted and PRIQTY is specified, SECQTY is either 10% of PRIQTY or 3 times the page size of the table space, whichever is larger. (For LOB table spaces, SECQTY is either 10% of PRIQTY or 50 times the page size of the table space, whichever is larger.)

- Omission of the ERASE clause is an implicit specification of ERASE NO.

PRIQTY *integer*

Specifies the minimum primary space allocation for a DB2-managed data set of the table space or partition. This clause can be specified only if the data set is managed by DB2, and if one of the following is true:

- USING STOGROUP is specified.
- A USING clause is not specified.

If PRIQTY is specified, the primary space allocation is at least *n* kilobytes, where *n* is the value of *integer* with the following exceptions:

- For 4KB page sizes, if *integer* is less than 12, *n* is 12.
- For 8KB page sizes, if *integer* is less than 24, *n* is 24.
- For 16KB page sizes, if *integer* is less than 48, *n* is 48.
- For 32KB page sizes, if *integer* is less than 96, *n* is 96.
- For any page size, if *integer* is greater than 4194304 kilobytes, *n* is 4194304.

For LOB table spaces, the exceptions are:

- For 4KB pages sizes, if *integer* is less than 200, *n* is 200.
- For 8KB pages sizes, if *integer* is less than 400, *n* is 400.
- For 16KB pages sizes, if *integer* is less than 800, *n* is 800.
- For 32KB pages sizes, if *integer* is less than 1600, *n* is 1600.
- For any page size, if *integer* is greater than 4194304 kilobytes, *n* is 4194304.

```
#  
#  
# If the running DB2 is under DFP 1.5, the maximum value allowed for PRIQTY is  
# 65GB (67108864 kilobytes); otherwise, the existing maximum value of 4GB  
# (4194304 kilobytes) applies.
```

If USING STOGROUP is specified and PRIQTY is omitted, the value of PRIQTY is the default specified in the description of USING STOGROUP.

DB2 specifies the primary space allocation to access method services using the smallest multiple of *p*KB not less than *n*, where *p* is the page size of the table space. The allocated space can be greater than the amount of space requested by DB2. For example, it could be the smallest number of tracks that will accommodate the request. To more closely estimate the actual amount of storage, see the description of the DEFINE CLUSTER command in *DFSMS/MVS: Access Method Services for the Integrated Catalog*.

At least one of the volumes of the identified storage group must have enough available space for the primary quantity. Otherwise, the primary space allocation will fail.

See “Altering storage attributes” on page 477 to determine how and when changes to PRIQTY take effect.

SECQTY *integer*

Specifies the minimum secondary space allocation for a DB2-managed data set of the table space or partition. This clause can be specified only if the data set is managed by DB2, and if one of the following is true:

- USING STOGROUP is specified.
- A USING clause is not specified.

If SECQTY is specified, the secondary space allocation is at least *n* kilobytes, where *n* is the value of *integer* with the following exceptions:

If *integer* is greater than 4194304 kilobytes, *n* is 4194304. A value of 0 for *integer* indicates that no data set can be extended.

For LOB table spaces the exceptions are:

ALTER TABLESPACE

- For 4KB page sizes, if *integer* is greater than 0 and less than 200, *n* is 200.
- For 8KB page sizes, if *integer* is greater than 0 and less than 400, *n* is 400.
- For 16KB page sizes, if *integer* is greater than 0 and less than 800, *n* is 800.
- For 32KB page sizes, if *integer* is greater than 0 and less than 1600, *n* is 1600.
- For any page size, if *integer* is greater than 4194304 kilobytes, *n* is 4194304.

The maximum value allowed for SECQTY is 4GB (4194304 kilobytes).

If USING STOGROUP is specified and SECQTY is omitted, the value of SECQTY is the default specified in the description of USING STOGROUP.

DB2 specifies the secondary space allocation to access method services using the smallest multiple of *pKB* not less than *n*, where *p* is the page size of the table space. The allocated space can be greater than the amount of space requested by DB2. For example, it could be the smallest number of tracks that will accommodate the request. To more closely estimate the actual amount of storage, see the description of the DEFINE CLUSTER command in *DFSMS/MVS: Access Method Services for the Integrated Catalog*.

See “Altering storage attributes” on page 477 to determine how and when changes to SECQTY take effect.

ERASE

Indicates whether the DB2-managed data sets for the table space or partition are to be erased before they are deleted during the execution of a utility or an SQL statement that drops the table space.

NO

Does not erase the data sets. Operations involving data set deletion will perform better than ERASE YES. However, the data is still accessible, though not through DB2.

YES

Erases the data sets. As a security measure, DB2 overwrites all data in the data sets with zeros before they are deleted.

This clause can be specified only if the data set is managed by DB2, and if one of the following is true:

- USING STOGROUP is specified.
- A USING clause is not specified.

If you specify ERASE, the table space or partition must be in the stopped state when the ALTER TABLESPACE statement is executed. See “Altering storage attributes” on page 477 to determine how and when changes take effect.

COMPRESS

Specifies whether data compression applies to the rows of the table space or partition. Do not specify COMPRESS for a LOB table space.

YES

Specifies data compression. The rows are not compressed until the LOAD or REORG utility is run on the table in the table space or partition.

NO

Specifies no data compression. Inserted rows will not be compressed. Updated rows will be decompressed. The dictionary used for compression will be erased when the LOAD REPLACE, LOAD RESUME NO, or REORG

utility is run. See Part 5 (Volume 2) of *DB2 Administration Guide* for more information about the dictionary and data compression.

GBPCACHE

In a data sharing environment, specifies what pages of the table space or partition are written to the group buffer pool in a data sharing environment. In a non-data-sharing environment, you can specify GBPCACHE for a table space other than one in a work file or TEMP database, but it is ignored. Do not specify GBPCACHE for a table space in a work file database or in a TEMP database in either environment (data sharing or not). In addition, you cannot alter the GBPCACHE value of some DB2 catalog table spaces; for a list of these table spaces, see “SQL statements allowed on the catalog” on page 1011.

CHANGED

When there is inter-DB2 R/W interest on the table space or partition, updated pages are written to the group buffer pool. When there is no inter-DB2 R/W interest, the group buffer pool is not used. Inter-DB2 R/W interest exists when more than one member in the data sharing group has the table space or partition open, and at least one member has it open for update.

If the table space is in a group buffer pool that is defined to be used only for cross-validation (GBPCACHE NO), CHANGED is ignored and no pages are cached to the group buffer pool.

ALL

Indicates that pages are to be cached in the group buffer pool as they are read in from DASD.

Exception: In the case of a single updating DB2 when no other DB2s have any interest in the page set, no pages are cached in the group buffer pool.

Hiperpools are not used for table spaces or partitions that are defined with GBPCACHE ALL.

If the table space is in a group buffer pool that is defined to be used only for cross-validation (GBPCACHE NO), ALL is ignored and no pages are cached to the group buffer pool.

SYSTEM

Indicates that only changed system pages within the LOB table space are to be cached to the group buffer pool. A system page is a space map page or any other page that does not contain actual data values.

SYSTEM is the default for a LOB table space. Use SYSTEM only for a LOB table space.

NONE

Indicates that no pages are to be cached to the group buffer pool. DB2 uses the group buffer pool only for cross-validation.

If you specify NONE, the table space or partition must not be in recover pending status when the ALTER TABLESPACE statement is executed.

If you specify GBPCACHE in a data sharing environment, the table space or partition must be in the stopped state when the ALTER TABLESPACE statement is executed.

LOCKPART

Indicates whether selective partition locking (SPL) is to be used when locking a partitioned table space.

ALTER TABLESPACE

YES If all the conditions that are required for SPL are met, specifies that only the partitions accessed will be locked. If all the conditions that are required for SPL are not met, every partition of the table space is locked. LOCKPART YES is not allowed with LOCKSIZE TABLESPACE.

NO Specifies that selective partition locking is not used. The table space is locked with a single lock on the last partition. This has the effect of locking all partitions in the table space.

To alter the LOCKPART option, you must stop the entire table space with the STOP DATABASE command. Use LOCKPART only for partitioned table spaces.

MAXROWS *integer*

Specifies the maximum number of rows that DB2 will consider placing on each data page. The integer can range from 1 through 255.

The change takes effect immediately for new rows added. However, the space class settings for some pages may be incorrect and could cause unproductive page visits. It is highly recommended to reorganize the table space after altering MAXROWS.

If you specify MAXROWS, the table space must be in the stopped state when the ALTER TABLESPACE statement is executed. Do not specify MAXROWS for a LOB table space, a table space in a work file database, or the DB2 catalog table spaces that are listed under “SQL statements allowed on the catalog” on page 1011..

TRACKMOD

Specifies whether DB2 tracks modified pages in the space map pages of the table space or partition. Do not specify TRACKMOD for a LOB table space. For a table space in a TEMP database, DB2 uses TRACKMOD NO regardless of the value specified.

YES

DB2 tracks changed pages in the space map pages to improve the performance of incremental image copy. For data sharing, changing TRACKMOD to YES causes additional SCA (Shared Communications Area) storage to be used until the next full or incremental image copy is taken or until TRACKMOD is set back to NO.

NO

DB2 does not track changed pages in the space map pages. It uses the LRSN value in each page to determine whether a page has been changed.

LOG

Specifies whether changes to a LOB column in the table space are to be written to the log. Use LOG only for a LOB table space.

YES

Indicates that changes to a LOB column are to be written to the log. You cannot use YES if the auxiliary table in the table space stores a LOB column that is greater than 1 gigabyte in length.

When you change the value of LOG to YES, the LOB table space is placed in copy pending status.

NO

Indicates that changes to a LOB column are not to be written to the log.

LOG NO has no effect on a commit or rollback operation; the consistency of the database is maintained regardless of whether the LOB value is logged. All committed changes and changes that are rolled back reflect the expected results.

Even when LOG NO is specified, changes to system pages and to the auxiliary index are logged. During the log apply operation of the RECOVER utility, LPL recovery, or GPB recovery, all LOB values that were not logged are marked invalid and cannot be accessed by a SELECT or FETCH statement. Invalid LOB values can be updated or deleted.

CCSID *ccsid-value*

Identifies the CCSID value to be used for the table space. *ccsid-value* must identify a CCSID value that is compatible with the current value of the CCSID for the table space. See “Notes” on page 389 for a list that shows the CCSID to which a given CCSID can be changed and details about changing it.

Do not specify CCSID for a LOB table space or a table space in a TEMP database.

Notes

Running utilities: You cannot execute the ALTER TABLESPACE statement while a DB2 utility has control of the table space.

Altering more than one partition: To change FREEPAGE, PCTFREE, USING, PRIQTY, SECQTY, COMPRESS, ERASE, or GBPCACHE for more than one partition, you must use separate ALTER TABLESPACE statements.

Altering storage attributes: The USING, PRIQTY, SECQTY, and ERASE clauses define the storage attributes of the table space or partition. However, if you specify USING or ERASE when altering storage attributes, the table space or partition must be in the stopped state when the ALTER TABLESPACE statement is executed. You can use a STOP DATABASE...SPACENAM... command to stop the table space or partition.

If the catalog name changes, the changes take effect after you move the data and start the table space or partition using the START DATABASE...SPACENAM... command. The catalog name can be implicitly or explicitly changed by the ALTER TABLESPACE statement. The catalog name also changes when you move the data to a different device. See the procedures for moving data in Part 2 (Volume 1) of *DB2 Administration Guide*.

Changes to the secondary space allocation (SECQTY) take effect the next time DB2 extends the data set; however, the new value is not reflected in the integrated catalog until you use the REORG, RECOVER, or LOAD REPLACE utility on the table space or partition. The changes to the other storage attributes take effect the next time the page set is reset. For a non-LOB table space, the page set is reset when you use the REORG, RECOVER, or LOAD REPLACE utilities on the table space or partition. For a LOB table space, the page set is reset when RECOVER is run on the LOB table space or LOAD REPLACE is run on its associated base table space. If there is not enough storage to satisfy the primary space allocation, a REORG might fail. If you change the primary space allocation parameters or erase rule, you can have the changes take effect earlier if you move the data before you start the table space or partition.

ALTER TABLESPACE

Altering table spaces for DB2 catalog tables: For details on altering options on catalog tables, see “SQL statements allowed on the catalog” on page 1011.

Examples

Example 1: Alter table space DSN8S71E in database DSN8D71A. CLOSE NO means that the data sets of the table space are not to be closed when there are no current users of the table space.

```
ALTER TABLESPACE DSN8D71A.DSN8S71E  
CLOSE NO;
```

Example 2: Alter table space DSN8S71D in database DSN8D71A. BP2 is the buffer pool associated with the table space. PAGE is the level at which locking is to take place.

```
ALTER TABLESPACE DSN8D71A.DSN8S71D  
BUFFERPOOL BP2  
LOCKSIZE PAGE;
```

ASSOCIATE LOCATORS

The **ASSOCIATE LOCATORS** statement gets the result set locator value for each result set returned by a stored procedure.

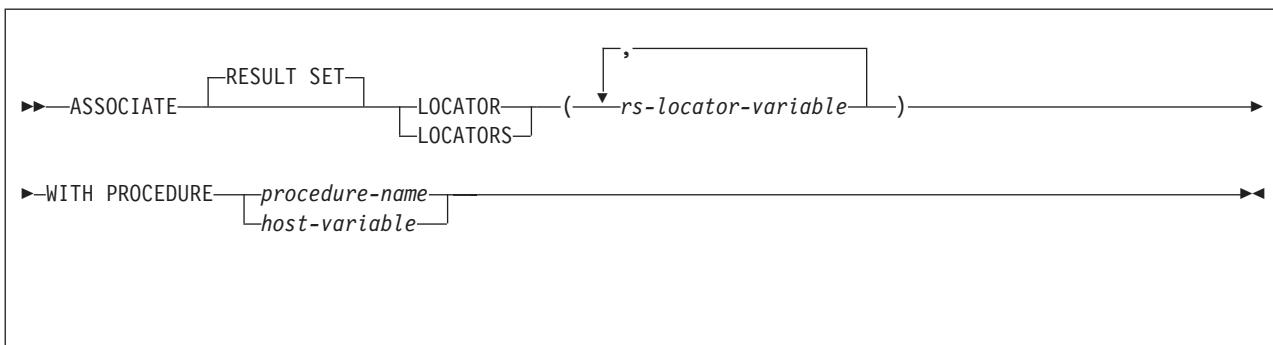
Invocation

This statement can be embedded in an application program. It is an executable statement that can be dynamically prepared. It cannot be issued interactively.

Authorization

None required.

Syntax



Description

rs-locator-variable

Identifies a result set locator variable that has been declared according to the rules for declaring result set locator variables.

One result set locator variable is required for each result set that the stored procedure returns. If the stored procedure returns fewer result sets than the number of result set locator variables specified, the extra variables are assigned a value of 0.

WITH PROCEDURE *procedure-name* or *host-variable*

IDENTIFYSRC procedure name or host variable
Identifies the stored procedure that returned result set locators by the specified procedure name or the procedure name contained in the host variable.

A procedure name is a qualified or unqualified name. Each part of the name must be composed of SBCS characters:

- A fully qualified procedure name is a three-part name. The first part is a long identifier that contains the location name that identifies the DBMS at which the procedure is stored. The second part is a short identifier that contains the schema name of the stored procedure. The last part is a long identifier that contains the name of the stored procedure. A period must separate each of the parts. Any or all of the parts can be a delimited identifier.
 - A two-part procedure name has one implicit qualifier. The implicit qualifier is the location name of the current server. The two parts identify the schema name and the name of the stored procedure. A period must separate the two parts.
 - An unqualified procedure name is a one-part name with one implicit qualifier. The implicit qualifier is the location name of the current server. An implicit

ASSOCIATE LOCATORS

schema name is not needed as a qualifier. Successful execution of the ASSOCIATE LOCATOR statement only requires that the unqualified procedure name in the statement is the same as the procedure name in the most recently executed CALL statement that was specified with an unqualified procedure name. (The implicit schema name for the unqualified name in the CALL statement is not considered in the match.) The rules for how the procedure name must be specified are described below.

If a host variable is used:

- It must be a character string variable with a length attribute that is not greater than 255.
- It must not be followed by an indicator variable.
- The value of the host variable is a specification that depends on the server. Regardless of the server, the specification must:
 - Be left justified within the host variable
 - Not contain embedded blanks
 - Be padded on the right with blanks if its length is less than that of the host variable

When the ASSOCIATE LOCATORS statement is executed, the procedure name or specification must identify a stored procedure that the requester has already invoked using the CALL statement.

The procedure name in the ASSOCIATE LOCATORS statement must be specified the same way that it was specified on the CALL statement. For example, if a two-part name was specified on the CALL statement, you must use a two-part name in the ASSOCIATE LOCATORS statement. However, there is one condition under which the names do not have to match. If the CALL statement was made with a three-part name and the current server is the same as the location in the three-part name, you can omit the location name and specify a two-part name.

Notes

More than one locator can be assigned to a result set. You can issue the same ASSOCIATE LOCATORS statement more than once with different result set locator variables.

If the number of result set locator variables that are listed in the ASSOCIATE LOCATORS statement is less than the number of locators returned by the stored procedure, all variables in the statement are assigned a value, and a warning is issued.

If the number of result set locator variables that are listed in the ASSOCIATE LOCATORS statement is greater than the number of locators returned by the stored procedure, the extra variables are assigned a value of 0.

The ASSOCIATE LOCATORS statement assigns result set locator values from the SQLVAR sections of the SQLDA to result set locator variables. For languages other than REXX, the first SQLDATA field is assigned to the first locator variable, the second SQLDATA field to the second locator variable, and so on. For REXX, the first SQLLOCATOR field is assigned to the first locator variables, the second SQLLOCATOR field to the second locator variable, and so on.

If a stored procedure is called more than once at the same location, only the most recent result sets are accessible.

If the ASSOCIATE LOCATORS statement contains host variables, the following conditions apply:

- If the statement is executed statically, the contents of the host variables are assumed to be in the encoding scheme that was specified in the ENCODING parameter when the package or plan that contains the statement was bound.
- If the statement is executed dynamically, the contents of the host variables are assumed to be in the encoding scheme that is specified in the APPLICATION ENCODING bind option.

Examples

The statements in the following examples are assumed to be in PL/I programs.

Example 1: Use result set locator variables LOC1 and LOC2 to get the result set locator values for the two result sets returned by stored procedure P1. Assume that the stored procedure is called with a one-part name from current server SITE2.

```
EXEC SQL CONNECT TO SITE2;
EXEC SQL CALL P1;
EXEC SQL ASSOCIATE RESULT SET LOCATORS (:LOC1, :LOC2)
    WITH PROCEDURE P1;
```

Example 2: Repeat the scenario in Example 1, but use a two-part name to specify an explicit schema name for the stored procedure to ensure that stored procedure P1 in schema MYSHEMA is used.

```
EXEC SQL CONNECT TO SITE2;
EXEC SQL CALL MYSHEMA.P1;
EXEC SQL ASSOCIATE RESULT SET LOCATORS (:LOC1, :LOC2)
    WITH PROCEDURE MYSHEMA.P1;
```

Example 3: Use result set locator variables LOC1 and LOC2 to get the result set locator values for the two result sets that are returned by the stored procedure named by host variable HV1. Assume that host variable HV1 contains the value SITE2.MYSHEMA.P1 and the stored procedure is called with a three-part name.

```
EXEC SQL CALL SITE2.MYSHEMA.P1;
EXEC SQL ASSOCIATE LOCATORS (:LOC1, :LOC2)
    WITH PROCEDURE :HV1;
```

The preceding example would be invalid if host variable HV1 had contained the value MYSHEMA.P1, a two-part name. For the example to be valid with that two-part name in host variable HV1, the current server must be the same as the location name that is specified on the CALL statement as the following statements demonstrate. This is the only condition under which the names do not have to be specified the same way and a three-part name on the CALL statement can be used with a two-part name on the ASSOCIATE LOCATORS statement.

```
EXEC SQL CONNECT TO SITE2;
EXEC SQL CALL SITE2.MYSHEMA.P1;
EXEC SQL ASSOCIATE LOCATORS (:LOC1, :LOC2)
    WITH PROCEDURE :HV1;
```

BEGIN DECLARE SECTION

BEGIN DECLARE SECTION

The BEGIN DECLARE SECTION statement marks the beginning of a host variable declare section.

Invocation

This statement can only be embedded in an application program. It is not an executable statement.

Authorization

None required.

Syntax

```
►►BEGIN DECLARE SECTION————►►
```

Description

The BEGIN DECLARE SECTION statement can be coded in the application program wherever variable declarations can appear in accordance with the rules of the host language. It is used to indicate the beginning of a host variable declaration section. A host variable section ends with an END DECLARE SECTION statement, described in “END DECLARE SECTION” on page 775.

The following rules are enforced by the precompiler only if the host language is C or the STDSQL(YES) precompiler option is specified:

- A variable referred to in an SQL statement must be declared within a host variable declaration section of the source program
- BEGIN DECLARE SECTION and END DECLARE SECTION statements must be paired and must not be nested.
- Host variable declaration sections can contain only host variable declarations, SQL INCLUDE statements that include host variable declarations, or DECLARE VARIABLE statements.

Notes

Host variable declaration sections are only required if the STDSQL(YES) option is specified or the host language is C. However, declare sections can be specified for any host language so that the source program can conform to IBM SQL. If declare sections are used, but not required, variables declared outside a declare section must not have the same name as variables declared within a declare section.

Example

```
EXEC SQL BEGIN DECLARE SECTION;  
      (host variable declarations)  
EXEC SQL END DECLARE SECTION;
```

CALL

The CALL statement invokes a stored procedure.

Invocation

This statement can be embedded in an application program. This statement can also be dynamically prepared, but only from an ODBC or CLI driver that supports dynamic CALL statements. IBM's ODBC and CLI drivers provide this capability.

Authorization

Invoking a stored procedure requires the EXECUTE privilege on the following:

- The stored procedure

You do not need the EXECUTE privilege on a stored procedure that was created prior to Version 6 of DB2 for OS/390 and z/OS.

- The stored procedure package and most packages that run under the stored procedure

The authorization that is required for which packages is explained in detail below under "Authorization to execute packages under the stored procedure".

Authorization to execute the stored procedure

The authorization ID that must have the EXECUTE privilege on the stored procedure depends on the form of the CALL statement:

- For static SQL programs that use the syntax `CALL procedure`, the owner of the plan or package that contains the CALL statement must have one of the following:
 - The EXECUTE privilege on the stored procedure
 - Ownership of the stored procedure
 - SYSADM authority
- For static SQL programs that use the syntax `CALL host variable` (ODBC applications use this form of the CALL statement), the authorization ID of the plan or package that contains the CALL statement must have one of the following:
 - The EXECUTE privilege on the stored procedure
 - Ownership of the stored procedure
 - SYSADM authority

The DYNAMICRULES behavior for the plan or package that contains the CALL statement determines both the authorization ID and the privilege set that is held by that authorization ID:

Run behavior

The privilege set is the union of the set of privileges held by the SQL authorization ID and each authorization ID of the process.

Bind behavior

The privilege set is the privileges that are held by the primary authorization ID of the owner of the package or plan.

Define behavior

The privilege set is the privileges that are held by the authorization ID of the owner (definer) of the stored procedure or user-defined function that issued the CALL statement.

Invoke behavior

The privilege set is the privileges that are held by the authorization ID of the invoker of the stored

CALL

procedure or user-defined function that issued the CALL statement. However, if the invoker is the primary authorization ID of the process or the CURRENT SQLID value, the privilege set is the union of the set of privileges that are held by each authorization ID.

For a list of the DYNAMICRULES values that specify run, bind, define, or invoke behavior, see Table 2 on page 44.

Authorization to execute packages under the stored procedure

The authorization that is required to run the stored procedure package and any packages that are used under the stored procedure apply to any form of the CALL statement as follows:

- ***Stored procedure package***

One of the authorization IDs that are defined below under “Set of authorization IDs” on page 485 must have at least one of the following on the stored procedure package:

- The EXECUTE privilege on the package
- Ownership of the package
- PACKADM authority for the package’s collection
- SYSADM authority

A PKLIST entry is not required for the stored procedure package.

- ***Packages other than user-defined function, trigger, and stored procedure packages***

One of the authorization IDs that are defined below under “Set of authorization IDs” on page 485 must have at least one of the following on any packages other than user-defined function and trigger packages that are used under the stored procedure:

- The EXECUTE privilege on the package
- Ownership of the package
- PACKADM authority for the package’s collection
- SYSADM authority

PKLIST entries are required for any of these packages that are used under the stored procedure.

- ***User-defined function packages and trigger packages***

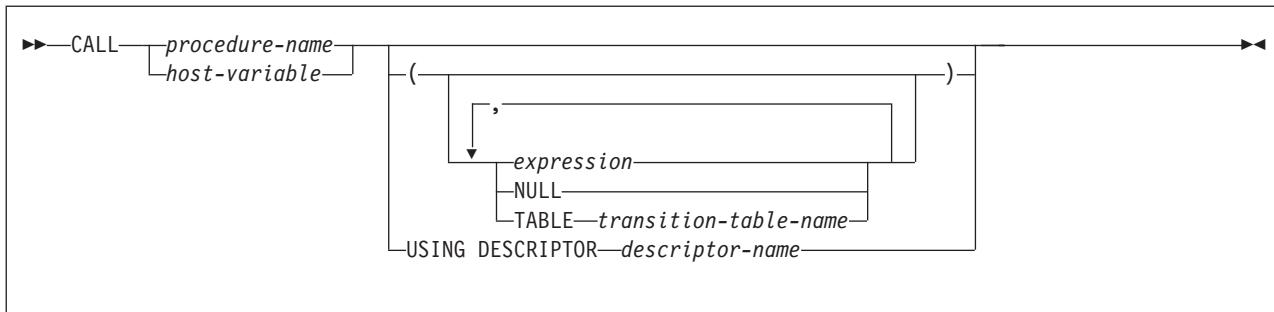
If a stored procedure or any application under the stored procedure invokes a user-defined function, DB2 requires only the owner (the definer) and not the invoker of the user-defined function to have EXECUTE authority on the user-defined function package. However, the authorization ID of the SQL statement that invokes the user-defined function must have EXECUTE authority on the function. Similarly, if a trigger is used under a stored procedure, DB2 does not require EXECUTE authority on the trigger package; however, the authorization ID of the SQL statement that activates the trigger must have EXECUTE authority on the trigger. For more information about the EXECUTE authority for user-defined functions, triggers, and user-defined function packages, see Part 3 of *DB2 Administration Guide*.

PKLIST entries are not required for any user-defined function packages or trigger packages that are used under the stored procedure.

Set of authorization IDs: DB2 checks the following authorization IDs in the order in which they are listed for the required authorization to execute the stored procedure package and any packages other than user-defined function and trigger packages as described above:

- The owner (the definer) of the stored procedure
- The owner of the plan that contains the statement that invokes the package if the application is local, the application is distributed and DB2 for OS/390 and z/OS is both the requester and the server, or the application uses Recoverable Resources Management Services attachment facility (RRSAF) and has no plan.
- The owner of the package that contains the statement that invokes the package if the application is distributed and DB2 for OS/390 and z/OS is the server but not the requester
- The authorization ID as determined by the value of the DYNAMICRULES bind option for the plan or package that contains the CALL statement if the CALL statement is in the form of CALL *host variable*

Syntax



Description

procedure-name or host-variable

Identifies the stored procedure to call. The procedure name can be specified as a character string constant or within a host variable.

A procedure name is a qualified or unqualified name. Each part of the name must be composed of SBCS characters:

- A fully qualified procedure name is a three-part name. The first part is a long identifier that contains the location name that identifies the DBMS at which the procedure is stored. The second part is a short identifier that contains the schema name of the stored procedure. The last part is a long identifier that contains the name of the stored procedure. A period must separate each of the parts. Any or all of the parts can be a delimited identifier.
- A two-part procedure name has one implicit qualifier. The implicit qualifier is the location name of the current server. The two parts identify the schema name and the name of the stored procedure. A period must separate the two parts.
- An unqualified procedure name is a one-part name with two implicit qualifiers. The first implicit qualifier is the location name of the current server. The second implicit qualifier depends on the server. If the server is DB2 for OS/390 and z/OS, the implicit qualifier is the schema name. DB2 uses the SQL path to determine the value of the schema name.

CALL

- If the procedure name is specified as a literal on the CALL statement (CALL *procedure-name*), the SQL path is the value of the PATH bind option that is associated with the calling package or plan.
- If a host variable is specified for the procedure name on the CALL statement (CALL *host-variable*), the SQL path is the value of the CURRENT PATH special register.

DB2 searches the schema names in the SQL path from left to right until a stored procedure with the specified schema name is found in the DB2 catalog. When a matching *schema.procedure-name* is found, the search stops only if the following conditions are true:

- The user is authorized to call the stored procedure.
- The number of parameters in the definition of the stored procedure matches the number of parameters specified on the CALL statement.

If the list of schemas in the SQL path is exhausted before the procedure name is resolved, an error is returned.

If a host variable is used:

- It must be a character string variable with a length attribute that is not greater than 254.
- It must not have a CLOB data type.
- It must not be followed by an indicator variable.
- The value of the host variable is a specification that depends on the server. Regardless of the server, the specification must:
 - Be left justified within the host variable
 - Not contain embedded blanks
 - Be padded on the right with blanks if its length is less than that of the host variable

In addition, the specification can:

- Contain upper and lowercase characters. Lowercase characters are not folded to uppercase.
- Use a delimited identifier for any part of the three-part procedure name.

If the server is DB2 for OS/390 and z/OS, the specification must be a procedure name as defined above.

When the CALL statement is executed, the procedure name or specification must identify a stored procedure that exists at the server.

When the package that contains the CALL statement is bound, the stored procedure that is invoked must be created if VALIDATE(BIND) is specified. Although the stored procedure does not need to be created at bind time if VALIDATE(RUN) is specified, it must be created when the CALL statement is executed.

Parameters (*expression*, **NULL**, **TABLE** *transition-table-name*)

Identifies a list of values to be passed as parameters to the stored procedure. If USING DESCRIPTOR is specified, each host variable described by the identified SQLDA is a parameter, or part of an expression that is a parameter, of the CALL statement. If host structures are not specified in the CALL statement, the *n*th parameter of the CALL statement corresponds to the *n*th parameter in the stored procedure, and the number of parameters in each must be the same. Otherwise, each reference to a host structure is replaced by a reference to

each of the variables contained in that host structure, and the resulting number of parameters must be the same as the number of parameters defined for the stored procedure.

However, a character FOR BIT DATA parameter cannot be passed as input for a parameter that is not defined as character FOR BIT DATA. Likewise, a character argument that is not FOR BIT DATA cannot be passed as input for a parameter that is defined as character FOR BIT DATA.

Each parameter of a stored procedure is described at the server. In addition to attributes such as data type and length, the description of each parameter indicates how the stored procedure uses it:

- IN means as an input value
- OUT means as an output value
- INOUT means both as an input and an output value

When the CALL statement is executed, the value of each of its parameters is assigned to the corresponding parameter of the stored procedure. In cases where the parameters of the CALL statement are not an exact match to the data types of the parameters of the stored procedure, each parameter specified in the CALL statement is converted to the data type of the corresponding parameter of the stored procedure at execution. The conversion occurs according to the same rules as assignment to columns. For details on the rules used to assign parameters, see “Assignment and comparison” on page 64.

Conversion can occur when precision, scale, length, encoding scheme, or CCSID differ between the parameter specified in the CALL statement and the data type of the corresponding parameter of the stored procedure. Conversion might occur for a character string parameter specified in the CALL statement when the corresponding parameter of the stored procedure has a different encoding scheme or CCSID. For example, an error occurs when the CALL statement passes mixed data that actually contains DBCS characters as input to a parameter of the stored procedure that is declared with an SBCS subtype. Likewise, an error occurs when the stored procedure returns mixed data that actually contains DBCS characters in the parameter of the CALL statement that has an SBCS subtype.

Control is passed to the stored procedure according to the calling conventions of the host language. When execution of the stored procedure is complete, the value of each parameter of the stored procedure is assigned to the corresponding parameter of the CALL statement defined as OUT or INOUT. If an error is returned by the procedure, OUT parameters are undefined, and INOUT parameters are unchanged.

#

expression

The parameter is the result of the specified expression, which is evaluated before the stored procedure is invoked.

If *expression* is a single host variable, the corresponding parameter of the procedure can be defined as IN, INOUT, or OUT. Otherwise, the corresponding parameter of the procedure must be defined as IN. In addition, the host variable can identify a structure. Any host variable or structure that is specified must be described in the application program according to the rules for declaring host structures and variables. A reference to a host structure is replaced by a reference to each of the variables contained in the host structure.

CALL

If the result of the expression can be the null value, either the description of the procedure must allow for null parameters or the corresponding parameter of the stored procedure must be defined as OUT.

The following additional rules apply depending on how the corresponding parameter was defined in the CREATE PROCEDURE statement for the procedure:

- IN *expression* can contain references to multiple host variables. In addition to the rules stated in “Expressions” on page 111, *expression* cannot include a column name or column function or a user-defined function that is sourced on a column function.
- INOUT or OUT *expression* can only be a single host variable.

NULL

The parameter is a null value. The corresponding parameter of the procedure must be defined as IN and the description of the procedure must allow for null parameters.

TABLE *transition-table-name*

The parameter is a transition table, and it is passed to the procedure as a table locator. You can use the CALL statement with the TABLE clause only within the definition of the triggered action of a trigger. The name of a transition table must be specified in the CALL statement if the corresponding parameter of the procedure was defined in the TABLE LIKE clause of the CREATE PROCEDURE statement. For information about creating a trigger, see “CREATE TRIGGER” on page 699 and *DB2 Application Programming and SQL Guide*.

There is no effect on the transition table on the return from the procedure regardless of whether the parameter was defined as IN, INOUT, or OUT.

USING DESCRIPTOR *descriptor-name*

Identifies an SQLDA that contains a valid description of the host variables that are to be passed as parameters to the stored procedure. If the stored procedure has no parameters, an SQLDA is ignored.

Before the CALL statement is processed, the user must set the following fields in the SQLDA:

- SQLN to indicate the number of SQLVAR occurrences provided in the SQLDA. This number must not be less than SQLD. This field is not part of the REXX SQLDA and therefore does not need to be set for REXX programs.
- SQLDABC to indicate the number of bytes of storage allocated for the SQLDA. This number must not be less than SQLN*44+16. This field is not part of the REXX SQLDA and therefore does not need to be set for REXX programs.
- SQLD to indicate the number of variables used in the SQLDA when processing the statement. This number must be the same as the number of parameters of the stored procedure.
- SQLVAR occurrences to indicate the attributes of the variables.

There are additional considerations for setting the fields of the SQLDA when a variable that is passed as a parameter to the stored procedure has a LOB data type or is a LOB locator. For more information, see “SQL descriptor area (SQLDA)” on page 986.

The SQL CALL statement ignores distinct type information in the SQLDA. Only the base SQL type information is used to process the input and output parameters described by the SQLDA.

See “Identifying an SQLDA in C or C++” on page 1003 for how to represent *descriptor-name* in C.

Notes

Improving performance: The capability of calling stored procedures is provided to improve the performance of DRDA distributed access (DB2 private protocol access is not supported). The capability is also useful for local operations. The server can be the local DB2. In which case, packages are still required.

All values of all parameters are passed from the requester to the server. To improve the performance of this operation, host variables that correspond to OUT parameters and have lengths of more than a few bytes should be set to null before the CALL statement is executed.

Using the *CALL* statement in a trigger: When a trigger issues a CALL statement to invoke a stored procedure, the parameters that are specified in the CALL statement cannot be host variables and the USING DESCRIPTOR clause cannot be specified.

Nesting *CALL* statements: A program that is executing as a stored procedure, a user-defined function, or a trigger can issue a CALL statement. When a stored procedure, user-defined function, or trigger calls a stored procedure, user-defined function, or trigger, the call is considered to be nested. Stored procedures, user-defined functions, and triggers can be nested up to 16 levels deep on a single system. Nesting can occur within a single DB2 subsystem or when a stored procedure or user-defined function is invoked at a remote server.

If a stored procedure returns any query result sets, the result sets are returned to the caller of the stored procedure. If the SQL CALL statement is nested, the result sets are visible only to the program that is at the previous nesting level. For example, Figure 6 illustrates a scenario in which a client program calls stored procedure PROCA, which in turn calls stored procedure PROCB. Only PROCA can access any result sets that PROCB returns; the client program has no access to the query result sets. The number of query result sets that PROCB returns does not count toward the maximum number of query results that PROCA can return.

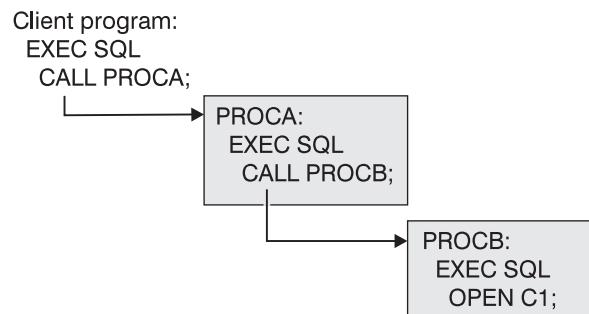


Figure 6. Nested CALL statements

CALL

Some stored procedures cannot be nested. A stored procedure, user-defined function, or trigger cannot call a stored procedure that is defined with the COMMIT ON RETURN attribute. A stored procedure can call another stored procedure only if they execute in the same type of address space; they must both execute in a DB2-established address space or in a WLM-established address space.

|
| **Using host variables:** If the CALL statement contains host variables, the contents of the host variables are assumed to be in the encoding scheme that was specified in the ENCODING parameter when the package or plan that contains the statement was bound.
|
|

Example

A PL/I application has been precompiled on DB2 ALPHA and a package was created at DB2 BETA with the BIND subcommand. A CREATE PROCEDURE statement was issued at BETA to define the procedure SUMARIZE, which allows nulls and has two parameters. The first parameter is defined as IN and the second parameter is defined as OUT. Some of the statements that the application that runs at DB2 ALPHA might use to call stored procedure SUMARIZE include:

```
EXEC SQL CONNECT TO BETA;  
V1 = 528671;  
IV = -1;  
EXEC SQL CALL SUMARIZE(:V1,:V2 INDICATOR :IV);
```

CLOSE

The CLOSE statement closes a cursor. If a temporary copy of a result table was created when the cursor was opened, that table is destroyed.

Invocation

This statement can only be embedded in an application program. It is an executable statement that cannot be dynamically prepared.

Authorization

See “DECLARE CURSOR” on page 718 for the authorization required to use a cursor.

Syntax

```
►►—CLOSE—cursor-name—►►
```

Description

cursor-name

Identifies the cursor to be closed. The cursor name must identify a declared cursor as explained in “DECLARE CURSOR” on page 718. When the CLOSE statement is executed, the cursor must be in the open state.

Notes

Any open cursors of an application process option are implicitly closed at the termination of a unit of work. However, explicitly closing cursors as soon as possible can improve performance. CLOSE does not cause a commit or rollback operation.

The cursor could have been allocated. See “ALLOCATE CURSOR” on page 386.

Example

A cursor is used to fetch one row at a time into the application program variables DNUM, DNAME, and MNUM. Finally, the cursor is closed. If the cursor is reopened, it is again located at the beginning of the rows to be fetched.

```

EXEC SQL DECLARE C1 CURSOR FOR
    SELECT DEPTNO, DEPTNAME, MGRNO
    FROM DSN8710.DEPT
    WHERE ADMRDEPT = 'A00'
    END-EXEC.

    EXEC SQL OPEN C1 END-EXEC.

    EXEC SQL FETCH C1 INTO :DNUM, :DNAME, :MNUM END-EXEC.

    IF SQLCODE = 100
        PERFORM DATA-NOT-FOUND
    ELSE
        PERFORM GET-REST-OF-DEPT
        UNTIL SQLCODE IS NOT EQUAL TO ZERO.

    EXEC SQL CLOSE C1 END-EXEC.

```

CLOSE

```
GET-REST-OF-DEPT.  
EXEC SQL FETCH C1 INTO :DNUM, :DNAME, :MNUM END-EXEC.
```

COMMENT

The COMMENT statement adds or replaces comments in the descriptions of various objects in the DB2 catalog at the current server. The objects that can have comments are aliases, columns, distinct types, stored procedures, tables, triggers, user-defined functions, views, and indexes.

Invocation

This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared.

Authorization

For a comment on a table, view, alias, index, or column, the privilege set that is defined below must include at least one of the following:

- Ownership of the table, view, alias, or index
- Ownership of the table on which the index is defined (indexes only)
- DBADM authority for its database (tables and indexes only)
- SYSADM or SYSCTRL authority

For a comment on a distinct type, stored procedure, trigger, or user-defined function, the privilege set that is defined below must include at least one of the following:

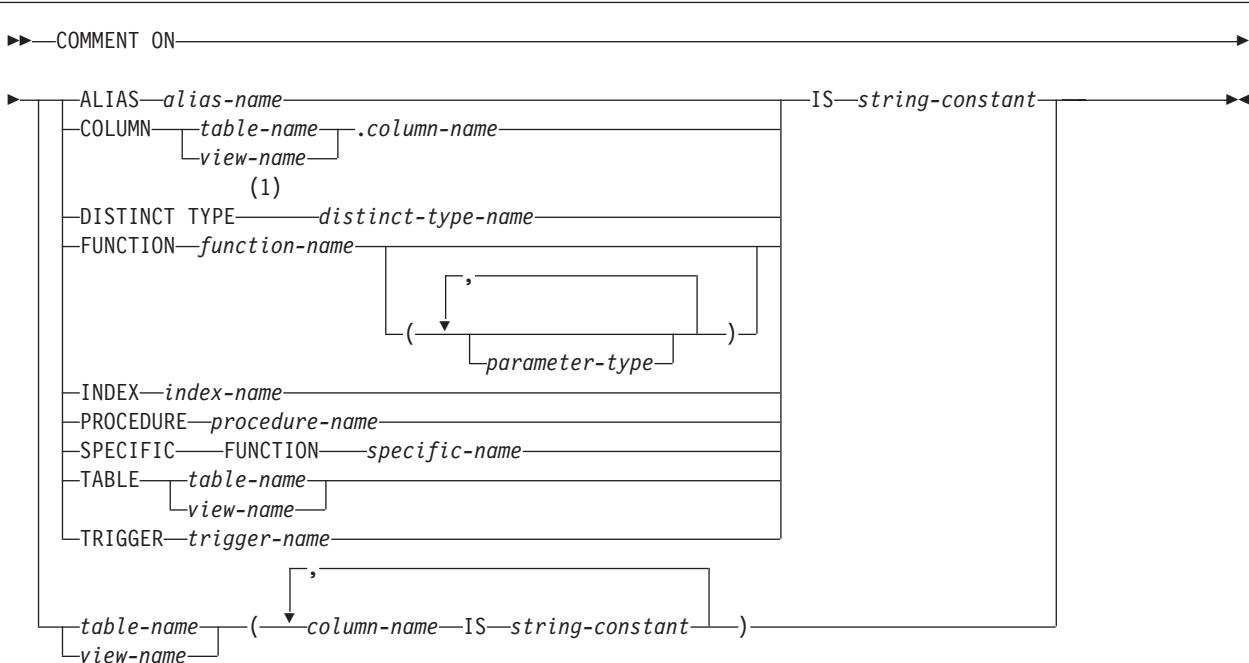
- Ownership of the distinct type, stored procedure, trigger, or user-defined function
- The ALTERIN privilege for the schema or all schemas (for the addition of comments)
- SYSADM or SYSCTRL authority

The authorization ID that matches the schema name implicitly has the ALTERIN privilege on the schema.

Privilege set: If the statement is embedded in an application program, the privilege set is the privileges that are held by the authorization ID of the owner of the plan or package. If the statement is dynamically prepared, the privilege set is determined by the DYNAMICRULES behavior in effect (run, bind, define, or invoke) and is summarized in Table 40 on page 382. (For more information on these behaviors, including a list of the DYNAMICRULES bind option values that determine them, see “Authorization IDs and dynamic SQL” on page 43.)

COMMENT

Syntax



Notes:

- 1 DATA can be used as a synonym for DISTINCT.

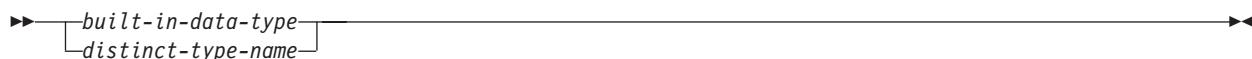
parameter-type



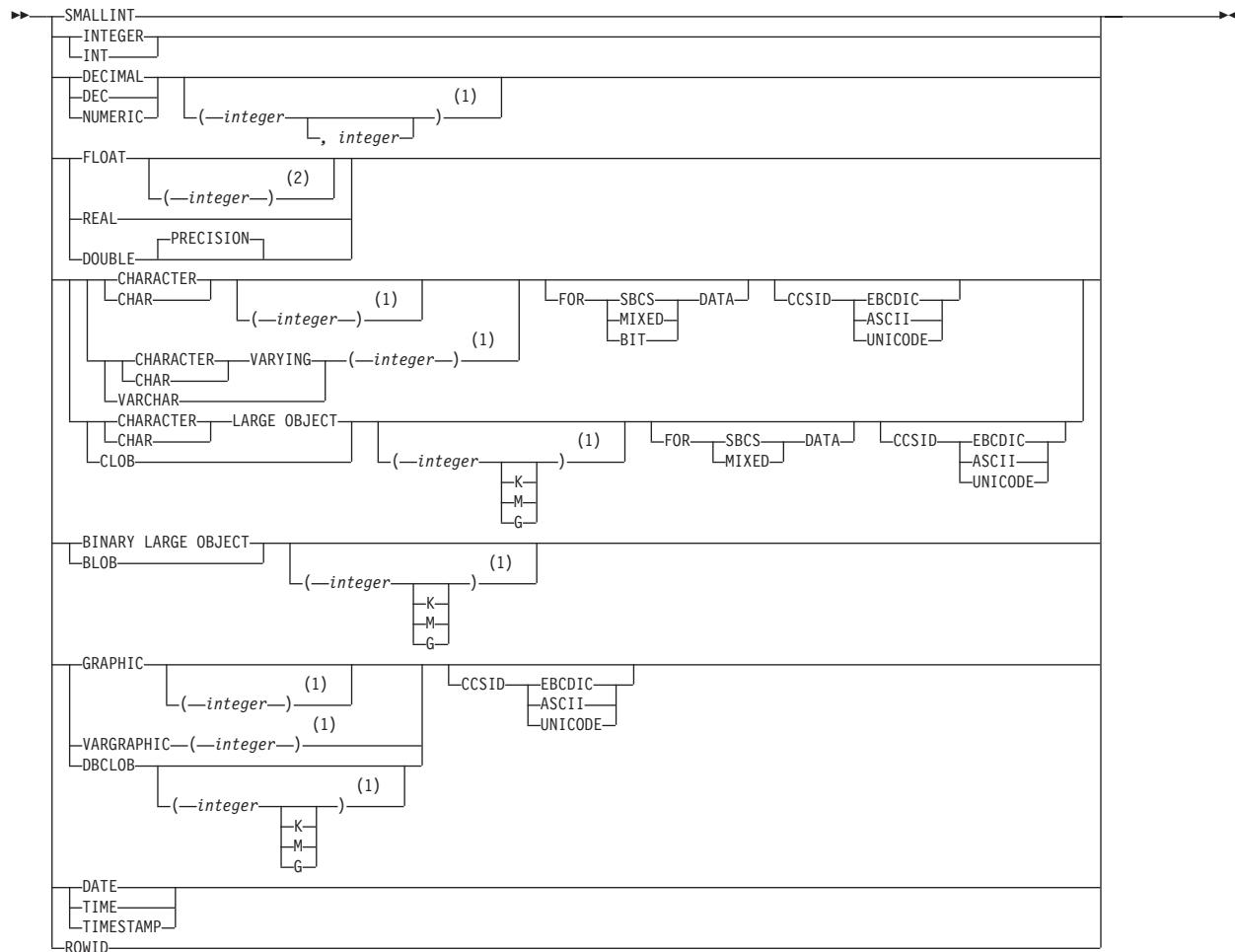
Notes:

- 1 AS LOCATOR can be specified only for a LOB data type or a distinct type that is based on a LOB data type.

data-type



built-in-data-type



Notes:

- 1 The values that are specified for length, precision, or scale attributes must match the values that were specified when the function was created. Coding specific values is optional. Empty parentheses, (), can be used instead to indicate that DB2 ignores the attributes when determining whether data types match.
- 2 The value that is specified does not have to match the value that was specified when the function was created because matching is based on data type (REAL or DOUBLE). $1 \leq integer \leq 21$ indicates REAL and $22 \leq integer \leq 53$ indicates DOUBLE. Coding a specific value is optional. Empty parentheses cannot be used.

Description

ALIAS *alias-name*

Identifies the alias to which the comment applies. *alias-name* must identify an alias that exists at the current server. The comment is placed in the REMARKS column of the SYSIBM.SYSTABLES catalog table for the row that describes the alias.

COMMENT

COLUMN *table-name.column-name or view-name.column-name*

Identifies the column to which the comment applies. The name must identify a column of a table or view that exists at the current server. The name must not identify a column of a declared temporary table. The comment is placed into the REMARKS column of the SYSIBM.SYSCOLUMNS catalog table, for the row that describes the column.

Do not use TABLE or COLUMN to comment on more than one column in a table or view. Give the table or view name and then, in parentheses, a list in the form:

```
column-name IS string-constant,  
column-name IS string-constant,...
```

The column names must not be qualified, each name must identify a column of the specified table or view, and that table or view must exist at the current server.

DISTINCT TYPE *distinct-type-name*

Identifies the distinct type to which the comment applies. *distinct-type-name* must identify a distinct type that exists at the current server. The comment is placed in the REMARKS column of the SYSIBM.SYSDATATYPES catalog table for the row that describes the distinct type.

FUNCTION

Identifies the function to which the comment applies. The function must exist at the current server, and it must be a function that was defined with the CREATE FUNCTION statement or a cast function that was generated by a CREATE DISTINCT TYPE statement. The comment is placed in the REMARKS column of the SYSIBM.SYSROUTINES catalog table for the row that describes the function.

If the function was defined with a table parameter (the LIKE TABLE was specified in the CREATE FUNCTION statement to indicate that one of the input parameters is a transition table), the function signature cannot be used to identify the function. Instead, identify the function with its function name, if unique, or with its specific name.

FUNCTION *function-name*

Identifies the particular function, and is valid only if there is exactly one function with *function-name*.

FUNCTION *function-name (parameter-type,...)*

Provides the function signature, which uniquely identifies the function. If *function-name()* is specified, the function that is identified must have zero parameters.

function-name

Identifies the name of the function.

(parameter-type,...)

Identifies the parameters of the function.

The data types of the parameters must match the data types that were specified on the CREATE FUNCTION statement in the corresponding position. The number of data types and the logical concatenation of the data types are used to identify the specific function.

For data types that have a length, precision, or scale attribute, you can specify a value or use a set of empty parentheses:

- Empty parentheses indicate that DB2 ignores the attribute when determining whether the data types match.

#

FLOAT cannot be specified with empty parentheses because its parameter value indicates different data types (REAL or DOUBLE).

- If you use a specific value for a length, precision, or scale attribute, the value must exactly match the value that was specified (implicitly or explicitly) in the CREATE FUNCTION statement.

The specific value for FLOAT(*n*) does not have exactly match the defined value of the source function because 1<=n<= 21 indicates REAL and 22<=n<=53 indicates DOUBLE. Matching is based on whether the data type is REAL or DOUBLE.

- If length, precision, or scale is not explicitly specified, and empty parentheses are not specified, the default length of the data type is implied. For example:

CHAR	CHAR(1)
GRAPHIC	GRAPHIC(1)
DECIMAL	DECIMAL(5,0)
FLOAT	DOUBLE (length of 8)

The implicit length must exactly match the value that was specified (implicitly or explicitly) in the CREATE FUNCTION statement. For a complete list of the default lengths of data types, see “CREATE TABLE” on page 653.

For data types with a subtype or encoding scheme attribute, specifying the FOR DATA clause or CCSID clause is optional. Omission of either clause indicates that DB2 ignores the attribute when determining whether the data types match. If you specify either clause, it must match the value that was implicitly or explicitly specified in the CREATE FUNCTION statement.

SPECIFIC FUNCTION *specific-name*

Identifies the particular function using the specific name either specified or defaulted to when the function was created.

INDEX *index-name*

Identifies the index to which the comment applies. *index-name* must identify an index that exists at the current server. The comment is placed in the REMARKS column of the SYSIBM.SYSINDEXES catalog table for the row that describes the index.

PROCEDURE *procedure-name*

Identifies the stored procedure to which the comment applies. *procedure-name* must identify a stored procedure that has been defined with the CREATE PROCEDURE statement at the current server. The comment is placed in the REMARKS column of the SYSIBM.SYSROUTINES catalog table for the row that describes the stored procedure.

TABLE *table-name* or *view-name*

Identifies the table or view to which the comment applies. *table-name* or *view-name* must identify a table, auxiliary table, or view that exists at the current server. *table-name* must not identify a declared temporary table. The comment is placed in the REMARKS column of the SYSIBM.SYSTABLES catalog table for the row that describes the table or view.

TRIGGER *trigger-name*

Identifies the trigger to which the comment applies. *trigger-name* must identify a trigger that exists at the current server. The comment is placed in the REMARKS column of the SYSIBM.SYSTRIGGERS catalog table for the row that describes the trigger.

COMMENT

IS *string-constant*

Introduces the comment that you want to make. *string-constant* can be any SQL character string constant of up to 254 characters.

Examples

Example 1: Enter a comment on table DSN8710.EMP.

```
COMMENT ON TABLE DSN8710.EMP  
IS 'REFLECTS 1ST QTR 81 REORG';
```

Example 2: Enter a comment on view DSN8710.VDEPT.

```
COMMENT ON TABLE DSN8710.VDEPT  
IS 'VIEW OF TABLE DSN8710.DEPT';
```

Example 3: Enter a comment on the DEPTNO column of table DSN8710.DEPT.

```
COMMENT ON COLUMN DSN8710.DEPT.DEPTNO  
IS 'DEPARTMENT ID - UNIQUE';
```

Example 4: Enter comments on the two columns in table DSN8710.DEPT.

```
COMMENT ON DSN8710.DEPT  
(MGRNO IS 'EMPLOYEE NUMBER OF DEPARTMENT MANAGER',  
ADMREDEPT IS 'DEPARTMENT NUMBER OF ADMINISTERING DEPARTMENT');
```

Example 5: Assume that you are SMITH and that you created the distinct type DOCUMENT in schema SMITH. Enter comments on DOCUMENT.

```
COMMENT ON DISTINCT TYPE DOCUMENT  
IS 'CONTAINS DATE, TABLE OF CONTENTS, BODY, INDEX, and GLOSSARY';
```

Example 6: Assume that you are SMITH and you know that ATOMIC_WEIGHT is the only function with that name in schema CHEM. Enter comments on ATOMIC_WEIGHT.

```
COMMENT ON FUNCTION CHEM.ATOMIC_WEIGHT  
IS 'TAKES ATOMIC NUMBER AND GIVES ATOMIC WEIGHT';
```

Example 7: Assume that you are SMITH and that you created the function CENTER in schema SMITH. Enter comments on CENTER, using the signature to uniquely identify the function instance.

```
COMMENT ON FUNCTION CENTER (INTEGER, FLOAT)  
IS 'USES THE CHEBYCHEV METHOD';
```

Example 8: Assume that you are SMITH and that you created another function named CENTER in schema JOHNSON. You gave the function the specific name FOCUS97. Enter comments on CENTER, using the specific name to identify the function instance.

```
COMMENT ON SPECIFIC FUNCTION JOHNSON.FOCUS97  
IS 'USES THE SQUARING TECHNIQUE';
```

Example 9: Assume that you are SMITH and that stored procedure OSMOSIS is in schema BIOLOGY. Enter comments on OSMOSIS.

```
COMMENT ON PROCEDURE BIOLOGY.OSMOSIS  
IS 'CALCULATIONS THAT MODEL OSMOSIS';
```

Example 11: Assume that you are SMITH and that trigger BONUS is in your schema. Enter comments on BONUS.

```
COMMENT ON TRIGGER BONUS  
IS 'LIMITS BONUSES TO 10% OF SALARY';
```

COMMIT

The COMMIT statement ends a unit of recovery and commits the relational database changes that were made in that unit of recovery. If relational databases are the only recoverable resources used by the application process, COMMIT also ends the unit of work.

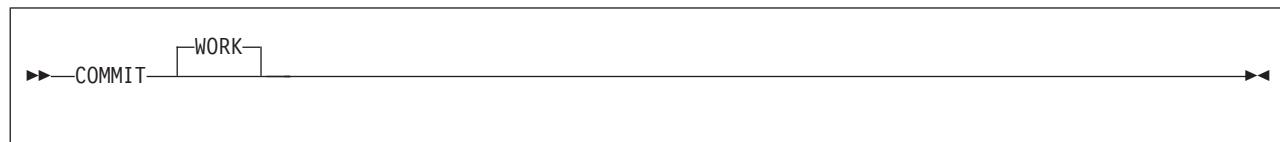
Invocation

This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared. It cannot be used in the IMS or CICS environment.

Authorization

None required.

Syntax



Description

The unit of recovery in which the statement is executed is ended and a new unit of recovery is effectively started for the process. All changes made by ALTER, COMMENT ON, CREATE, DELETE, DROP, EXPLAIN, GRANT, INSERT, LABEL ON, RENAME, REVOKE, and UPDATE statements executed during the unit of recovery are committed, and all savepoints that were set within the unit of recovery are released. SQL connections are ended when any of the following apply:

- The connection is in the release pending status
- The connection is not in the release pending status but it is a remote connection and:
 - The DISCONNECT(AUTOMATIC) bind option is in effect, or
 - The DISCONNECT(CONDITIONAL) bind option is in effect and an open WITH HOLD cursor is not associated with the connection.

For existing connections, all LOB locators are disassociated, except for those locators for which a HOLD LOCATOR statement has been issued without a corresponding FREE LOCATOR statement. All open cursors that were declared without the WITH HOLD option are closed. All open cursors that were declared with the WITH HOLD option are preserved, along with any SELECT statements that were prepared for those cursors. All other prepared statements are destroyed unless dynamic caching is enabled for your system. In that case, all prepared SELECT, INSERT, UPDATE, and DELETE statements that are bound with DYNAMICKEEP(YES) are kept past the commit.

Prepared statements cannot be kept past a commit if any of the following is true:

- SQL RELEASE has been issued for that site.
- Bind option DISCONNECT(AUTOMATIC) was used.
- Bind option DISCONNECT(CONDITIONAL) was used and there are no hold cursors for that site.

COMMIT

All implicitly acquired locks are released, except:

- Locks that are required for the cursors that were not closed
- Table and table space locks when the RELEASE parameter on the bind command was not RELEASE(COMMIT)
- LOB locks and LOB table space locks that are required for held LOB locators

For an explanation of the duration of explicitly acquired locks, see Part 5 (Volume 2) of *DB2 Administration Guide*.

All rows of every created temporary table of the application process are deleted with the exception that the rows of a created temporary table are not deleted if any program in the application process has an open WITH HOLD cursor that is dependent on that table. In addition, if RELEASE(COMMIT) is in effect, the logical work files for the created temporary tables whose rows are deleted are also deleted.

All rows of every declared temporary table of the application process are deleted with these exceptions:

- The rows of a declared temporary table that is defined with the ON COMMIT PRESERVE ROWS attribute are not deleted.
- The rows of a declared temporary table that is defined with the ON COMMIT DELETE ROWS attribute are not deleted if any program in the application process has an open WITH HOLD cursor that is dependent on that table.

Notes

The COMMIT statement cannot be used in the IMS or CICS environment. To cause a commit operation in these environments, SQL programs must use the call prescribed by their transaction manager. The effect of these commit operations on DB2 data is the same as that of the SQL COMMIT statement.

In all DB2 environments, the normal termination of a process is an implicit commit operation.

|
| The COMMIT statement cannot be used in a stored procedure if the procedure is in
| the calling chain of a user-defined function or a trigger or if the caller is using a
| two-phase commit.

Issuing a COMMIT statement might cause special registers to be re-initialized.
Whether a special register is affected by a commit depends on whether the special
register has been explicitly set within the application process. For example, assume
that the PATH special register has not been explicitly set with a SET PATH
statement in the application process. After a commit, the value of PATH is
re-initialized. For information on the initialization of PATH, which can take the
current value of CURRENT SQLID into consideration, see "CURRENT PATH" on
page 88.

Example

Commit all DB2 database changes made since the unit of recovery was started.

```
COMMIT WORK;
```

CONNECT

The CONNECT statement connects the application process to a designated server. This server is then the *current server* for the process. The statement can be a Type 1 or a Type 2 CONNECT statement. “When an application process has a current server” on page 502 describes what happens when the process has a current server and “Establishing a different server” on page 503 describes using a different server than the current server.

CONNECT (Type 1) and CONNECT (Type 2) differences

The two types of CONNECT statements have same syntax but different semantics, as summarized below. Both types of the CONNECT statement are used for DRDA access; however, the level of function available for each type is different. For a description of an individual type of CONNECT, see:

“CONNECT (Type 1)” on page 504

“CONNECT (Type 2)” on page 510

The following table summarizes the differences between CONNECT (Type 1) and CONNECT (Type 2) rules:

Table 42. CONNECT (Type 1) and CONNECT (Type 2) differences

Type 1 rules	Type 2 rules
CONNECT statements can be executed only when the application process is in the connectable state. Only one CONNECT statement can be executed within the same unit of work.	More than one CONNECT statement can be executed within the same unit of work. There are no rules about the connectable state.
If a CONNECT statement fails because the application process is not in the connectable state, the SQL connection status of the application process is unchanged.	If a CONNECT statement fails, the current SQL connection is unchanged and any subsequent SQL statements are executed by that server, unless the failure prevents the execution of SQL statements by that server.
If a CONNECT statement fails for any other reason, the application process is placed in the unconnected state.	
CONNECT ends any existing connections of the application process. Accordingly, CONNECT also closes any open cursors of the application process. (The only cursors that can possibly be open when CONNECT is successfully executed are those defined with the WITH HOLD option.)	CONNECT does not end connections and does not close cursors.

CONNECT

Table 42. CONNECT (Type 1) and CONNECT (Type 2) differences (continued)

Type 1 rules	Type 2 rules
A CONNECT to the current server is executed like any other CONNECT (Type 1) statement.	If the SQLRULES(STD) bind option is in effect, a CONNECT to an existing SQL connection of the application process is an error. Thus, a CONNECT to the current server is an error. For example, an error occurs if the first CONNECT is a CONNECT TO <i>x</i> where <i>x</i> is the local DB2.
	If the SQLRULES(DB2) bind option is in effect, a CONNECT to an existing SQL connection is not an error. Thus, if <i>x</i> is an existing SQL connection of the application process, CONNECT TO <i>x</i> makes <i>x</i> its current connection. If <i>x</i> is already the current connection, CONNECT TO <i>x</i> has no effect on the state of any connections.

Determining the CONNECT rules that apply: The following table explains how to determine the CONNECT rules that apply:

Table 43. Determining the CONNECT rules that apply

If the precompiler option...	is...	then the rules for...
CONNECT(1)	specified	CONNECT (Type 1) apply
CONNECT(2)	specified	CONNECT (Type 2) apply
CONNECT	omitted	CONNECT (Type 2) implicitly apply.

The CONNECT rules that apply to an application process are determined by the first CONNECT statement that is executed (successfully or unsuccessfully) by that application process:

- If it is a CONNECT (Type 1), then CONNECT (Type 1) rules apply and CONNECT (Type 2) statements are invalid.
- If it is a CONNECT (Type 2), then CONNECT (Type 2) rules apply and CONNECT (Type 1) statements are invalid.

Programs containing CONNECT statements that are precompiled with different CONNECT precompiler options cannot execute as part of the same application process. An error occurs when an attempt is made to execute the invalid CONNECT statement.

When an application process has a current server

The *current server* is the DBMS to which an application is actively connected. The following rules apply when an application process has a current server:

- Static SQL statements executed by the application process are taken from a package that was bound at that server. However, this execution does not apply to SQL statements, such as CONNECT and RELEASE, that are not represented in packages. Furthermore, if the current server is the local DB2 subsystem, SQL statements can also be taken from a DBRM that has been bound with the application plan. This is the case if the CURRENT PACKAGESET special register is blank and the name of the application program executing the SQL statement is the same as the name of a DBRM.

- The package from which SQL statements are taken is determined by the name of the application program executing the SQL statement, the package list of the application plan, and CURRENT PACKAGESET.

The last part of the package name is the same as the name of the application program, unless a member name is specified during the bind process along with the DBRMLIB DD statement. The qualifier of the package name (the collection ID) can be determined by the package list or by the CURRENT PACKAGESET special register. For more information, see “SET CURRENT PACKAGESET” on page 912.

- Dynamic and static SQL statements that refer to objects at the server are executed at the server. Statements that refer to objects at yet another DB2 (which is possible only if the server is a DB2 subsystem) are executed at that DB2 rather than at the server.

Establishing a different server

The initial server of an application process is the local DB2 subsystem. A different server can be established by the explicit or implicit execution of a CONNECT statement.

The CURRENTSERVER bind option can affect which CONNECT rule is in effect. When an application process executes an SQL statement other than COMMIT, CONNECT TO, CONNECT RESET, SET CONNECTION, or ROLLBACK, a CONNECT (Type 1) statement is implicitly executed if both of the following rules apply:

- The CURRENTSERVER bind option was specified when the application plan was bound or rebound and the identified server is not the local DB2.
- An implicit or explicit CONNECT statement has not been executed by the application process.

For example, if CURRENTSERVER *x* was specified and the first SQL statement executed by the application process is an OPEN statement, a CONNECT TO *x* (Type 1) is executed before the OPEN statement is executed. If the implicit CONNECT fails, the application process is in the unconnected state. Regardless of whether or not the implied CONNECT is successful, the application process cannot execute a CONNECT (Type 2) statement because CONNECT (Type 1) rules are in effect.

In new distributed applications, use CONNECT (Type 2) and do not use the CURRENTSERVER bind option.

CONNECT (Type 1)

CONNECT (Type 1)

The CONNECT (Type 1) statement connects the application process to a designated server. This server is then the current server for the process. The CONNECT (Type 1) statement is used for DRDA access using the restricted level of function available in DB2 Version 2 Release 3. Differences between the two types of statements are described in “CONNECT (Type 1) and CONNECT (Type 2) differences” on page 501.

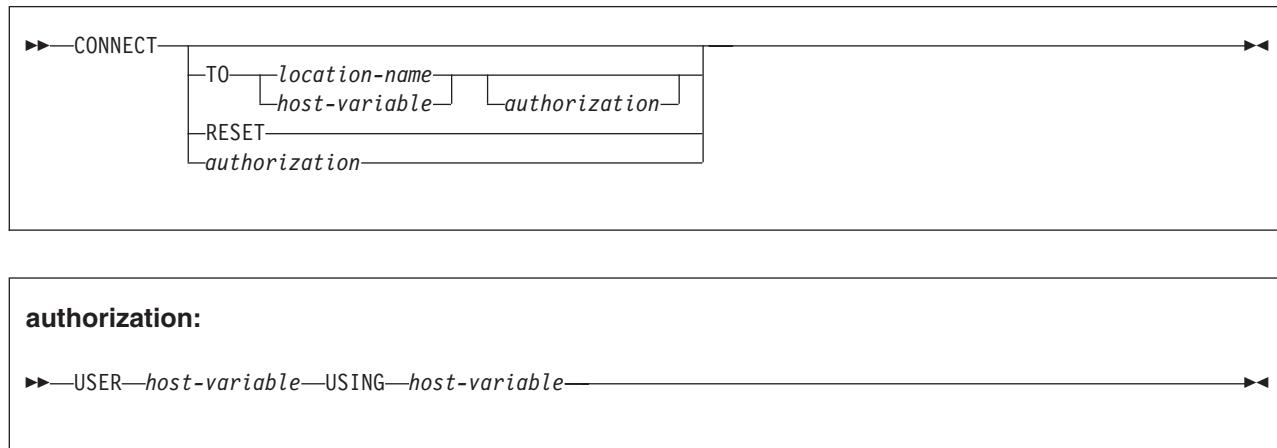
Invocation

This statement can only be embedded in an application program. It is an executable statement that cannot be dynamically prepared.

Authorization

The primary authorization ID of the process or the authorization ID specified on the CONNECT statement must be authorized to connect to the identified server or to the local DB2. The server or the local DB2 performs the authorization check and determines the specific authorization required. See Part 3 (Volume 1) of *DB2 Administration Guide* for further information.

Syntax



Description

TO *location-name* or *host-variable*

Identifies the server by the specified location name or the location name contained in the host variable. If a host variable is specified:

- It must be a character string variable with a length attribute that is not greater than 16. (A C NUL-terminated character string can be up to 17 bytes long.)
- It must not have a CLOB data type.
- It must not be followed by an indicator variable.
- The location name must be left-justified within the host variable and must conform to the rules for forming an ordinary location identifier.
- If the length of the location name is less than the length of the host variable, it must be padded on the right with blanks.

When the CONNECT statement is executed:

- The location name must identify a server known to the local DB2 subsystem. Hence, it must either be the location name of the local DB2 subsystem or it must appear in the LOCATION column of the SYSIBM.LOCATIONS table.
- The application process must be in a *connectable state*. (Connection states are explained in “Connection states” on page 507.)

RESET

CONNECT RESET is equivalent to CONNECT TO *x* where *x* is the location name of the local DB2 subsystem.

authorization

Specifies an authorization ID and a password that is used to verify that the authorization ID is authorized to connect to the server.

In general, the following statements are true:

- A CONNECT statement with the TO clause and the USER/USING clause can be executed only if there is no current or dormant connection to the named server. However, if the named server is the local DB2 subsystem and the CONNECT statement is the first SQL statement that is executed after the DB2 thread is created, the CONNECT statement executes successfully.
- A CONNECT statement without the TO clause but with the USER/USING clause can be executed only if no current or dormant connection to the local DB2 subsystem exist. However, if the CONNECT statement is the first SQL statement that is executed after the DB2 thread is created, the CONNECT statement executes successfully.

Authorization cannot be specified when the connection type is IMS or CICS. An attempt to do so causes an SQL error.

USER *host-variable*

Identifies the user ID trying to connect to the server. The host-variable may be up to 255 characters, must be a character string variable with a length attribute that is not greater than 255, must be left-justified, and must not include an indicator variable. In addition, if the length of the user ID is less than the length of the host variable, it must be padded on the right with blanks.

For a connection to the local DB2, a user ID of greater than 8 characters causes an SQL error.

USING *host-variable*

Identifies the password of the user ID trying to connect to the server. The host-variable may be up to 255 characters, must be a character string variable with a length attribute not greater than 255, must be left-justified, and must not include an indicator variable. In addition, if the length of the password is less than the length of the host variable, it must be padded on the right with blanks.

For a connection to the local DB2, a password of greater than 8 characters causes an SQL error.

CONNECT USER/USING is equivalent to CONNECT TO *x* USER/USING where *x* is the location name of the local DB2 subsystem (which has the semantic of CONNECT RESET).

If the server is the local DB2, then:

- DB2 invokes RACF via the RACROUTE macro with REQUEST=VERIFY to verify the password.

CONNECT (Type 1)

- If the password is verified, DB2 then invokes RACF again via the RACROUTE macro with REQUEST=AUTH, to check whether the authorization ID is allowed to use DB2 resources defined to RACF.
- DB2 then invokes the connection exit routine if one has been defined.
- The connection then has a primary authorization ID, possibly one or more secondary IDs, and an SQL ID.

If the server is not the local DB2, the following rules apply.

- The SECURITY_OUT column in SYSIBM.LUNAMES for SNA or the SECURITY_OUT column in SYSIBM.IPNAMES for TCP/IP must have the values 'A' (already verified) or 'P' (password). When the value is 'A', the user ID and password specified on the CONNECT is still sent.
- For SNA, the ENCRYPTPSWDS column in SYSIBM.SYSLUNAMES must be not contain 'Y'.
- The server must support at least DRDA Level 3.
- The authorization ID and password are verified at the server.
- In all cases, outbound translation as specified in SYSIBM.USERNAMES is not done.

CONNECT with no operand

This form of the CONNECT statement returns information about the current server. The information is returned in the SQLERRP field of the SQLCA as described above. This form of CONNECT:

- Does not require the application process to be in the connectable state
- Does not change the connection state
- Does not close cursors
- Returns blanks if the application process is in the unconnected state

Notes

Successful connection: If execution of the CONNECT statement is successful:

- The application process is connected to the identified server.
- The existing connections of the application process are ended. (The existing connections include the previous SQL connection, if any, and all DB2 private connections, if any.) When a connection is ended, all resources acquired by the application process through the connection and all resources used to create and maintain the connection are deallocated. Thus, all cursors are closed, all prepared statements are destroyed, and so on.
- The location name is placed in the CURRENT SERVER special register.
- Information about the server is placed in the SQLERRP field of the SQLCA. If the server is an IBM relational database product, the information has the form *pppvvrrm*, where:
 - *ppp* is:
 - ARI for DB2 Server for VSE & VM
 - DSN for DB2 for MVS
 - QSQ for OS/400®
 - SQL for all other DB2 products
 - *vv* is a two-digit version identifier such as '07'.
 - *rr* is a two-digit release identifier such as '01'.
 - *m* is a one-digit modification level such as '0'.

For example, if the server is Version 7 of DB2 for OS/390 and z/OS with the latest maintenance, the value of SQLERRP is 'DSN07011'.

Unsuccessful connection: If execution of the CONNECT statement is unsuccessful, the SQLERRP field of the SQLCA is set to the name of the DB2 requester module that detected the error.

If execution of the CONNECT statement is unsuccessful because the application process is not in the connectable state, the connection state of the application process is unchanged. If execution of the CONNECT statement is unsuccessful for any other reason, CURRENT SERVER is set to blanks and the application process is placed in the connectable and unconnected state.

Connection states: In the following description of the connection states, CONNECT means CONNECT TO, CONNECT RESET, or CONNECT authorization, not the form of CONNECT with no operand. At any time, an application process is in one of four states:

- Connectable and connected
- Unconnectable and connected
- Unconnectable and unconnected
- Connectable and unconnected

The following diagram shows the state transitions:

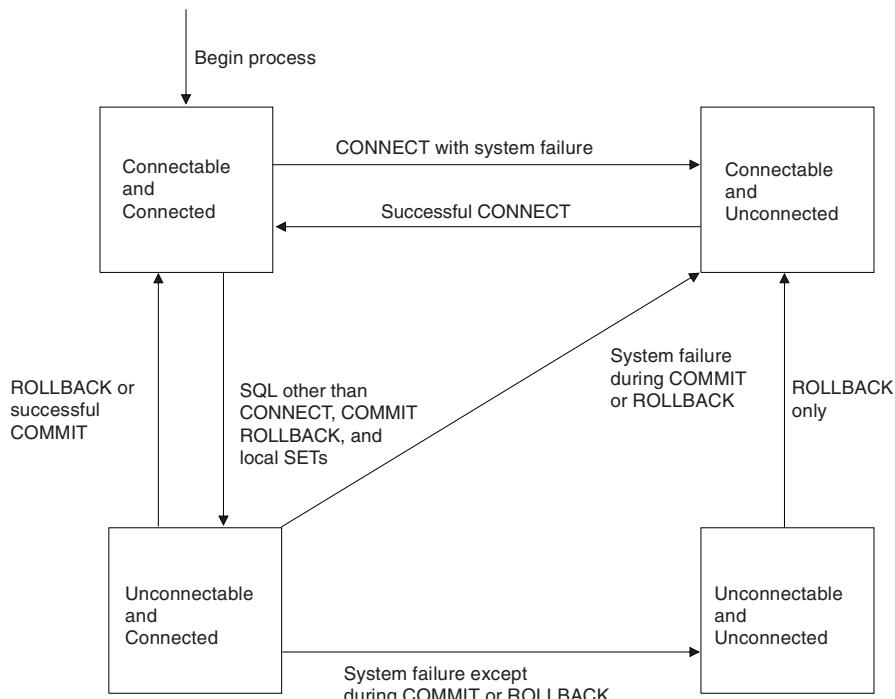


Figure 7. Connect state transitions

In the **connectable and connected state**, an application process is connected to a server and can execute CONNECT statements. This is the initial state. The process also enters this state when:

- It executes a rollback operation or successful commit from the unconnectable and connected state.
- It executes a successful CONNECT from the connectable and unconnected state.

CONNECT (Type 1)

In the **unconnectable and connected state**, an application process is connected to a server but cannot execute a CONNECT statement (SQLCODE -752). The process enters this state from the connectable and connected state when it executes any SQL statement other than CONNECT, COMMIT, ROLLBACK, or local SET (SET CURRENT PACKAGESET or SET *host-variable* = CURRENT PACKAGESET or CURRENT SERVER). A process cannot enter this state from the connectable and unconnected nor the unconnectable and unconnected states.

In the **unconnectable and unconnected state**, an application process is not connected to a server and cannot execute a CONNECT statement. The process enters this state from the unconnectable and connected state when the execution of an SQL statement other than COMMIT or ROLLBACK is unsuccessful because of a system failure that results in a rollback and deallocation of the conversation. The only SQL statement that can be successfully executed in this state is ROLLBACK. Any attempt to execute other SQL statements results in an error (SQLCODE -918).

In the **connectable and unconnected state**, an application process is not connected to a server. The process enters this state when:

- The execution of CONNECT is unsuccessful for any reason other than the application process not being in the connectable state.
- A system failure occurs during the execution of a COMMIT or ROLLBACK statement from the unconnectable and connected state.
- A ROLLBACK statement is executed from the unconnectable and unconnected state.

The only SQL statements that can be successfully executed in this state are CONNECT, COMMIT, ROLLBACK, and local SET statements. Any attempt to execute other SQL statements results in an error (SQLCODE -900). SET *host-variable* = CURRENT SERVER will set the host variable to blanks.

Additional rules: It is not an error to execute consecutive CONNECT statements because CONNECT itself does not remove the application process from the connectable state. It is an error to execute any SQL statement other than CONNECT, COMMIT, ROLLBACK, or local SET, and then execute CONNECT. To avoid the error, execute a commit or rollback operation before executing the CONNECT.

A CONNECT to the current server is treated like any other CONNECT. Such a CONNECT can cause the closing of cursors and the redundant deallocation and allocation of a conversation.

It may be the case that the SQL CONNECT statement returns, and indicates a successful execution when no physical connection yet exists. DB2 delays the physical connection process, when possible, to economize on the number of messages sent. Therefore, errors in CONNECT statement processing may be reported following the next executable SQL statement, not immediately following the CONNECT statement.

When CONNECT is used to connect back to the local DB2, the CURRENT SQLID special register is not reinitialized.

SET CONNECTION and RELEASE do not change the state of the application process from connectable to unconnectable.

The SQLRULES bind option has no effect on CONNECT (Type 1) statements.

If the CONNECT statement contains host variables, the contents of the host variables are assumed to be in the encoding scheme that was specified in the ENCODING parameter when the package or plan that contains the statement was bound.

Examples

Example 1: Connect the application to a DBMS whose location identifier is in the character-string variable LOCNAME.

```
EXEC SQL CONNECT TO :LOCNAME;
```

Example 2: Use the CONNECT statement to obtain information about the current server. The information is then stored in the SSQLERRP field of the SQLCA.

```
EXEC SQL CONNECT;
```

Example 3: An application has connected to a DB2 server that is not the local DBMS. During the connection, the application has opened a cursor and fetched rows from the cursor's result table. To connect to the local DBMS, the application executes the following statements:

```
EXEC SQL COMMIT WORK;  
EXEC SQL CONNECT RESET;
```

The commit operation is required because the OPEN statement for the cursor has caused the application to enter the unconnectable and connected state. If the cursor had been declared with WITH HOLD and had not been closed with a CLOSE statement, it would still be open after the execution of the COMMIT, but would be closed with the execution of the CONNECT.

Example 4: Connect the application to a DBMS whose location identifier is in the character-string variable LOC using the authorization identifier in the character-string variable AUTHID and the password in the character-string variable PASSWORD.

```
EXEC SQL CONNECT TO :LOC USER :AUTHID USING :PASSWORD;
```

CONNECT (Type 2)

CONNECT (Type 2)

The CONNECT (Type 2) statement connects the application process to a designated server. This server is then the current server for the process. Differences between the two types of statements are described in “CONNECT (Type 1) and CONNECT (Type 2) differences” on page 501. Refer to “Connection management for DRDA access and DB2 private protocol” on page 17 for more information about connection states.

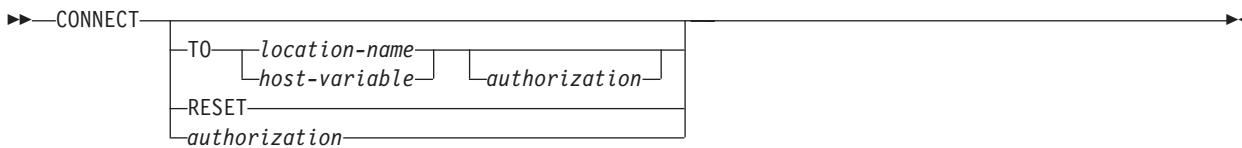
Invocation

This statement can only be embedded in an application program. It is an executable statement that cannot be dynamically prepared.

Authorization

The primary authorization ID of the process or the authorization ID specified on the CONNECT statement must be authorized to connect to the identified server or to the local DB2. The authorization check is performed by the database server when the statement is executed, and the specific authorization required is determined by that server. See Part 3 (Volume 1) of *DB2 Administration Guide* for further information.

Syntax



authorization:

```
►--USER--host-variable--USING--host-variable-->
```

Description

TO *location-name* or *host-variable*

Identifies the server by the specified location name or the location name contained in the host variable. If a host variable is specified:

- It must be a character string variable with a length attribute that is not greater than 16. (A C NUL-terminated character string can be up to 17 bytes long.)
- It must not have a CLOB data type.
- It must not be followed by an indicator variable.
- The location name must be left-justified within the host variable and must conform to the rules for forming an ordinary location identifier.
- If the length of the location name is less than the length of the host variable, it must be padded on the right with blanks.

Let S denote the specified location name or the location name contained in the host variable.

S must not identify a DB2 private connection of the application process. If the SQLRULES(STD) bind option is in effect, S must not identify an existing SQL connection of the application process.

S must identify a server known to the local DB2 subsystem. Hence, S must be the location name of the local DB2 subsystem or it must appear in the LOCATION column of the SYSIBM.LOCATIONS table.

RESET

CONNECT RESET is equivalent to CONNECT TO *x* where *x* is the location name of the local DB2 subsystem.

- If the SQLRULES(DB2) bind option is in effect, CONNECT RESET establishes the local DB2 subsystem as the current SQL connection
- If the SQLRULES(STD) bind option is in effect, CONNECT RESET establishes the local DB2 subsystem as the current SQL connection only if the connection does not exist.

authorization

Specifies an authorization ID and a password that is used to verify that the authorization ID is authorized to connect to the server.

In general, the following statements are true:

- A CONNECT statement with the TO clause and the USER/USING clause can be executed only if there is no current or dormant connection to the named server. However, if the named server is the local DB2 subsystem and the CONNECT statement is the first SQL statement that is executed after the DB2 thread is created, the CONNECT statement executes successfully
- A CONNECT statement without the TO clause but with the USER/USING clause can be executed only if no current or dormant connection to the local DB2 subsystem exist. However, if the CONNECT statement is the first SQL statement that is executed after the DB2 thread is created, the CONNECT statement executes successfully.

Authorization cannot be specified when the connection type is IMS or CICS. An attempt to do so causes an SQL error.

USER *host-variable*

Identifies the user ID trying to connect to the server. The host-variable may be up to 255 characters, must be a character string variable with a length attribute that is not greater than 255, must be left-justified, and must not include an indicator variable. In addition, if the length of the user ID is less than the length of the host variable, it must be padded on the right with blanks.

For a connection to the local DB2, a user ID of greater than 8 characters causes an SQL error.

USING *host-variable*

Identifies the password of the user ID trying to connect to the server. The host-variable may be up to 255 characters, must be a character string variable with a length attribute not greater than 255, must be left-justified, and must not include an indicator variable. In addition, if the length of the password is less than the length of the host variable, it must be padded on the right with blanks.

For a connection to the local DB2, a password of greater than 8 characters causes an SQL error.

CONNECT (Type 2)

CONNECT USER/USING is equivalent to CONNECT TO x USER/USING where x is the location name of the local DB2 subsystem (which has the semantic of CONNECT RESET).

If the server is the local DB2, then:

- DB2 invokes RACF via the RACROUTE macro with REQUEST=VERIFY to verify the password.
- If the password is verified, DB2 then invokes RACF again via the RACROUTE macro with REQUEST=AUTH, to check whether the authorization ID is allowed to use DB2 resources defined to RACF.
- DB2 then invokes the connection exit routine if one has been defined.
- The connection then has a primary authorization ID, possibly one or more secondary IDs, and an SQL ID.

If the server is not the local DB2, the following rules apply.

- The SECURITY_OUT column in SYSIBM.LUNAMES for SNA or the SECURITY_OUT column in SYSIBM.IPNAMES for TCP/IP must have the values 'A' (already verified) or 'P' (password). When the value is 'A', the user ID and password specified on the CONNECT is still sent.
- For SNA, the ENCRYPTIONSWDS column in SYSIBM.SYSLUNAMES must be not contain 'Y'.
- The server must support at least DRDA Level 3.
- The authorization ID and password are verified at the server.
- In all cases, outbound translation as specified in SYSIBM.USERNAMES is not done.

If the authorization check fails, the connection is placed in the connectable and unconnected state.

CONNECT with no operand

This form of the CONNECT statement returns information about the current server and has no effect on connection states. The information is returned in the SQLERRP field of the SQLCA as described above. SQLERRP is set to blanks if the application process is in the unconnected state.

Notes

When CONNECT is used to connect back to the local DB2, the CURRENT SQLID special register is not reinitialized unless USERID/USING is specified.

If the CONNECT statement is successful:

- S becomes the current connection of the application process in one of the following ways:
 - If S is not an existing SQL connection of the application process, an SQL connection to server S is created and placed in the current and held states. The previously current SQL connection, if any, is placed in the dormant state.
 - If S is a dormant SQL connection of the application process and the SQLRULES(DB2) option is in effect, S is placed in the current state. The previously current SQL connection, if any, is placed in the dormant state.
 - If S is the current SQL connection of the application process and the SQLRULES(DB2) option is in effect, the states of S and all other connections of the application process are unchanged.
- The location name is placed in the CURRENT SERVER special register.

- Information about server S is placed in the SQLERRP field of the SQLCA. If the server is an IBM relational database product, the information has the form *pppvvrrm*, where:

- *ppp* is:
 - ARI for DB2 Server for VSE & VM
 - DSN for DB2 for OS/390 and z/OS
 - QSQ for OS/400
 - SQL for all other DB2 products
- *vv* is a two-digit version identifier such as '07'.
- *rr* is a two-digit release identifier such as '01'.
- *m* is a one-digit modification level such as '0'.

For example, if the server is Version 7 of DB2 for OS/390 and z/OS with the latest maintenance, the value of SQLERRP is 'DSN07011'.

If the CONNECT statement is unsuccessful, the connection state of the application process and the states of its SQL connections are unchanged unless the failure was because an authorization check failed. If this is the case, the connection is placed in the connectable and unconnected state.

If the CONNECT statement contains host variables, the contents of the host variables are assumed to be in the encoding scheme that was specified in the ENCODING parameter when the package or plan that contains the statement was bound.

Examples

Example 1: Execute SQL statements at TOROLAB1 and TOROLAB2. The first CONNECT statement creates the TOROLAB1 connection. The second CONNECT statement creates the TOROLAB2 connection and places the TOROLAB1 connection in the dormant state.

```
EXEC SQL CONNECT TO TOROLAB1;
          (execute statements referencing objects at TOROLAB1)
EXEC SQL CONNECT TO TOROLAB2;
          (execute statements referencing objects at TOROLAB2)
```

Example 2: Connect the application to a DBMS whose location identifier is in the character-string variable LOC using the authorization identifier in the character-string variable AUTHID and the password in the character-string variable PASSWORD. Perform work for the user, and then release the connection and connect again using a different user ID and password.

```
# EXEC SQL CONNECT TO :LOC USER :AUTHID USING :PASSWORD;
#           (execute SQL statements accessing data on the server)
# RELEASE :LOC;
# EXEC SQL COMMIT;
#           (set AUTHID and PASSWORD to new values)
# EXEC SQL CONNECT TO :LOC USER :AUTHID USING :PASSWORD;
#           (execute SQL statements accessing data on the server)
```

#

CREATE ALIAS

CREATE ALIAS

The CREATE ALIAS statement defines an alias for a table or view. The definition is recorded in the DB2 catalog at the current server. The table or view does not have to be described in that catalog.

Invocation

This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is implicitly or explicitly specified.

Authorization

The privilege set that is defined below must include at least one of the following:

- The CREATEALIAS privilege
- SYSADM or SYSCtrl authority
- DBADM or DBCTRL authority on the database that contains the table, if the alias is for a table and the value of field DBADM CREATE AUTH on installation panel DSNTIPP is YES

Privilege set: If the statement is embedded in an application program, the privilege set is the privileges that held by the authorization ID of the owner of the plan or package. If the specified alias name includes a qualifier that is not the same as this authorization ID, the privilege set must include one of the following authorities:

- SYSADM or SYSCtrl authority
- DBADM or DBCTRL authority on the database that contains the table, if the alias is for a table and the value of field DBADM CREATE AUTH on installation panel DSNTIPP is YES

If the statement is dynamically prepared, the privilege set is the privileges that are held by the SQL authorization ID of the process. If the specified alias name includes a qualifier that is not the same as this authorization ID:

- The privilege set must include SYSADM or SYSCtrl authority.
- The privilege set includes DBADM or DBCTRL authority on the database, if the alias is for a table and the value of field DBADM CREATE AUTH on installation panel DSNTIPP is YES.
- The qualifier must be the same as one of the authorization IDs of the process and the privileges that are held by that authorization ID must include the CREATEALIAS privilege. This is an exception to the rule that the privilege set is the privileges that are held by the SQL authorization ID of the process.

Syntax

```
►►CREATE ALIAS—alias-name—FOR—table-name—view-name—►►
```

Description

alias-name

Names the alias. The name must not identify a table, view, alias, or synonym that exists at the current server.

If qualified, the name can be a two-part or three-part name. If a three-part name is used, the first part must match the value of the field DB2 LOCATION NAME on installation panel DSNTIPR at the current server. (If the current server is not the local DB2, this name is not necessarily the name in the CURRENT SERVER special register.) Whether the name is two-part or three-part, the authorization ID that qualifies the name is the owner of the alias.

If the alias name is unqualified and the statement is embedded in an application program, the owner of the alias is the authorization ID that serves as the implicit qualifier for unqualified object names. This is the authorization ID in the QUALIFIER operand when the plan or package was created or last rebound. If QUALIFIER was not used, the owner of the alias is the owner of the package or plan.

If the alias name is unqualified and the statement is dynamically prepared, the SQL authorization ID is the owner of the alias.

The owner has the privilege to drop the alias.

FOR *table-name or view-name*

Identifies the table or view for which the alias is defined. If a table is identified, it must not be an auxiliary table or a declared temporary table. The table or view need not exist at the time the alias is defined. If it does exist, it can be at the current server or at another server. The name must not be the same as the alias name and must not identify an alias that exists at the current server.

Notes

An alias can be defined for a table, view, or alias that is not at the current server. When so defined, the existence of the referenced object is not verified at the time the alias is created. But the object must exist when a statement that contains the alias is executed. And if that object is also an alias, it must refer to a table or view at the server where that alias is defined.

A warning occurs if an alias is defined for a table or view that is local to the current server but does not exist.

|
| When a table is moved from one location to another, the alias for that table must be
| dropped and then re-created with the new location name. When an application is
| moved from one location to another location, aliases must exist at the new location
| for all tables that are referred to by the application. Aliases at the old location can
| be dropped if they are no longer needed.

When an application uses three-part name aliases for remote objects and DRDA
access, the application program must be bound at each location that is specified in
the three-part names. Also, you need to define the alias at the remote site as well
as at the local site.

Example

Create an alias for a catalog table at a DB2 with location name DB2USCALABOA5281.

```
CREATE ALIAS LATABLES FOR DB2USCALABOA5281.SYSIBM.SYSTABLES;
```

CREATE AUXILIARY TABLE

The CREATE AUXILIARY TABLE statement creates an auxiliary table at the current server for storing LOB data.

Invocation

Do not use this statement if the value of special register CURRENT RULES is 'STD' when the statement is executed. When the register's value is 'STD' and a base table is created with LOB columns or altered such that LOB columns are added, DB2 automatically creates the LOB table space, auxiliary table, and index on the auxiliary table for each LOB column. DB2 chooses the names and characteristics of these objects. For more information about the names and the characteristics, see "Creating a table with LOB columns" on page 675.

This statement can be embedded in an application program or issued interactively if the value of special register CURRENT RULES is 'DB2' when the statement is executed. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is implicitly or explicitly specified.

Authorization

The privilege set that is defined below must include at least one of the following:

- The CREATETAB privilege for the database implicitly or explicitly specified by the IN clause
- DBADM, DBCTRL, or DBMAINT authority for the database
- SYSADM or SYSCTRL authority

Privilege set: If the statement is embedded in an application program, the privilege set is the privileges that are held by the authorization ID of the owner of the plan or package. If the specified table name includes a qualifier that is not the same as this authorization ID, the privilege set must include SYSADM or SYSCTRL authority, DBADM authority for the database, or DBCTRL authority for the database.

If the statement is dynamically prepared, the privilege set is the privileges that are held by the SQL authorization ID of the process. However, if the specified table name includes a qualifier that is not the same as this authorization ID, the following rules apply:

1. If the privilege set includes SYSADM or SYSCTRL authority, DBADM authority for the database, or DBCTRL authority for the database, any qualifier is valid.
2. If the privilege set does not include any of the authorities listed in item 1 above, the qualifier is valid only if it is the same as one of the authorization IDs of the process and the privilege set that are held by that authorization ID includes all²⁸ privileges needed to create the table.

28. Exception: The CREATETAB privilege is checked on the SQL authorization ID of the process.

Syntax

```

    ➤ CREATE [AUXILIARY | AUX] TABLE aux-table-name IN [database-name.] table-space-name
    ➤ STORES table-name COLUMN column-name PART integer
  
```

Description

AUXILIARY or AUX

Specifies a table that is used to store the LOB data for a LOB column (or a column with a distinct type that is based on a LOB data type).

aux-table-name

Names the auxiliary table. The name must not identify a table, view, alias, or synonym that exists at the current server.

If qualified, the name can be a two-part or three-part name. If a three-part name is used, the first part must match the value of field DB2 LOCATION NAME on installation panel DSNTIPR at the current server. (If the current server is not the local DB2, this name is not necessarily the name in the CURRENT SERVER special register.) Whether the name is two-part or three-part, the authorization ID that qualifies the name is the table's owner.

If the table name is unqualified and the statement is embedded in a program, the owner of the table is the authorization ID that serves as the implicit qualifier for unqualified object names. This is the authorization ID in the QUALIFIER operand when the plan or package was created or last rebound. If QUALIFIER was not used, the owner of the table is the owner of the package or plan.

If the table name is unqualified and the statement is dynamically prepared, the SQL authorization ID is the owner of the table.

IN *database-name.table-space-name* or IN *table-space-name*

Identifies the table space in which the auxiliary table is created. The name must identify an empty LOB table space that currently exists at the current server. The LOB table space must be in the same database as its associated base table.

If you specify a database and a table space, the table space must belong to the specified database. If you specify only a table space, it must belong to database DSNDB04.

STORES *table-name* COLUMN *column-name*

Identifies the base table and the column of that table that is to be stored in the auxiliary table. If the base table is nonpartitioned, an auxiliary table must not already exist for the specified column. If the base table is partitioned, an auxiliary table must not already exist for the specified column and specified partition.

The encoding scheme for the LOB data stored in the auxiliary table is the same as the encoding scheme for the base table. It is either ASCII, EBCDIC, or UNICODE depending on the value of the CCSID clause when the base table was created.

CREATE AUXILIARY TABLE

The auxiliary table can store a BLOB, CLOB, or DBCLOB value that is greater than 1 gigabyte in length only if the LOB table space for the auxiliary table was defined with LOG NO.

PART *integer*

Specifies the partition of the base table for which the auxiliary table is to store the specified column. You can specify PART only if the base table is defined in a partitioned table space, and no other auxiliary table exists for the same LOB column of the base table.

Notes

Determining the number of auxiliary tables to create: The number of auxiliary tables to create depends on the number of LOB columns in the base table and whether the base table is partitioned. If the base table is nonpartitioned, you need one LOB table space and one auxiliary table for each LOB column in the base table. If the base table is partitioned, you need one LOB table space and one auxiliary table for each partition for each LOB column. For example if the base table has four partitions and two LOB columns, you need to create a total of eight auxiliary tables in eight different LOB table spaces.

```
#  
#  
#  
#  
#  
#  
#  
# Creating auxiliary tables for LOB columns: If your base table is nonpartitioned,  
# you must create one LOB table space and one auxiliary table for each LOB column.  
# If your base table is partitioned, for each LOB column, you must create one LOB  
# table space and one auxiliary table for each partition. For example, if your base  
# table has three partitions, create three LOB table spaces and three auxiliary tables  
# for each LOB column.
```

Example

Assume that a column named EMP_PHOTO with a data type of BLOB(110K) has been added to sample employee table DSN8710.EMP for each employee's photo. Create auxiliary table EMP_PHOTO_ATAB to store the BLOB data for the BLOB column in LOB table space DSN8D71A.PHOTOLTS.

```
CREATE AUX TABLE EMP_PHOTO_ATAB  
  IN DSN8D71A.PHOTOLTS  
  STORES DSN8710.EMP  
  COLUMN EMP_PHOTO;
```

CREATE DATABASE

The CREATE DATABASE statement defines a DB2 database at the current server.

Invocation

This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is implicitly or explicitly specified.

Authorization

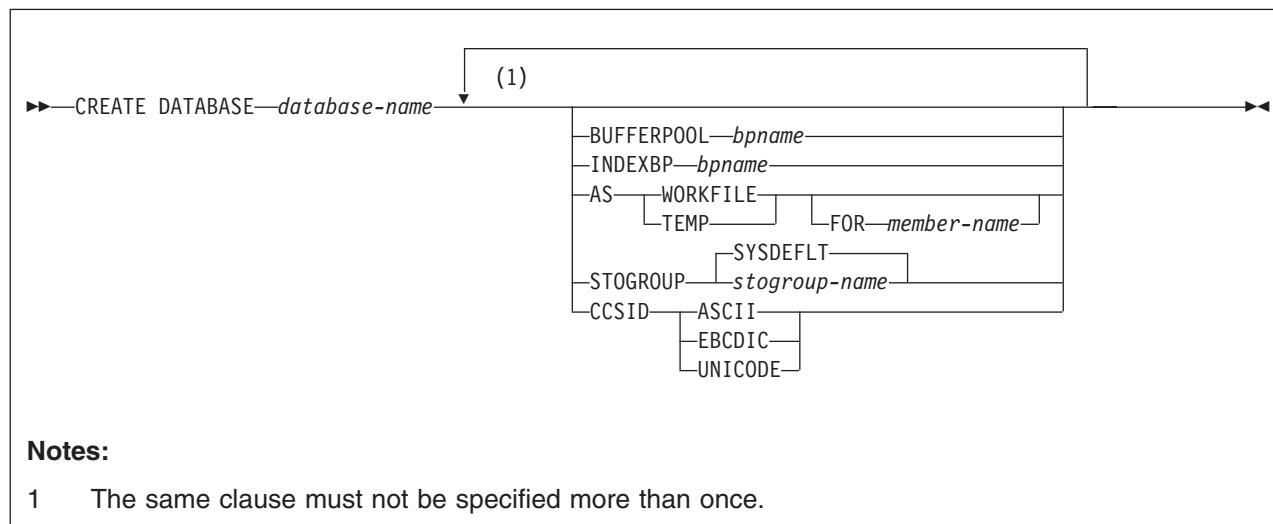
The privilege set that is defined below must include at least one of the following:

- The CREATEDBA privilege
- The CREATEDBC privilege
- SYSADM or SYSCTRL authority

Privilege set: If the statement is embedded in an application program, the privilege set is the privileges that are held by the authorization ID of the owner of the plan or package. If the statement is dynamically prepared, the privilege set is the privileges that are held by the SQL authorization ID of the process.

See “Notes” on page 521 for the authorization effect of a successful CREATE DATABASE statement.

Syntax



Description

database-name

Names the database. The name must not start with DSNDB and must not identify a database that exists at the current server. If the database is to be a work file database in a data sharing environment, DSNDB07 is an acceptable work file database name. However, only one member of a data sharing group can use DSNDB07 as the name of its work file database.

BUFFERPOOL *bpname*

Specifies the default buffer pool name to be used for table spaces created

CREATE DATABASE

within the database. If the database is a work file database, 8KB and 16KB buffer pools cannot be specified. See “Naming conventions” on page 34 for more details about *bpname*.

If you omit the BUFFERPOOL clause, the buffer pool specified for user data on installation panel DSNTIP1 is used. The default value for the user data field on that panel is BP0.

INDEXBP *bpname*

Specifies the default buffer pool name to be used for the indexes created within the database. The name must identify a 4KB buffer pool. See “Naming conventions” on page 34 for more details about *bpname*. If the database is a work file database, INDEXBP cannot be specified.

If you omit the INDEXBP clause, the buffer pool specified for user indexes on installation panel DSNTIP1 is used. The default value for the user indexes field on that panel is BP0.

AS WORKFILE or **AS TEMP**

Indicates that this is a work file database or a database for declared temporary tables (a TEMP database).

AS WORKFILE

Specifies the database is a work file database. AS WORKFILE can be specified only in a data sharing environment. Only one work file database can be created for each DB2 member.

AS TEMP

Specifies the database is for declared temporary tables only. AS TEMP must be specified to create a database that will be used for declared temporary tables; otherwise, the database will not be used for declared temporary tables. Only one TEMP database can be created for each DB2 subsystem or data sharing member. A TEMP database cannot be shared between DB2 subsystems or data sharing members.

PUBLIC implicitly receives the CREATETAB privilege (without GRANT authority) to define a declared temporary table in the TEMP database. This implicit privilege is not recorded in the DB2 catalog and cannot be revoked.

FOR *member-name*

Specifies the member for which this database is to be created. Specify FOR *member-name* only in a data sharing environment.

If FOR *member-name* is not specified, the member is the DB2 subsystem on which the CREATE DATABASE statement is executed.

The CCSID clause is not supported for a work file database or a TEMP database. A TEMP database can contain a mixture of encoding schemes. If you specify AS WORKFILE or AS TEMP, do not use the CCSID clause.

STOGROUP *stogroup-name*

Specifies the storage group to be used, as required, as a default storage group to support DASD space requirements for table spaces and indexes within the database. The default is SYSDEFLT.

CCSID *encoding-scheme*

Specifies the default encoding scheme for data stored in the database. The default applies to table spaces created in the database. All tables stored within a table space must use the same encoding scheme.

ASCII Specifies that the data must be encoded using the ASCII CCSIDs of the server.

EBCDIC

Specifies that the data must be encoded using the EBCDIC CCSIDs of the server.

UNICODE

Specifies that the data must be encoded using the UNICODE CCSIDs of the server.

Usually, each encoding scheme requires only a single CCSID. Additional CCSIDs are needed when mixed, graphic, or UNICODE data is used.

The option defaults to the value of field DEF ENCODING SCHEME on installation panel DSNTIPF.

Do not use the CCSID clause if you specify the AS WORKFILE or AS TEMP clause.

Notes

If the statement is embedded in an application program, the owner of the plan or package is the owner of the database. If the statement is dynamically prepared, the SQL authorization ID of the process is the owner of the database.

If the owner of the database has the CREATEDBA, SYSADM, or SYSCTRL authority, the owner acquires DBADM authority for the database. DBADM authority for a database includes table privileges on all tables in that database. Thus, if a user with SYSCTRL authority creates a database, that user has table privileges on all tables in that database. This is an exception to the rule that SYSCTRL authority does not include table privileges.

If the owner of the database has the CREATEDBC privilege, but not the CREATEDBA privilege, the owner acquires DBCTRL authority for the database. In this case, no authorization ID has DBADM authority for the database until it is granted by an authorization ID with SYSADM authority.

Examples

Example 1: Create database DSN8D71P. Specify DSN8G710 as the default storage group to be used for the table spaces and indexes in the database. Specify 8KB buffer pool BP8K1 as the default buffer pool to be used for table spaces in the database, and BP2 as the default buffer pool to be used for indexes in the database.

```
CREATE DATABASE DSN8D71P
  STOGROUP DSN8G710
  BUFFERPOOL BP8K1
  INDEXBP BP2;
```

Example 2: Create database DSN8TEMP. Use the defaults for the default storage group and default buffer pool names. Specify ASCII as the default encoding scheme for data stored in the database.

```
CREATE DATABASE DSN8TEMP
  CCSID ASCII;
```

CREATE DISTINCT TYPE

CREATE DISTINCT TYPE

The CREATE DISTINCT TYPE statement defines a distinct type, which is a data type that a user defines. A distinct type must be sourced on one of the built-in data types. Successful execution of the statement also generates:

- A function to cast between the distinct type and its source type
- A function to cast between the source type and its distinct type
- As appropriate, support for the use of comparison operators with the distinct type

Invocation

This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is implicitly or explicitly specified.

Authorization

The privilege set that is defined below must include at least one of the following:

- The CREATEIN privilege for the schema or all schemas
- SYSADM or SYSCTRL authority

The authorization ID that matches the schema name implicitly has the CREATEIN privilege on the schema.

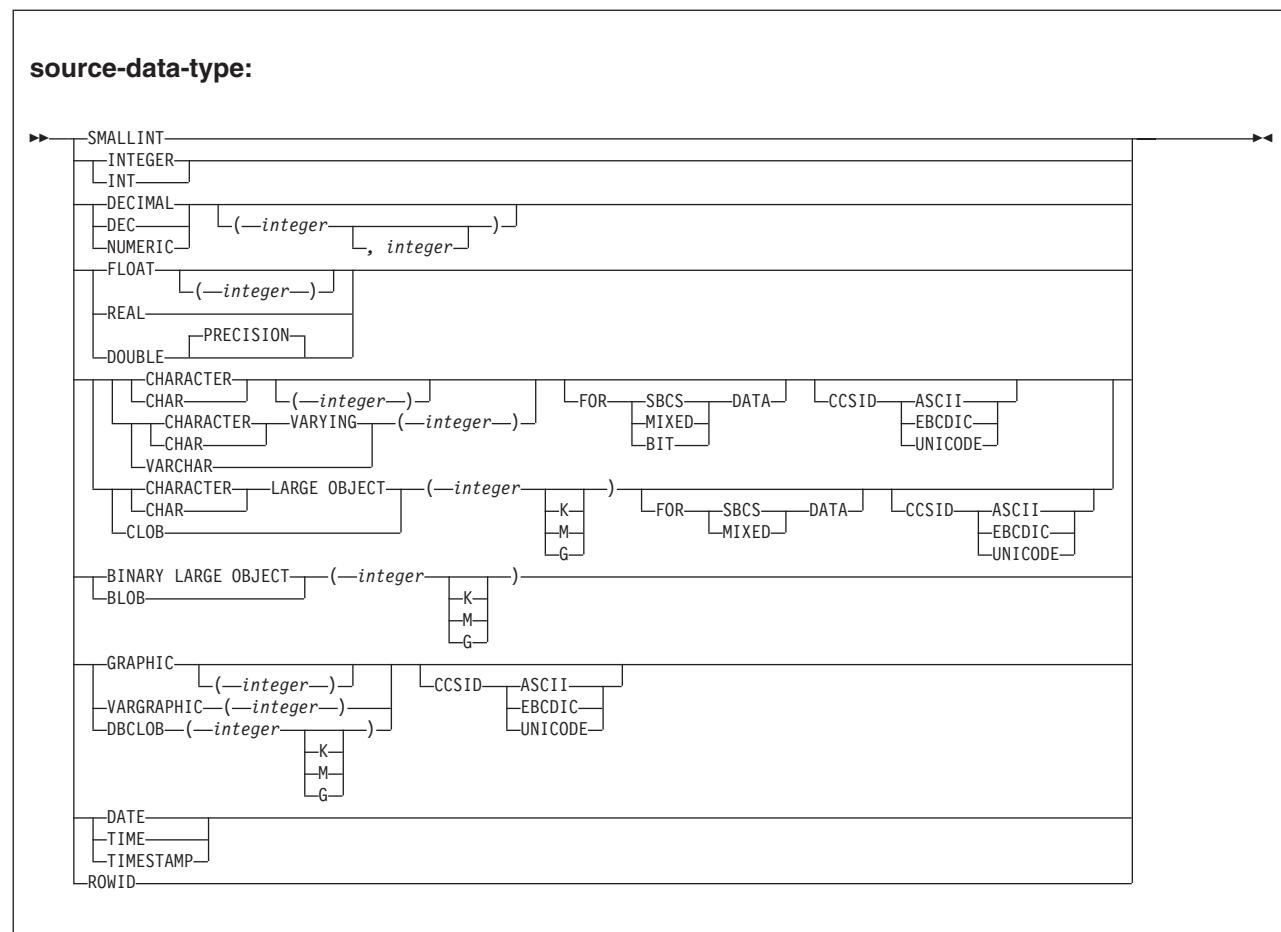
Privilege set: If the statement is embedded in an application program, the privilege set is the privileges that are held by the authorization ID of the owner of the plan or package.

If the statement is dynamically prepared, the privilege set is the privileges that are held by the SQL authorization ID of the process. The specified distinct type name can include a schema name (a qualifier). However, if the schema name is not the same as the SQL authorization ID, one of the following conditions must be met:

- The privilege set includes SYSADM or SYSCTRL authority.
- The SQL authorization ID of the process has the CREATEIN privilege on the schema.

Syntax

```
►►CREATE DISTINCT TYPE—distinct-type-name—AS—source-data-type—►►
```



Description

distinct-type-name

Names the distinct type. The name is implicitly or explicitly qualified by a schema name. The name, together with the implicit or explicit schema name, must not identify a distinct type that exists at the current server.

- The unqualified form of *distinct-type-name* is a long SQL identifier. *distinct-type-name* must not be the name of a built-in data type, BOOLEAN, or any of following system-reserved keywords even if you specify them as delimited identifiers:

ALL	LIKE	UNIQUE
AND	MATCH	UNKNOWN
ANY	NOT	=
BETWEEN	NULL	~=
DISTINCT	ONLY	<
EXCEPT	OR	≤
EXISTS	OVERLAPS	~<
FALSE	SIMILAR	>
FOR	SOME	≥
FROM	TABLE	→
IN	TRUE	↔
IS	TYPE	

The unqualified name is implicitly qualified with a schema name according to the following rules:

CREATE DISTINCT TYPE

If the CREATE DISTINCT TYPE statement is embedded in a program, the schema name is the authorization ID in the QUALIFIER bind option when the plan or package was created or last rebound. If QUALIFIER was not specified, the schema name is the owner of the plan or package.

If the CREATE DISTINCT TYPE statement is dynamically prepared, the schema name is the SQL authorization ID in the CURRENT SQLID special register.

- The qualified form of *distinct-type-name* is a short SQL identifier (the schema name) followed by a period and a long SQL identifier.

A schema name must not begin with 'SYS' unless the schema name is 'SYSADM'.

The owner of the distinct type is determined by how the CREATE DISTINCT TYPE statement is invoked:

- If the statement is embedded in a program, the owner is the authorization ID of the owner of the plan or package.
- If the statement is dynamically prepared, the owner is the SQL authorization ID in the CURRENT SQLID special register.

Although the information is not recorded in the catalog, the owner is given the USAGE privilege on the distinct type. The owner is also given the EXECUTE privilege with the GRANT option on each of the generated cast functions.

source-data-type

Specifies the data type that is used as the basis for the internal representation of the distinct type. The data type must be a built-in data type. You can use any of the built-in data types that are allowed for the CREATE TABLE statement except LONG VARCHAR or LONG VARGRAPHIC. Use VARCHAR or VARGRAPHIC with an explicit length instead.

If you do not specify a specific value for the data types that have length, precision, or scale attributes (CHAR, GRAPHIC, DECIMAL, NUMERIC, FLOAT), the defaults are as follows:

CHAR	CHAR(1)
GRAPHIC	GRAPHIC(1)
DECIMAL	DECIMAL(5,0)
FLOAT	DOUBLE (length of 8)

For more information on built-in data types, see "built-in-data-type" on page 658.

If the distinct type is sourced on a character string data type, the FOR clause indicates the subtype. If you do not specify the FOR clause, the distinct type is defined with the default subtype. For ASCII or EBCDIC data, the default is SBCS when the value of field MIXED DATA on installation panel DSNTIPF is NO. The default is MIXED when the value is YES. For UNICODE character data, the default subtype is mixed.

If the distinct type is sourced on a string data type, the CCSID clause indicates whether the encoding scheme of the data is ASCII, EBCDIC or UNICODE. If you do not specify CCSID ASCII, CCSID EBCDIC, or UNICODE, the encoding scheme is the value of field DEF ENCODING SCHEME on installation panel DSNTIPF.

Notes

Source data types with DBCS or mixed data: When the implicit or explicit encoding scheme is ASCII or EBCDIC and the source data type is graphic or a character type is MIXED DATA, then the value of field FOR MIXED DATA on installation panel DSNTIPF must be YES; otherwise, an error occurs.

Generated cast functions: The successful execution of the CREATE DISTINCT TYPE statement causes DB2 to generate the following cast functions:

- A function to convert from the distinct type to its source data type
- A function to convert from the source data type to the distinct type
- A function to cast from a data type *A* to distinct type *DT*, where *A* is promotable to the source data type *S* of distinct type *DT*

For some source data types, DB2 supports an additional function to convert from:

- INTEGER to the distinct type if the source type is SMALLINT
- VARCHAR to the distinct type if the source type is CHAR
- VARGRAPHIC to the distinct type if the source type is GRAPHIC
- DOUBLE to the distinct type if the source type is REAL

The cast functions are created as if the following statements were executed:

```
CREATE FUNCTION source-type-name (distinct-type-name)
    RETURNS source-type-name ...

CREATE FUNCTION distinct-type-name (source-type-name)
    RETURNS distinct-type-name ...
```

Even if you specified a length, precision, or scale for the source data type in the CREATE DISTINCT TYPE statement, the name of the cast function that converts from the distinct type to the source type is simply the name of the source data type. The data type of the value that the cast function returns includes any length, precision, or scale values that you specified for the source data type. (See Table 44 on page 526 for details.)

The name of the cast function that converts from the source type to the distinct type is the name of the distinct type. The input parameter of the cast function has the same data type as the source data type, including the length, precision, and scale.

For example, assume that a distinct type named T_SHOESIZE is created with the following statement:

```
CREATE DISTINCT TYPE CLAIRE.T_SHOESIZE AS VARCHAR(2)
```

When the statement is executed, DB2 also generates the following cast functions. VARCHAR converts from the distinct type to the source type, and T_SHOESIZE converts from the source type to the distinct type.

```
FUNCTION CLAIRE.VARCHAR (CLAIRE.T_SHOESIZE) RETURNS SYSIBM.VARCHAR (2)
FUNCTION CLAIRE.T_SHOESIZE (SYSIBM.VARCHAR (2)) RETURNS CLAIRE.T_SHOESIZE
```

Notice that function VARCHAR returns a value with a data type of VARCHAR(2) and that function T_SHOESIZE has an input parameter with a data type of VARCHAR(2).

The schema of the generated cast functions is the same as the schema of the distinct type. No other function with the same name and function signature must already exist in the database.

In the preceding example, if T_SHOESIZE had been sourced on a SMALLINT, CHAR, or GRAPHIC data type instead of a VARCHAR data type, another cast

CREATE DISTINCT TYPE

function would have been generated in addition to the two functions to cast between the distinct type and the source data type. For example, assume that T_SHOESIZE is created with this statement:

```
CREATE DISTINCT TYPE CLAIRE.T_SHOESIZE AS CHAR(2)
```

When the statement is executed, DB2 generates these cast functions:

```
FUNCTION CLAIRE.CHAR (CLAIRE.T_SHOESIZE) RETURNS SYSIBM.CHAR (2)
FUNCTION CLAIRE.T_SHOESIZE (SYSIBM.CHAR (2)) RETURNS CLAIRE.T_SHOESIZE
FUNCTION CLAIRE.T_SHOESIZE (SYSIBM.VARCHAR (2)) RETURNS CLAIRE.T_SHOESIZE
```

Notice that the third function enables the casting of a VARCHAR(2) to T_SHOESIZE. This additional function is created to enable casting a constant, such as 'AB', directly to the distinct type. Without the additional function, you would have to first cast 'AB', which has a data type of VARCHAR, to a data type of CHAR and then cast it to the distinct type.

You cannot explicitly drop a generated cast function. The cast functions that are generated for a distinct type are implicitly dropped when the distinct type is dropped with the DROP statement.

For each built-in data type that can be the source data type for a distinct type, Table 44 gives the names of the generated cast functions, the data types of the input parameters, and the data types of the values that the functions returns.

Table 44. CAST functions on distinct types

Source type name	Function name	Parameter-type	Return-type
CHAR CHARACTER	<i>distinct</i>	CHAR (n)	<i>distinct</i>
	CHAR	<i>distinct</i>	CHAR (n)
	<i>distinct</i>	VARCHAR (n)	<i>distinct</i>
VARCHAR	<i>distinct</i>	VARCHAR (n)	<i>distinct</i>
CHARACTER VARYING CHAR VARYING	VARCHAR	<i>distinct</i>	VARCHAR (n)
CLOB	<i>distinct</i>	CLOB (n)	<i>distinct</i>
	CLOB	<i>distinct</i>	CLOB (n)
BLOB	<i>distinct</i>	BLOB (n)	<i>distinct</i>
	BLOB	<i>distinct</i>	BLOB (n)
GRAPHIC	<i>distinct</i>	GRAPHIC (n)	<i>distinct</i>
	GRAPHIC	<i>distinct</i>	GRAPHIC (n)
	<i>distinct</i>	VARGRAPHIC (n)	<i>distinct</i>
VARGRAPHIC	<i>distinct</i>	VARGRAPHIC (n)	<i>distinct</i>
	VARGRAPHIC	<i>distinct</i>	VARGRAPHIC (n)
DBCLOB	<i>distinct</i>	DBCLOB (n)	<i>distinct</i>
	DBCLOB	<i>distinct</i>	DBCLOB (n)
SMALLINT	<i>distinct</i>	SMALLINT	<i>distinct</i>
	<i>distinct</i>	INTEGER	<i>distinct</i>
	SMALLINT	<i>distinct</i>	SMALLINT
INTEGER	<i>distinct</i>	INTEGER	<i>distinct</i>
	INTEGER	<i>distinct</i>	INTEGER

Table 44. CAST functions on distinct types (continued)

Source type name	Function name	Parameter-type	Return-type
DECIMAL	<i>distinct</i>	DECIMAL (p,s)	<i>distinct</i>
	DECIMAL	<i>distinct</i>	DECIMAL (p,s)
NUMERIC	<i>distinct</i>	DECIMAL (p,s)	<i>distinct</i>
	DECIMAL	<i>distinct</i>	DECIMAL (p,s)
REAL	<i>distinct</i>	REAL	<i>distinct</i>
	<i>distinct</i>	DOUBLE	<i>distinct</i>
	REAL	<i>distinct</i>	REAL
FLOAT(<i>n</i>) where <i>n</i> <=21	<i>distinct</i>	REAL	<i>distinct</i>
	<i>distinct</i>	DOUBLE	<i>distinct</i>
	REAL	<i>distinct</i>	REAL
FLOAT(<i>n</i>) where <i>n</i> >21	<i>distinct</i>	DOUBLE	<i>distinct</i>
	DOUBLE	<i>distinct</i>	DOUBLE
FLOAT	<i>distinct</i>	DOUBLE	<i>distinct</i>
	DOUBLE	<i>distinct</i>	DOUBLE
DOUBLE	<i>distinct</i>	DOUBLE	<i>distinct</i>
	DOUBLE	<i>distinct</i>	DOUBLE
DOUBLE PRECISION	<i>distinct</i>	DOUBLE	<i>distinct</i>
	DOUBLE	<i>distinct</i>	DOUBLE
DATE	<i>distinct</i>	DATE	<i>distinct</i>
	DATE	<i>distinct</i>	DATE
TIME	<i>distinct</i>	TIME	<i>distinct</i>
	TIME	<i>distinct</i>	TIME
TIMESTAMP	<i>distinct</i>	TIMESTAMP	<i>distinct</i>
	TIMESTAMP	<i>distinct</i>	TIMESTAMP
ROWID	<i>distinct</i>	ROWID	<i>distinct</i>
	ROWID	<i>distinct</i>	ROWID

Notes: In the table, *distinct* represents *distinct-type-name*.

NUMERIC and FLOAT are not recommended when creating a distinct type for a portable application. Use DECIMAL and DOUBLE (or REAL) instead.

Built-in functions: When a distinct type is defined, the built-in functions (such as AVG, MAX, and LENGTH) are not automatically supported for the distinct type. You can use a built-in function on a distinct type only after a sourced user-defined function, which is based on the built-in function, has been created for the distinct type. For information on defining sourced user-defined functions, see “CREATE FUNCTION (sourced)” on page 571.

Compatibility: The WITH COMPARISONS clause, which specifies that system-generated comparison operators are to be created for comparing two instances of the distinct type, can be used but only for compatibility with versions of DB2 that require it. If the source data type is either BLOB, CLOB, or DBCLOB and WITH COMPAISONS is specified, an error occurs.

CREATE DISTINCT TYPE

Examples

Example 1: Create a distinct type named SHOESIZE that is sourced on an INTEGER data type.

```
CREATE DISTINCT TYPE SHOESIZE AS INTEGER;
```

The successful execution of this statement also generates two cast functions. Function INTEGER(SHOESIZE) returns a value with data type INTEGER, and function SHOESIZE(INTEGER) returns a value with distinct type SHOESIZE.

Example 2: Create a distinct type named MILES that is sourced on a DOUBLE data type.

```
CREATE DISTINCT TYPE MILES AS DOUBLE;
```

The successful execution of this statement also generates two cast functions. Function DOUBLE(MILES) returns a value with data type DOUBLE, and function MILES(DOUBLE) returns a value with distinct type MILES.

CREATE FUNCTION

The CREATE FUNCTION statement registers a user-defined function with a database server. You can register four different types of functions with this statement, each of which is described separately.

- External scalar

The function is written in a programming language and returns a scalar value. The external executable is registered with a database server along with various attributes of the function. See “CREATE FUNCTION (external scalar)” on page 530.

- External table

The function is written in a programming language and returns a complete table. The external executable is registered with a database server along with various attributes of the function. See “CREATE FUNCTION (external table)” on page 553.

- Sourced

The function is implemented by invoking another function (either built-in, external, or sourced) that is already registered with a database server. See “CREATE FUNCTION (sourced)” on page 571.

- SQL

The function content is specified in the RETURN clause of the CREATE FUNCTION statement. See “CREATE FUNCTION (SQL scalar)” on page 585.

CREATE FUNCTION (external scalar)

This CREATE FUNCTION statement registers a user-defined external scalar function with a database server.

A scalar function returns a single value each time it is invoked.

Invocation

This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is implicitly or explicitly specified.

Authorization

The privilege set defined below must include at least one of the following:

- The CREATEIN privilege for the schema or all schemas
- SYSADM or SYSCTRL authority

The authorization ID that matches the schema name implicitly has the CREATEIN privilege on the schema.

Privilege set: If the statement is embedded in an application program, the privilege set is the privileges that are held by the authorization ID of the owner of the plan or package.

If the statement is dynamically prepared, the privilege set is the privileges that are held by the SQL authorization ID of the process. The specified function name can include a schema name (a qualifier). However, if the schema name is not the same as the SQL authorization ID, one of the following conditions must be met:

- The privilege set includes SYSADM or SYSCTRL authority.
- The SQL authorization ID of the process has the CREATEIN privilege on the schema.

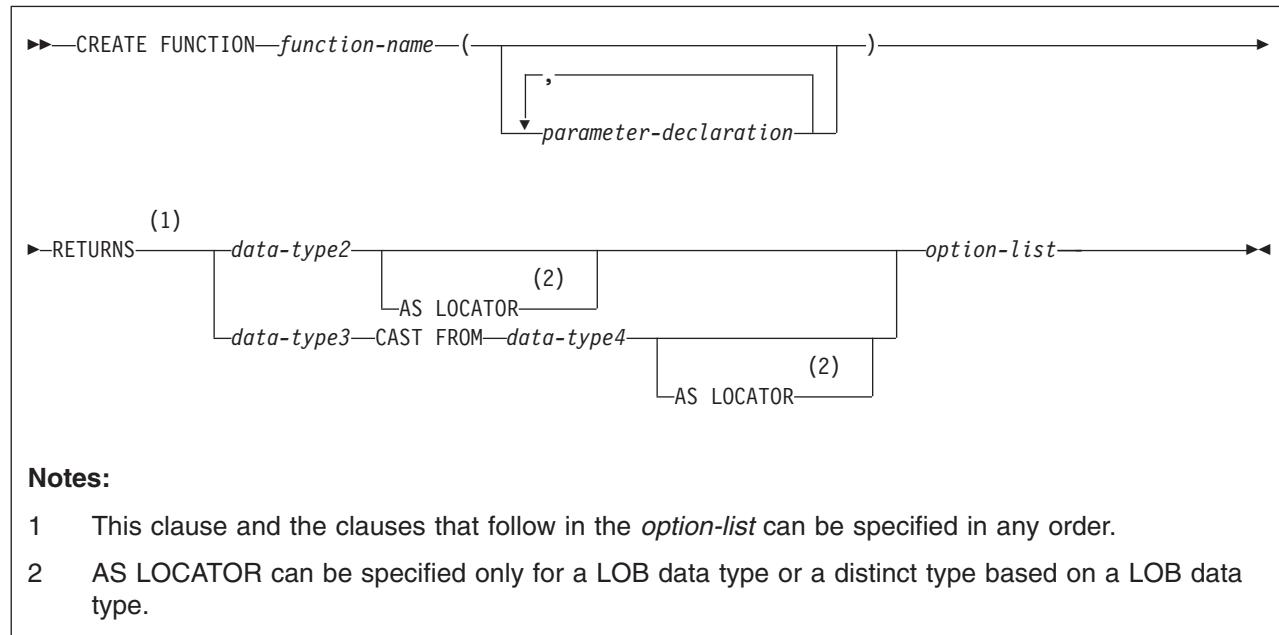
Additional privileges are required if the function uses a table as a parameter, refers to a distinct type, or is to run in a MVS workload manager (WLM) environment.

These privileges are:

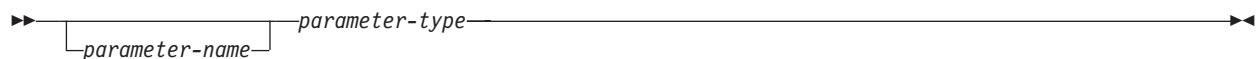
- The SELECT privilege on any table that is an input parameter to the function.
- The USAGE privilege on each distinct type that the function references.
- Authority to create programs in the specified WLM environment. This authorization is obtained from an external security product, such as RACF.

When LANGUAGE is JAVA and a *jar-name* is specified in the EXTERNAL NAME clause, the privilege set must include USAGE on the JAR, the Java ARchive file.

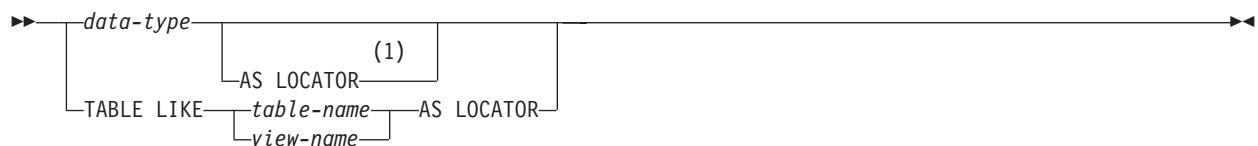
Syntax



parameter-declaration:



parameter-type:



Notes:

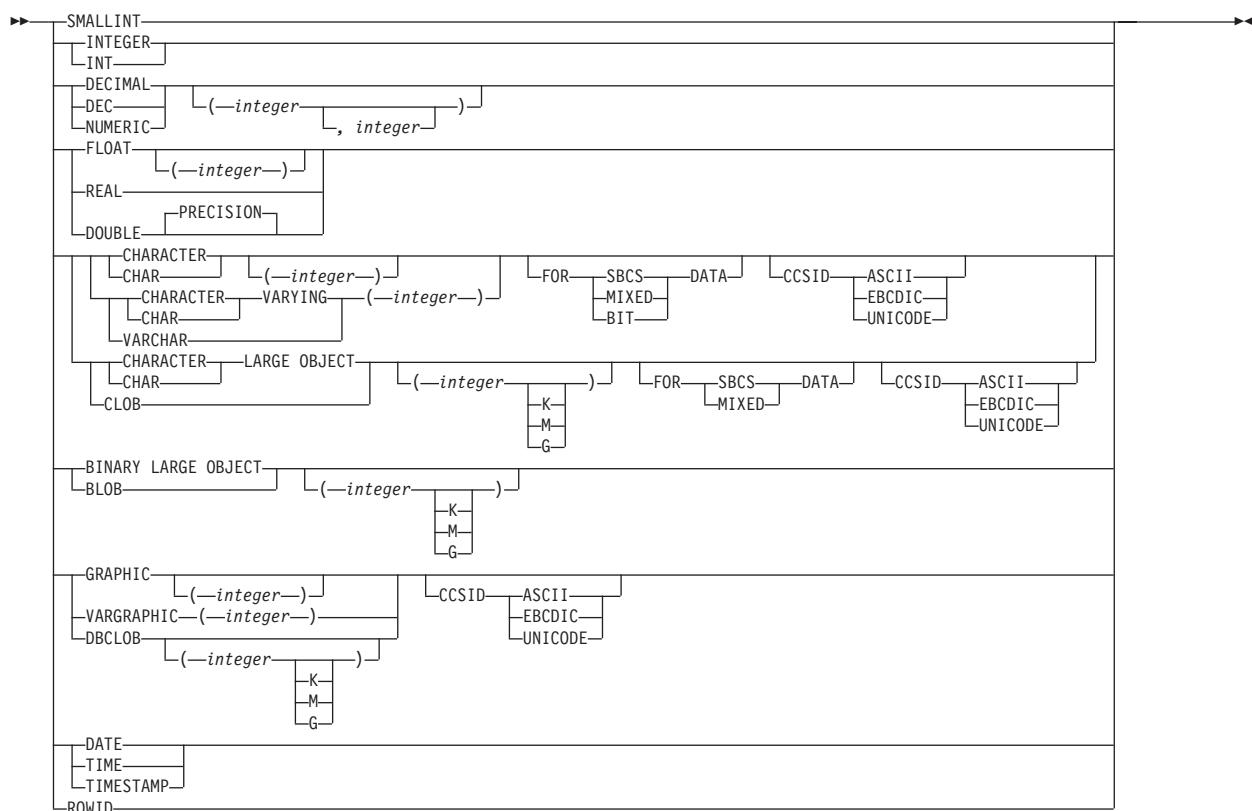
- 1 AS LOCATOR can be specified only for a LOB data type or a distinct type based on a LOB data type.

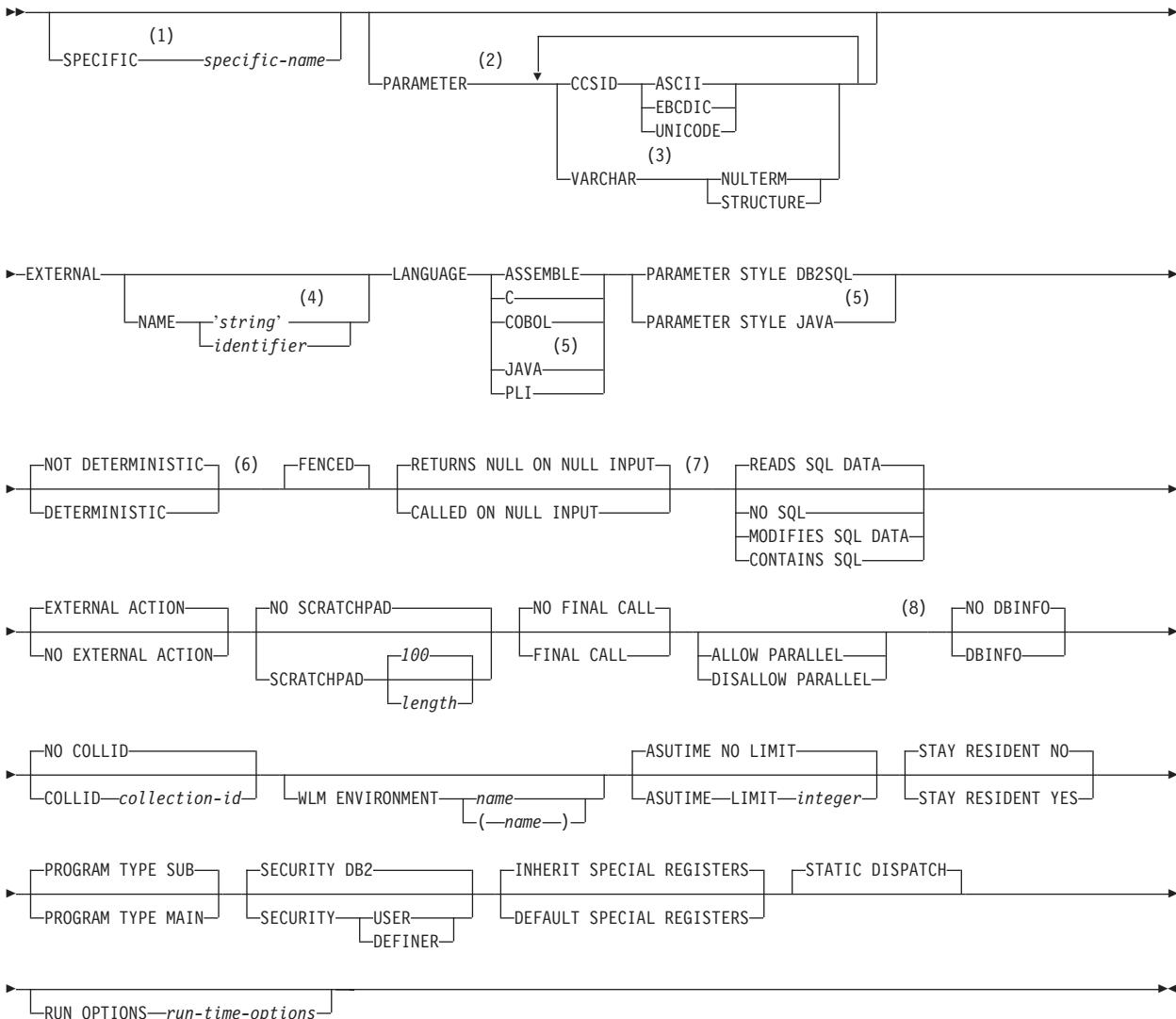
CREATE FUNCTION (external scalar)

data-type:

```
►► [built-in-data-type] [distinct-type-name] ◀◀
```

built-in-data-type:

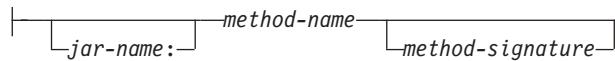


option-list:**Notes:**

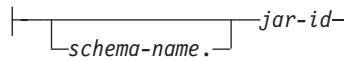
- 1 The clauses in the *option-list* can be specified in any order.
- 2 The same clause must not be specified more than once.
- 3 The VARCHAR clause can only be specified if LANGUAGE C is specified.
- # 4 With LANGUAGE JAVA, use a valid *external-java-routine-name*.
- I 5 LANGUAGE JAVA or PARAMETER STYLE JAVA cannot be specified without the other.
- 6 Synonyms include NOT NULL CALL for RETURNS NULL ON NULL INPUT and NULL CALL for CALLED ON NULL INPUT.
- 7 Synonyms include NOT DETERMINISTIC for NOT VARIANT and NOT VARIANT for DETERMINISTIC.
- 8 If NOT DETERMINISTIC, EXTERNAL ACTION, SCRATCHPAD, or FINAL CALL is specified, DISALLOW PARALLEL is the default.

CREATE FUNCTION (external scalar)

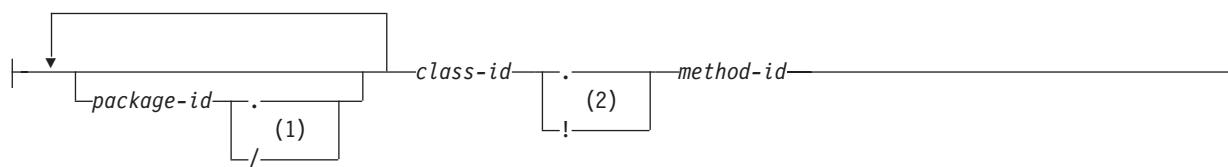
external-java-routine-name:



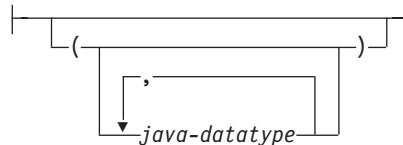
jar-name:



method-name:



method-signature:



Notes:

1 The slash (/) is supported for compatibility with DB2 for OS/390 Version 5 and Version 6.

2 The exclamation point (!) is supported for compatibility with DB2 UWO.

Description

function-name

Names the user-defined function. The name is implicitly or explicitly qualified by a schema name. The combination of name, schema name, the number of parameters, and the data type of each parameter²⁹ (without regard for any length, precision, scale, subtype or encoding scheme attributes of the data type) must not identify a user-defined function that exists at the current server.

You can use the same name for more than one function if the function signature of each function is unique.

- The unqualified form of *function-name* is a long SQL identifier.

The name must not be any of the following system-reserved keywords even if you specify them as delimited identifiers:

ALL
AND
ANY

LIKE
MATCH
NOT

UNIQUE
UNKNOWN
=

29. If the function has more than 30 parameters, only the first 30 parameters are used to determine whether the function is unique.

BETWEEN	NULL	=
DISTINCT	ONLY	<
EXCEPT	OR	≤
EXISTS	OVERLAPS	¬≤
FALSE	SIMILAR	>
FOR	SOME	≥
FROM	TABLE	¬≥
IN	TRUE	↔
IS	TYPE	

The unqualified function name is implicitly qualified with a schema name according to the following rules:

- If the statement is embedded in a program, the schema name is the authorization ID in the QUALIFIER bind option when the plan or package was created or last rebound. If QUALIFIER was not specified, the schema name is the owner of the plan or package.
- If the statement is dynamically prepared, the schema name is the SQL authorization ID in the CURRENT SQLID special register.
- The qualified form of *function-name* is a short SQL identifier (the schema name) followed by a period and a long SQL identifier.

The schema name must not begin with 'SYS' unless the schema name is 'SYSADM'.

The owner of the function is determined by how the CREATE FUNCTION statement is invoked:

- If the statement is embedded in a program, the owner is the authorization ID of the owner of the plan or package.
- If the statement is dynamically prepared, the owner is the SQL authorization ID in the CURRENT SQLID special register.

The owner is implicitly given the EXECUTE privilege with the GRANT option for the function.

(parameter-declaration,...)

#

Identifies the number of input parameters of the function, and specifies the data type of each parameter. All the parameters for a function are input parameters and are nullable. There must be one entry in the list for each parameter that the function expects to receive. Although not required, you can give each parameter a name.

A function can have no parameters. In this case, you must code an empty set of parentheses, for example:

```
CREATE FUNCTION WOOFER()
```

parameter-name

Specifies the name of the input parameter. The name is a long SQL identifier, and each name in the parameter list must not be the same as any other name.

data-type

Specifies the data type of the input parameter. The data type can be a built-in data type or a distinct type.

built-in-data-type

The data type of the input parameter is a built-in data type. You can use the same built-in data types as for the CREATE TABLE statement except LONG VARCHAR or LONG VARGRAPHIC. Use VARCHAR or VARGRAPHIC with an explicit length instead.

CREATE FUNCTION (external scalar)

If you do not specify a specific value for the data types that have length, precision, or scale attributes (CHAR, GRAPHIC, DECIMAL, NUMERIC, FLOAT), the defaults are as follows:

CHAR	CHAR(1)
GRAPHIC	GRAPHIC(1)
DECIMAL	DECIMAL(5,0)
FLOAT	DOUBLE (length of 8)

For information on the data types, including the subtype of character data types (the FOR *subtype* DATA clause), see “built-in-data-type” on page 658.

For parameters with a string data type, the CCSID clause indicates whether the encoding scheme of the parameter value is ASCII, EBCDIC, or UNICODE. If you do not specify CCSID ASCII, CCSID EBCDIC, or CCSID UNICODE, the encoding scheme is the value of field DEF ENCODING SCHEME on installation panel DSNTIPF.

distinct-type-name

The data type of the input parameter is a distinct type. Any length, precision, scale, subtype, or encoding scheme attributes for the parameter are those of the source type of the distinct type.

If you specify the name of the distinct type without a schema name, DB2 resolves the schema name by searching the schemas in the SQL path.

The implicitly or explicitly specified encoding scheme of all the parameters with a string data type must be the same—either all ASCII, all EBCDIC, or all UNICODE.

Although parameters with a character data type have an implicitly or explicitly specified subtype (BIT, SBCS, or MIXED), the function program can receive character data of any subtype. Therefore, conversion of the input data to the subtype of the parameter might occur when the function is invoked. An error occurs if mixed data that actually contains DBCS characters is used as the value for an input parameter that is declared with an SBCS subtype.

Parameters with a datetime data type or a distinct type are passed to the function as a different data type:

- A datetime type parameter is passed as a character data type, and the data is passed in ISO format.

The encoding scheme for a datetime type parameter is the same as the implicitly or explicitly specified encoding scheme of any character or graphic string parameters. If no character or graphic string parameters are passed, the encoding scheme is the value of field DEF ENCODING SCHEME on installation panel DSNTIPF.

- A distinct type parameter is passed as the source type of the distinct type.

AS LOCATOR

Specifies that a locator to the value of the parameter is passed to the function instead of the actual value. Specify AS LOCATOR only for parameters with a LOB data type or a distinct type based on a LOB

data type. Passing locators instead of values can result in fewer bytes being passed to the function, especially when the value of the parameter is very large.

The AS LOCATOR clause has no effect on determining whether data types can be promoted, nor does it affect the function signature, which is used in function resolution.

TABLE LIKE *table-name* or *view-name* AS LOCATOR

Specifies that the parameter is a transition table. However, when the function is invoked, the actual values in the transition table are not passed to the function. A single value is passed instead. This single value is a locator to the table, which the function uses to access the columns of the transition table. A function with a table parameter can only be invoked from the triggered action of a trigger.

The use of TABLE LIKE provides an implicit definition of the transition table. It specifies that the transition table has the same number of columns as the identified table or view. The columns have the same data type, length, precision, scale, subtype, and encoding scheme as the identified table or view, as they are described in catalog tables SYSCOLUMNS and SYSTABLESPACE.

The *name* specified after TABLE LIKE must identify a table or view that exists at the current server. The name must not identify a declared temporary table. The name does not have to be the same name as the table that is associated with the transition table for the trigger. An unqualified table or view name is implicitly qualified according to the following rules:

- If the CREATE FUNCTION statement is embedded in a program, the implicit qualifier is the authorization ID in the QUALIFIER bind option when the plan or package was created or last rebound. If QUALIFIER was not used, the implicit qualifier is the owner of the plan or package.
- If the CREATE FUNCTION statement is dynamically prepared, the implicit qualifier is the SQL authorization ID in the CURRENT_SQLID special register.

When the function is invoked, the corresponding columns of the transition table identified by the table locator and the table or view identified in the TABLE LIKE clause must have the same definition. The data type, length, precision, scale, and encoding scheme of these columns must match exactly. The description of the table or view at the time the CREATE FUNCTION statement was executed is used.

Additionally, a character FOR BIT DATA column of the transition table cannot be passed as input for a table parameter for which the corresponding column of the table specified at the definition is not defined as character FOR BIT DATA. (The definiton occurs with the CREATE FUNCTION statement.) Likewise, a character column of the transition table that is not FOR BIT DATA cannot be passed as input for a table parameter for which the corresponding column of the table specified at the definition is defined as character FOR BIT DATA.

For more information about using table locators, see *DB2 Application Programming and SQL Guide*.

CREATE FUNCTION (external scalar)

RETURNS

Identifies the output of the function. Consider this clause in conjunction with the optional CAST FROM clause.

data-type2

Specifies the data type of the output. The output parameter is nullable.

The same considerations that apply to the data type of input parameter, as described under “data-type” on page 535, apply to the data type of the output of the function.

AS LOCATOR

Specifies that the function returns a locator to the value rather than the actual value. You can specify AS LOCATOR only if the output from the function has a LOB data type or a distinct type based on a LOB data type.

data-type3 CAST FROM data-type4

Specifies the data type of the output of the function (*data-type4*) and the data type in which that output is returned to the invoking statement (*data-type3*). The two data types can be different. For example, for the following definition, the function returns a DOUBLE value, which DB2 converts to a DECIMAL value and then passes to the statement that invoked the function:

```
CREATE FUNCTION SQRT(DECIMAL(15,0))
  RETURNS DECIMAL(15,0) CAST FROM DOUBLE
  ...
```

The value of *data-type4* must not be a distinct type and must be castable to *data-type3*. The value for *data-type3* can be any built-in data type or distinct type. (For information on casting data types, see “Casting between data types” on page 62.) The encoding scheme of the parameters, if they are string data types, must be the same.

AS LOCATOR

Specifies that the function returns a locator to the value rather than the value. You can specify AS LOCATOR only if *data-type4* is a LOB data type or a distinct type based on a LOB data type.

SPECIFIC *specific-name*

Specifies a unique name for the function. The name is implicitly or explicitly qualified with a schema name. The name, including the schema name, must not identify the specific name of another function that exists at the current server.

The unqualified form of *specific-name* is a long SQL identifier. The qualified form is a short SQL identifier (the schema name) followed by a period and a long SQL identifier.

If you do not specify a schema name, it is the same as the explicit or implicit schema name of the function name (*function-name*). If you specify a schema name, it must be the same as the explicit or implicit schema name of the function name.

If you do not specify the SPECIFIC clause, the default specific name is the name of the function. However, if the function name does not provide a unique specific name or if the function name is a single asterisk, DB2 generates a specific name in the form of:

SQLxxxxxxxxxxxx

where 'xxxxxxxxxxxx' is a string of 12 characters that make the name unique.

The specific name is stored in the SPECIFIC column of the SYSROUTINES catalog table. The specific name can be used to uniquely identify the function in several SQL statements (such as ALTER FUNCTION, COMMENT ON, DROP, GRANT, and REVOKE) and must be used in DB2 commands (START FUNCTION, STOP FUNCTION, and DISPLAY FUNCTION). However, the function cannot be invoked by its specific name.

```
#  
#  
#  
#  
PARAMETER CCSID or VARCHAR  
Specifies the encoding scheme for string parameters, and in the case of  
LANGUAGE C, specifies that representation of variable length string  
parameters.  
  
#  
#  
#  
#  
CCSID  
Indicates whether the encoding scheme for string parameters is ASCII,  
EBCDIC, or UNICODE. The default encoding scheme is the value specified  
in the CCSID clauses of the parameter list or RETURNS clause, or in the  
field DEF ENCODING SCHEME on installation panel DSNTIPF.  
  
#  
#  
#  
#  
#  
#  
This clause provides a convenient way to specify the encoding scheme for  
all string parameters. If individual CCSID clauses are specified for individual  
parameters in addition to this PARAMETER CCSID clause, the value  
specified in all of the CCSID clauses must be the same value that is  
specified in this clause.  
  
#  
#  
#  
#  
#  
VARCHAR  
Specifies that the representation of the values of varying length character  
string-parameters, including, if applicable, the output of the function, for  
functions which specify LANGUAGE C.  
  
#  
#  
#  
#  
#  
This option can only be specified if LANGUAGE C is also specified or  
SQLCODE -628, SQLSTATE 42613 is returned.  
  
#  
#  
NULTERM  
Specifies that variable length character string parameters are  
represented in a NUL-terminated string form.  
  
#  
#  
STRUCTURE  
Specifies that variable length character string parameters are  
represented in a VARCHAR structure form.  
  
#  
#  
#  
#  
#  
#  
Using the PARAMETER VARCHAR clause, there is no way to specify the  
VARCHAR form of an individual parameter as these is with PARAMETER  
CCSID. The PARAMETER VARCHAR clause only applies to parameters in  
a function's parameter list and in the RETURNS clause. It does not apply to  
system-generated parameters of the routine such as message tokens and  
DBINFO.  
  
#  
#  
#  
#  
#  
#  
In a data sharing environment, you should not specify the PARAMETER  
VARCHAR clause until all members of the data sharing group support the  
clause. If some group members support this clause and others do not, and  
PARAMETER VARCHAR is specified in an external routine, the routine will  
encounter different parameter forms depending on which group member  
invokes the routine.  
  
EXTERNAL  
Specifies the program that runs when the function is invoked.
```

CREATE FUNCTION (external scalar)

DB2 loads the load module when the function is invoked. The load module is created when the program that contains the function body is compiled and link-edited. The load module does not need to exist when the CREATE FUNCTION statement is executed. However, it must exist and be accessible by the current server when the function is invoked.

You can specify the EXTERNAL clause in one of the following ways:

EXTERNAL

EXTERNAL NAME PKJVSP1

EXTERNAL NAME 'PKJVSP1'

If you specify an external program name, you must use the NAME keyword. For example, this syntax is not valid:

EXTERNAL PKJVSP1

NAME '*string*' or *identifier*

Identifies the user-written code that implements the user-defined function.

If LANGUAGE is JAVA, '*string*' must be specified and enclosed in single quotation marks, with no extraneous blanks within the single quotation marks. It must specify a valid *external-java-routine-name*. If multiple '*string*'s are specified, the total length of all of them must not be greater than 1305 bytes and they must be separated by a space or a line break. Do not specify a JAR for a JAVA function for which NO SQL is also specified.

An *external-java-routine-name* contains the following parts:

jar-name

Identifies the name given to the JAR when it was installed in the database. The name contains *jar-id*, which can optionally be qualified with a schema. Examples are "myJar" and "mySchema.myJar." The unqualified *jar-id* is implicitly qualified with a schema name according to the following rules:

- If the statement is embedded in a program, the schema name is the authorization ID in the QUALIFIER bind option when the package or plan was created or last rebound. If the QUALIFIER was not specified, the schema name is the owner of the package or plan.
- If the statement is dynamically prepared, the schema name is the SQL authorization ID in the CURRENT SQLID special register.

If *jar-name* is specified, it must exist when the CREATE FUNCTION statement is processed.

If *jar-name* is not specified, the function is loaded from the class file directly instead of being loaded from a JAR file. DB2 for DB2 for OS/390 and z/OS searches the directories in the CLASSPATH associated with the WLM Environment. Environmental variables for Java routines are specified in a dataset identified in a JAVAENV DD card on the JCL used to start the address space for a WLM-managed function.

method-name

Identifies the name of the method and must not be longer than 254 bytes. Its package, class, and method ID's are specific to Java and as such are not limited to 18 bytes. In addition, the rules for what these can contain are not necessarily the same as the rules for an SQL ordinary identifier.

package-id

Identifies the package list that the class identifier is part of. If the class is part of a package, the method name must include the complete package prefix, such as "myPacks.UserFuncs." The Java virtual machine looks in the directory "/myPacks/UserFuncs/" for the classes.

class-id

Identifies the class identifier of the Java object.

method-id

Identifies the method identifier with the Java class to be invoked.

method-signature

Identifies a list of zero or more Java data types for the parameter list and must not be longer than 1024 bytes. Specify the *method-signature* if the user-defined function involves any input or output parameters that can be NULL. When the function being created is called, DB2 searches for a Java method with the exact *method-signature*. The number of *java-datatype* elements specified indicates how many parameters that the Java method must have.

A Java procedure can have no parameters. In this case, you code an empty set of parentheses for *method-signature*. If a Java *method-signature* is not specified, DB2 searches for a Java method with a signature derived from the default JDBC types associated with the SQL types specified in the parameter list of the CREATE FUNCTION statement.

For other values of LANGUAGE, the name can be a string constant that is no longer than 8 characters or a short identifier. It must conform to the naming conventions for MVS load modules. Alphabetical extenders for national languages can be used as the first character and as subsequent characters in the load module name.

If you do not specify the NAME clause, 'NAME *function-name*' is implicit. In this case, *function-name* must not be longer than 8 characters.

LANGUAGE

Specifies the application programming language in which the function program is written. All programs must be designed to run in IBM's Language Environment environment.

ASSEMBLE

The function is written in Assembler.

C The function is written in C or C++.

COBOL

The function is written in COBOL, including the object-oriented language extensions.

JAVA

The user-defined function is written in Java byte code and is executed in the OS/390 Java Virtual Machine. When LANGUAGE JAVA is specified, the EXTERNAL NAME clause must also be specified with a valid *external-java-routine-name* and PARAMETER STYLE must be specified with JAVA.

Do not specify LANGUAGE JAVA when SCRATCHPAD, FINAL CALL, DBINFO, PROGRAM TYPE MAIN, or RUN OPTIONS is specified.

CREATE FUNCTION (external scalar)

PLI

The function is written in PL/I.

PARAMETER STYLE

Specifies the linkage convention that the function program uses to receive input parameters from and pass return values to the invoking SQL statement.

DB2SQL

Indicates that parameters for indicator variables are associated with each input and return value to allow for null values. The parameters that are passed between the invoking SQL statement and the function include:

- The first n parameters are the input parameters that are specified for the function
- A parameter for the result of the function
- n parameters for the indicator variables for the input parameters
- A parameter for the indicator variable for the result
- The SQLSTATE to be returned to DB2
- The qualified name of the function
- The specific name of the function
- The SQL diagnostic string to be returned to DB2

Zero to three additional parameters might also be passed:

- The scratchpad, if SCRATCHPAD is specified
- The call type, if FINAL CALL is specified
- The DBINFO structure, if DBINFO is specified

#

JAVA

Indicates that the user-defined function uses a convention for passing parameters that conforms to the Java and SQLJ specifications.

PARAMETER STYLE JAVA can be specified only if LANGUAGE is JAVA. JAVA must be specified for PARAMETER STYLE when LANGUAGE is JAVA.

|
|
|
|

NOT DETERMINISTIC or DETERMINISTIC

Specifies whether the function returns the same results for identical input arguments.

NOT DETERMINISTIC

The function might not return the same result for identical input arguments. The function depends on some state values that affect the results. DB2 uses this information to disable the merging of views and table expressions when processing SELECT, UPDATE, DELETE, or INSERT statements that reference this function. An example of a function that is not deterministic is one that generates random numbers, or any function that contains SQL statements.

NOT DETERMINISTIC is the default.

Some functions that are not deterministic can receive incorrect results if the function is executed by parallel tasks. Specify the DISALLOW PARALLEL clause for these functions.

DETERMINISTIC

The function always returns the same result for identical input arguments.

An example of a deterministic function is a function that calculates the square root of the input. DB2 uses this information to enable the merging of views and table expressions for SELECT, UPDATE, DELETE, or INSERT statements that reference this function. DETERMINISTIC is not the default. If applicable, specify DETERMINISTIC to prevent non-optimal access paths from being chosen for SQL statements that reference this function.

#

DB2 does not verify that the function program is consistent with the specification of DETERMINISTIC or NOT DETERMINISTIC.

FENCED

Specifies that the external function runs in an external address space to prevent the function from corrupting DB2 storage.

FENCED is the default.

RETURNS NULL ON NULL INPUT or CALLED ON NULL INPUT

Specifies whether the function is called if any of the input arguments is null at execution time.

RETURNS NULL ON INPUT

The function is not called if any of the input arguments is null. The result is the null value. RETURNS NULL ON INPUT is the default.

CALLED ON NULL INPUT

The function is called regardless of whether any of the input arguments is null, making the function responsible for testing for null argument values.

The function can return a null or nonnull value.

NO SQL, MODIFIES SQL DATA, READS SQL DATA, or CONTAINS SQL

Indicates whether the function can execute any SQL statements and, if so, what type. DB2 verifies that the SQL issued by the function is consistent with this specification. See Table 72 on page 972 for a detailed list of the SQL statements that can be executed under each data access indication.

NO SQL

The function does not execute SQL statements. Do not specify NO SQL for a JAVA function that uses a JAR.

MODIFIES SQL DATA

The function can execute any SQL statement except those statements that are not supported in any function. Do not specify MODIFIES SQL DATA when ALLOW PARALLEL is specified.

READS SQL DATA

The function does not execute SQL statements that modify data. SQL statements that are not supported in any function return a different error.

READS SQL DATA is the default.

CONTAINS SQL

The function does not execute SQL statements that read or modify data. SQL statements that are not supported in any function return a different error.

EXTERNAL ACTION or NO EXTERNAL ACTION

Specifies whether the function takes an action that changes the state of an object that DB2 does not manage. An example of an external action is sending a message or writing a record to a file.

Because DB2 uses the RRS attachment for external functions, DB2 can participate in two-phase commit with any other resource manager that uses RRS. For resource managers that do not use RRS, there is no coordination of commit or rollback operations on non-DB2 resources.

EXTERNAL ACTION

The function can take an action that changes the state of an object that DB2 does not manage.

CREATE FUNCTION (external scalar)

Some SQL statements that invoke functions with external actions can result in incorrect results if parallel tasks execute the function. For example, if the function sends a note for each initial call to it, one note is sent for each parallel task instead of once for the function. Specify the DISALLOW PARALLEL clause for functions that do not work correctly with parallelism.

If you specify EXTERNAL ACTION, DB2:

- Materializes the views and table expressions in SELECT, UPDATE, DELETE or INSERT statements that refer to the function. This materialization can adversely affect the access paths that are chosen for the SQL statements that reference this function. Do not specify EXTERNAL ACTION if the function does not have an external action.
- Does not move the function from one task control block (TCB) to another between FETCH operations.
- Does not allow another function or stored procedure to use the TCB until the cursor is closed. This is also applicable for cursors declared WITH HOLD.

The only changes to resources made outside of DB2 that are under the control of commit and rollback operations are those changes made under RRS control.

EXTERNAL ACTION is the default.

NO EXTERNAL ACTION

The function does not take any action that changes the state of an object that DB2 does not manage. DB2 uses this information to enable the merging of views and table expressions for SELECT, UPDATE, DELETE, or INSERT statements that reference this function. NO EXTERNAL ACTION is not the default. If applicable, specify NO EXTERNAL ACTION to prevent non-optimal access paths from being chosen for SQL statements that reference this function.

DB2 does not verify that the function program is consistent with the specification of EXTERNAL ACTION or NO EXTERNAL ACTION.

NO SCRATCHPAD or SCRATCHPAD

Specifies whether DB2 is to provide a scratchpad for the function. It is strongly recommended that external functions be reentrant, and a scratchpad provides an area for the function to save information from one invocation to the next.

NO SCRATCHPAD

A scratchpad is not allocated and passed to the function. NO SCRATCHPAD is the default.

SCRATCHPAD *length*

When the function is invoked for the first time, DB2 allocates memory for a scratchpad. A scratchpad has the following characteristics:

- *length* must be between 1 and 32767. The default value is 100 bytes.
- DB2 initializes the scratchpad to all binary zeros (X'00"s).
- The scope of a scratchpad is the SQL statement. For each reference to the function in an SQL statement, there is one scratchpad. For example, assuming that function UDFX was defined with the SCRATCHPAD keyword, three scratchpads are allocated for the three references to UDFX in the following SQL statement:

```
SELECT A, UDFX(A) FROM TABLEB  
WHERE UDFX(A) > 103 OR UDFX(A) < 19;
```

If the function is run under parallel tasks, one scratchpad is allocated for each parallel task of each reference to the function in the SQL statement. This can lead to unpredictable results. For example, if a function uses the scratchpad to count the number of times that it is invoked, the count reflects the number of invocations done by the parallel task and not the SQL statement. Specify the DISALLOW PARALLEL clause for functions that will not work correctly with parallelism.

- The scratchpad is persistent. DB2 preserves its content from one invocation of the function to the next. Any changes that the function makes to the scratchpad on one call are still there on the next call. DB2 initializes the scratchpads when it begins to execute an SQL statement. DB2 does not reset scratchpads when a correlated subquery begins to execute.
- The scratchpad can be a central point for the system resources that the function acquires. If the function acquires system resources, specify FINAL CALL to ensure that DB2 calls the function one more time so that the function can free those system resources.

Each time the function invoked, DB2 passes an additional argument to the function that contains the address of the scratchpad.

If you specify SCRATCHPAD, DB2:

- Does not move the function from one task control block (TCB) to another between FETCH operations.
- Does not allow another function or stored procedure to use the TCB until the cursor is closed. This is also applicable for cursors declared WITH HOLD.

| Do not specify SCRATCHPAD when LANGUAGE JAVA is specified.

NO FINAL CALL or FINAL CALL

Specifies whether a *final call* is made to the function. A final call enables the function to free any system resources that it has acquired. A final call is useful when the function has been defined with the SCRATCHPAD keyword and the function acquires system resource and anchors them in the scratchpad.

NO FINAL CALL

A final call is not made to the function. The function does not receive an additional argument that specifies the type of call. NO FINAL CALL is the default.

FINAL CALL

A final call is made to the function. To differentiate between final calls and other calls, the function receives an additional argument that specifies the type of call. The types of calls are:

Normal call

SQL arguments are passed and the function is expected to return a result.

First call

The first call to the function for this reference to the function in this SQL statement. A first call is a normal call—SQL arguments are passed and the function is expected to return a result.

Final call

The last call to the function to enable the function to free resources.

CREATE FUNCTION (external scalar)

A final call is not a normal call. If an error occurs, DB2 attempts to make the final call unless the function abended. A final call occurs at these times:

- *End of statement*: When the cursor is closed for cursor-oriented statements, or the execution of the statement has completed.
- *End of a parallel task*: When the function is executed by parallel tasks.
- *End of transaction*: When normal end of statement processing does not occur. For example, the logic of an application, for some reason, bypasses closing the cursor.

If a commit operation occurs while a cursor defined as WITH HOLD is open, a final call is made when the cursor is closed or the application ends. If a commit occurs at the end of a parallel task, a final call is made regardless of whether a cursor defined as WITH HOLD is open.

If a commit, rollback, or abort operation causes the final call, the function cannot issue any SQL statements when it is invoked.

Some functions that use a final call can receive incorrect results if parallel tasks execute the function. For example, if a function sends a note for each final call to it, one note is sent for each parallel task instead of once for the function. Specify the DISALLOW PARALLEL clause for functions that have inappropriate actions when executed in parallel.

| Do not specify FINAL CALL when LANGUAGE JAVA is specified.

ALLOW or DISALLOW PARALLEL

For a single reference to the function, specifies whether parallelism can be used when the function is invoked. Although parallelism can be used for most scalar functions, some functions such as those that depend on a single copy of the scratchpad cannot be invoked with parallel tasks. Consider these characteristics when determining which clause to use:

- If all invocations of the function are completely independent from one another, specify ALLOW PARALLEL.
- If each invocation of the function updates the scratchpad, providing values that are of interest to the next invocation, such as incrementing a counter, specify DISALLOW PARALLEL.
- If the scratchpad is used only so that some expensive initialization processing is performed a minimal number of times, specify ALLOW PARALLEL.
- If the function performs some external action that should apply to only one partition, specify DISALLOW PARALLEL.
- If the function is defined with MODIFIES SQL DATA, specify DISALLOW PARALLEL, not ALLOW PARALLEL.

ALLOW PARALLEL

Specifies that DB2 can consider parallelism for the function. Parallelism is not forced on the SQL statement that invokes the function or on any SQL statement in the function. Existing restrictions on parallelism apply.

DISALLOW PARALLEL

Specifies that DB2 does not consider parallelism for the function.

The default is DISALLOW PARALLEL, if you specify one or more of the following clauses:

- NOT DETERMINISTIC
- EXTERNAL ACTION
- FINAL CALL
- MODIFIES SQL DATA
- SCRATCHPAD

Otherwise, ALLOW PARALLEL is the default.

NO DBINFO or DBINFO

Specifies whether specific information that DB2 knows is passed to the function when it is invoked.

NO DBINFO

No additional information is passed. NO DBINFO is the default.

DBINFO

An additional argument is passed when the function is invoked. The argument is a structure that contains information such as the application run-time authorization ID, the schema name, the name of a table or column that the function might be inserting into or updating, and identification of the database server that invoked the function. For details about the argument and its structure, see *DB2 Application Programming and SQL Guide*.

| Do not specify DBINFO when LANGUAGE JAVA is specified.

NO COLLID or COLLID *collection-id*

Identifies the package collection that is to be used when the function is executed. This is the package collection into which the DBRM that is associated with the function program is bound.

NO COLLID

The package collection for the function is the same as the package collection of the program that invokes the function. If a trigger invokes the function, the collection of the trigger package is used. If the invoking program does not use a package, the package collection is the value of the CURRENT PACKAGESET special register.

NO COLLID is the default.

COLLID *collection-id*

The name of the package collection that is to be used when the function is executed.

WLM ENVIRONMENT

Identifies the MVS workload manager (WLM) application environment in which the function is to run. The *name* of the WLM environment is a long identifier.

If you do not specify WLM ENVIRONMENT, the function runs in the WLM-established stored procedure address space that is specified at installation time.

name

The WLM environment in which the function must run. If another user-defined function or a stored procedure calls the function and that calling routine is running in an address space that is not associated with the WLM environment, DB2 routes the function request to a different MVS address space.

(name,*)

When an SQL application program directly invokes the function, the WLM environment in which the function runs.

CREATE FUNCTION (external scalar)

If another user-defined function or a stored procedure calls the function, the function runs in same environment that the calling routine uses. In this case, authorization to run the function in the WLM environment is not checked because the authorization of the calling routine suffices.

Users must have the appropriate authorization to execute functions in the specified WLM environment. For an example of a RACF command that provides this authorization, see “Running external functions in WLM environments” on page 551.

ASUTIME

Specifies the total amount of processor time, in CPU service units, that a single invocation of the function can run. The value is unrelated to the ASUTIME column of the resource limit specification table.

When you are debugging a function, setting a limit can be helpful if the function gets caught in a loop. For information on service units, see *OS/390 MVS Initialization and Tuning Guide*.

NO LIMIT

There is no limit on the service units. NO LIMIT is the default.

LIMIT *integer*

The limit on the service units is a positive integer in the range of 1 to 2G. If the function uses more service units than the specified value, DB2 cancels the function.

STAY RESIDENT

Specifies whether the load module for the function is to remain resident in memory when the function ends.

NO

The load module is deleted from memory after the function ends. Use NO for non-reentrant functions. NO is the default.

YES

The load module remains resident in memory after the function ends. Use YES for reentrant functions.

PROGRAM TYPE

Specifies whether the function program runs as a main routine or a subroutine.

SUB

The function runs as a subroutine. SUB is the default.

MAIN

The function runs as a main routine. Do not specify PROGRAM TYPE MAIN when LANGUAGE JAVA is specified.

SECURITY

Specifies how the function interacts with an external security product, such as RACF, to control access to non-SQL resources.

DB2

The function does not require an external security environment. If the function accesses resources that an external security product protects, the access is performed using the authorization ID that is associated with the WLM-established stored procedure address space.

DB2 is the default.

USER

An external security environment should be established for the function. If

the function accesses resources that the external security product protects, the access is performed using the primary authorization ID of the process that invoked the function.

DEFINER

An external security environment should be established for the function. If the function accesses resources that the external security product protects, the access is performed using the authorization ID of the owner of the function.

INHERIT SPECIAL REGISTERS

Indicates that the values of special registers are inherited according to the rules listed in the table for characteristics of special registers in a user-defined function in Table 19 on page 93.

DEFAULT SPECIAL REGISTERS

Indicates that special registers are initialized to the default values, as indicated by the rules in the table for characteristics of special registers in a user-defined function in Table 19 on page 93.

STATIC DISPATCH

At function resolution time, DB2 chooses a function based on the static (or declared) types of the function parameters. STATIC DISPATCH is the default.

RUN OPTIONS *run-time-options*

Specifies the Language Environment run-time options to be used for the function. You must specify *run-time-options* as a character string that is no longer than 254 bytes. If you do not specify RUN OPTIONS or pass an empty string, DB2 does not pass any run-time options to Language Environment, and Language Environment uses its installation defaults. For a description of the Language Environment run-time options, see *OS/390 Language Environment for OS/390 & VM Programming Reference*.

Do not specify RUN OPTIONS when LANGUAGE JAVA is specified.

Notes

Choosing data types for parameters: When you choose the data types of the input and output parameters for your function, consider the rules of promotion that can affect the values of the parameters. (See “Promotion of data types” on page 61). For example, a constant that is one of the input arguments to the function might have a built-in data type that is different from the data type that the function expects, and more significantly, might not be promotable to that expected data type. Based on the rules of promotion, using the following data types for parameters is recommended:

- INTEGER instead of SMALLINT
- DOUBLE instead of REAL
- VARCHAR instead of CHAR
- VARGRAPHIC instead of GRAPHIC

For portability of functions across platforms that are not DB2 for OS/390 and z/OS, do not use the following data types, which might have different representations on different platforms:

- FLOAT. Use DOUBLE or REAL instead.
- NUMERIC. Use DECIMAL instead.

Specifying the encoding scheme for parameters: The implicitly or explicitly specified encoding scheme of all the parameters with a string data type (both input and output parameters) must be the same—either all ASCII or all EBCDIC.

CREATE FUNCTION (external scalar)

Determining the uniqueness of functions in a schema: At the current server, the function signature of each function, which is the qualified function name combined with the number and data types of the input parameters, must be unique. If the function has more than 30 input parameters, only the data types of the first 30 are used to determine uniqueness. This means that two different schemas can each contain a function with the same name that have the same data types for all of their corresponding data types. However, a single schema must not contain multiple functions with the same name that have the same data types for all of their corresponding data types.

When determining whether corresponding data types match, DB2 does not consider any length, precision, scale, subtype or encoding scheme attributes in the comparison. DB2 considers the synonyms of data types (DECIMAL and NUMERIC, REAL and FLOAT, and DOUBLE and FLOAT) a match. Therefore, CHAR(8) and CHAR(35) are considered to be the same, as are DECIMAL(11,2), DECIMAL(4,3), and NUMERIC(4,2).

Assume that the following statements are executed to create four functions in the same schema. The second and fourth statements fail because they create functions that are duplicates of the functions that the first and third statements created.

```
CREATE FUNCTION PART (INT, CHAR(15)) ...
CREATE FUNCTION PART (INTEGER, CHAR(40)) ...

CREATE FUNCTION ANGLE (DECIMAL(12,2)) ...
CREATE FUNCTION ANGLE (DEC(10,7)) ...
```

```
# Character string representation considerations: The PARAMETER VARCHAR clause is specific to LANGUAGE C functions because of the native use of NUL-terminated strings in C. VARCHAR structure representation is useful when character string data is known to contain embedded NUL-terminators. It is also useful when it cannot be guaranteed that character string data does not contain embedded NUL-terminators.
```

```
# PARAMETER VARCHAR does not apply to fixed length character strings,
# VARCHAR FOR BIT DATA, CLOB, DBCLOB, or implicitly generated parameters.
# The clause does not apply to VARCHAR FOR BIT DATA because BIT DATA can
# contain X'00' characters, and its value representation starts with length information.
# It does not apply to LOB data because a LOB value representation starts with
# length information.
```

```
# PARAMETER VARCHAR does not apply to optional parameters that are implicitly
# provided to an external function. For example, a CREATE FUNCTION statement for
# LANGUAGE C must also specify PARAMETER STYLE SQL, which returns an
# SQLSTATE NUL-terminated character string; that SQLSTATE will not be
# represented in VARCHAR structured form. Likewise, none of the parameters that
# represent the qualified name of the function, the specific name of the function, or
# the SQL diagnostic string that is returned to the database manager will be
# represented in VARCHAR structured form.
```

Overriding a built-in function: Giving an external function the same name as a built-in function is not a recommended practice unless you are trying to change the functionality of the built-in function.

If you do intend to create an external function with the same name as a built-in function, be careful to maintain the uniqueness of its function signature. If your function has the same name and data types of the corresponding parameters of the

built-in function but implements different logic, DB2 might choose the wrong function when the function is invoked with an unqualified function name. Thus, the application might fail, or perhaps even worse, run successfully but provide an inappropriate result.

Running external functions in WLM environments: You can use the WLM ENVIRONMENT clause to identify the MVS address space in which a function or is to run. Using different WLM environments lets you isolate one group of programs from another. For example, you might choose to isolate programs based on security requirements and place all payroll applications in one WLM environment because those applications deal with data, such as employee salaries.

To prevent a user from defining functions in sensitive WLM environments, DB2 invokes the external security manager to determine whether the user has authorization to issue CREATE FUNCTION statements that refer to the specified WLM environment. The following example shows the RACF command that authorizes DB2 user DB2USER1 to register a function on DB2 subsystem DB2A that runs in the WLM environment named PAYROLL.

```
PERMIT DB2A.WLMENV.PAYROLL CLASS(DSNR) ID(DB2USER1) ACCESS(READ)
```

Scrollable cursors specified with user-defined functions: A row can be fetched more than once with a scrollable cursor. Therefore, if a scrollable cursor is defined with a non-deterministic function in the select list of the cursor, a row can be fetched multiple times with different results for each fetch. (However, the value of a non-deterministic function in the WHERE clause of a scrollable cursor is captured when the cursor is opened and remains unchanged until the cursor is closed.) Similarly, if a scrollable cursor is defined with a user-defined function with external action, the action is executed with every fetch.

Examples

Example 1: Assume that you want to write an external function program in C that implements the following logic:

```
output = 2 * input - 4
```

The function should return a null value if and only if one of the input arguments is null. The simplest way to avoid a function call and get a null result when an input value is null is to specify RETURNS NULL ON NULL INPUT on the CREATE FUNCTION statement or allow it to be the default. Write the statement needed to register the function, using the specific name MINENULL1.

```
CREATE FUNCTION NTEST1 (SMALLINT)
RETURNS SMALLINT
EXTERNAL NAME 'NTESTMOD'
SPECIFIC MINENULL1
LANGUAGE C
DETERMINISTIC
NO SQL
FENCED
PARAMETER STYLE DB2SQL
RETURNS NULL ON NULL INPUT
NO EXTERNAL ACTION;
```

Example 2: Assume that user Smith wants to register an external function named CENTER in schema SMITH. The function program will be written in C and will be reentrant. Write the statement that Smith needs to register the function, letting DB2 generate a specific name for the function.

CREATE FUNCTION (external scalar)

```
CREATE FUNCTION CENTER (INTEGER, FLOAT)
  RETURNS FLOAT
  EXTERNAL NAME 'MIDDLE'
  LANGUAGE C
  DETERMINISTIC
  NO SQL
  FENCED
  PARAMETER STYLE DB2SQL
  NO EXTERNAL ACTION
  STAY RESIDENT YES;
```

Example 3: Assume that user McBride (who has administrative authority) wants to register an external function named CENTER in the SMITH schema. McBride plans to give the function specific name FOCUS98. The function program uses a scratchpad to perform some one-time only initialization and save the results. The function program returns a value with a FLOAT data type. Write the statement McBride needs to register the function and ensure that when the function is invoked, it returns a value with a data type of DECIMAL(8,4).

```
CREATE FUNCTION SMITH.CENTER (FLOAT, FLOAT, FLOAT)
  RETURNS DECIMAL(8,4) CAST FROM FLOAT
  EXTERNAL NAME 'CMOD'
  SPECIFIC FOCUS98
  LANGUAGE C
  DETERMINISTIC
  NO SQL
  FENCED
  PARAMETER STYLE DB2SQL
  NO EXTERNAL ACTION
  SCRATCHPAD
  NO FINAL CALL;
```

Example 4: The following example registers a Java user-defined function that returns the position of the first vowel in a string. The user-defined function is written in Java, is to be run fenced, and is the FINDVWL method of class JAVAUDFS.

```
CREATE FUNCTION FINDV (CLOB(100K))
  RETURNS INTEGER
  FENCED
  LANGUAGE JAVA
  PARAMETER STYLE JAVA
  EXTERNAL NAME 'JAVAUDFS.FINDVWL'
  NO EXTERNAL ACTION
  CALLED ON NULL INPUT
  DETERMINISTIC
  NO SQL;
```

CREATE FUNCTION (external table)

This CREATE FUNCTION statement registers a user-defined external table function with a database server.

A table function can be used in the FROM clause of a SELECT. It returns a table to the SELECT one row at a time.

Invocation

This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is implicitly or explicitly specified.

Authorization

The privilege set defined below must include at least one of the following:

- The CREATEIN privilege for the schema or all schemas
- SYSADM or SYSCTRL authority

The authorization ID that matches the schema name implicitly has the CREATEIN privilege on the schema.

Privilege set: If the statement is embedded in an application program, the privilege set is the privileges that are held by the authorization ID of the owner of the plan or package.

If the statement is dynamically prepared, the privilege set is the privileges that are held by the SQL authorization ID of the process. The specified function name can include a schema name (a qualifier). However, if the schema name is not the same as the SQL authorization ID, one of the following conditions must be met:

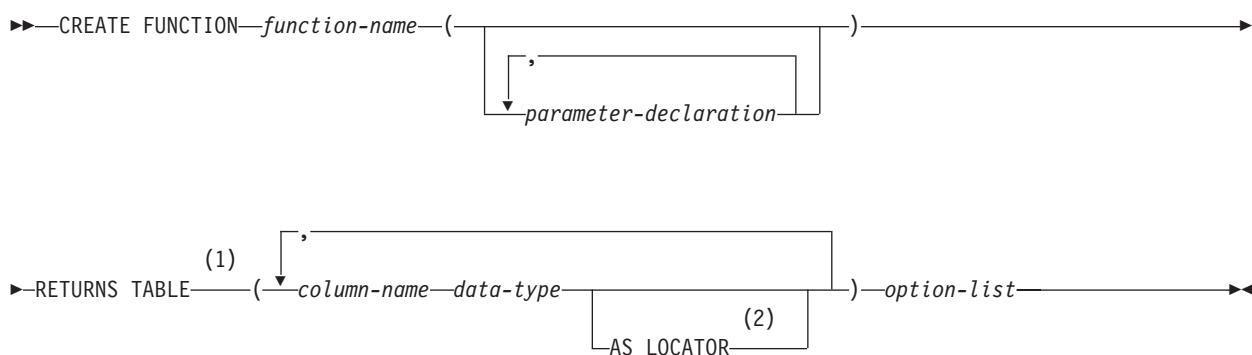
- The privilege set includes SYSADM or SYSCTRL authority.
- The SQL authorization ID of the process has the CREATEIN privilege on the schema.

Additional privileges are required if the function uses a table as a parameter, refers to a distinct type, or is to run in an MVS workload manager (WLM) environment. These privileges are:

- The SELECT privilege on any table that is an input parameter to the function.
- The USAGE privilege on each distinct type that the function references.
- Authority to create programs in the specified WLM environment. This authorization is obtained from an external security product, such as RACF.

CREATE FUNCTION (external table)

Syntax



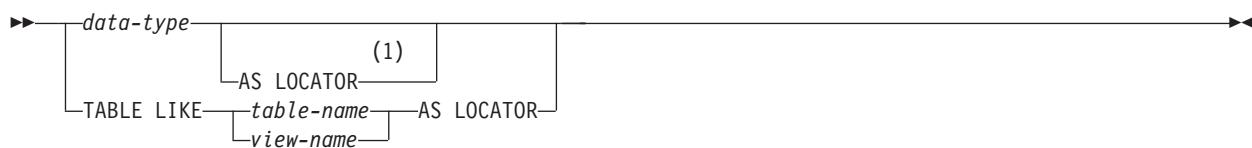
Notes:

- 1 This clause and the clauses that follow in the *option-list* can be specified in any order.
- 2 AS LOCATOR can be specified only for a LOB data type or a distinct type based on a LOB data type.

parameter-declaration:



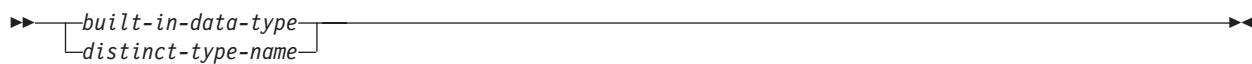
parameter-type:

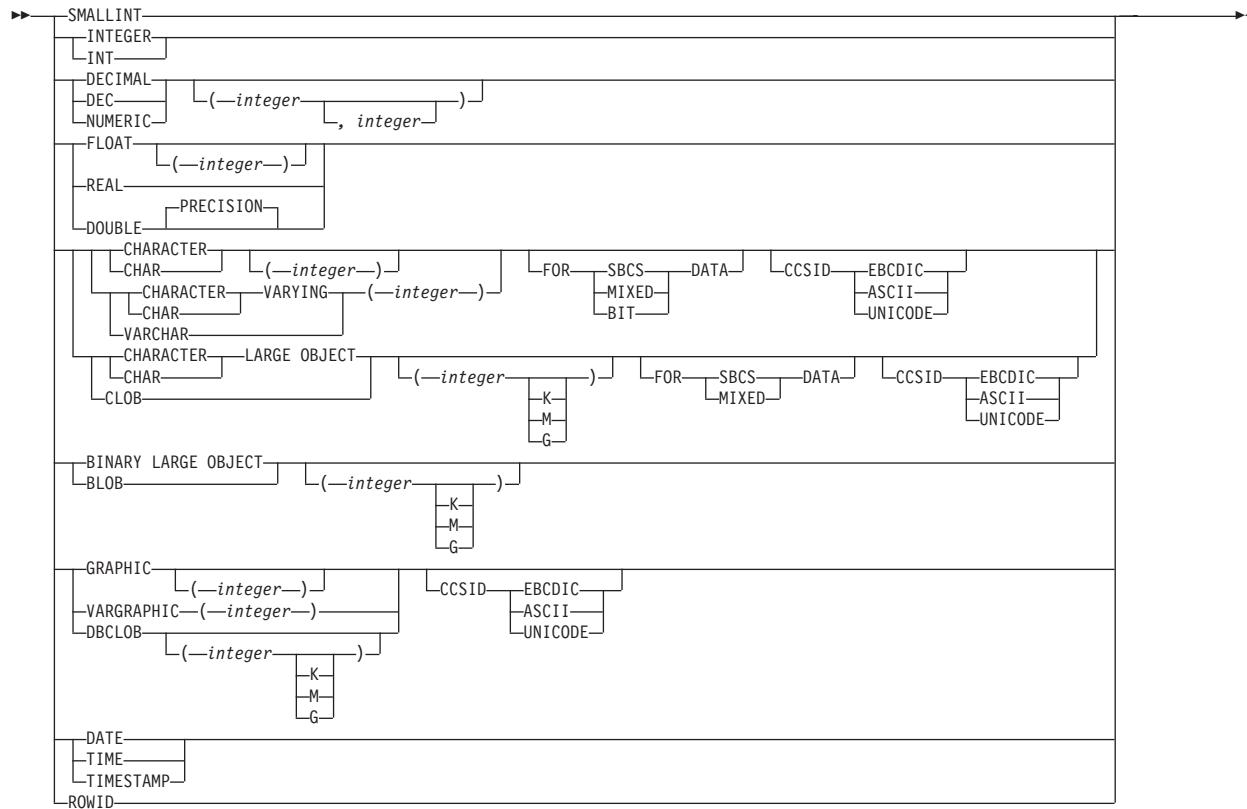


Notes:

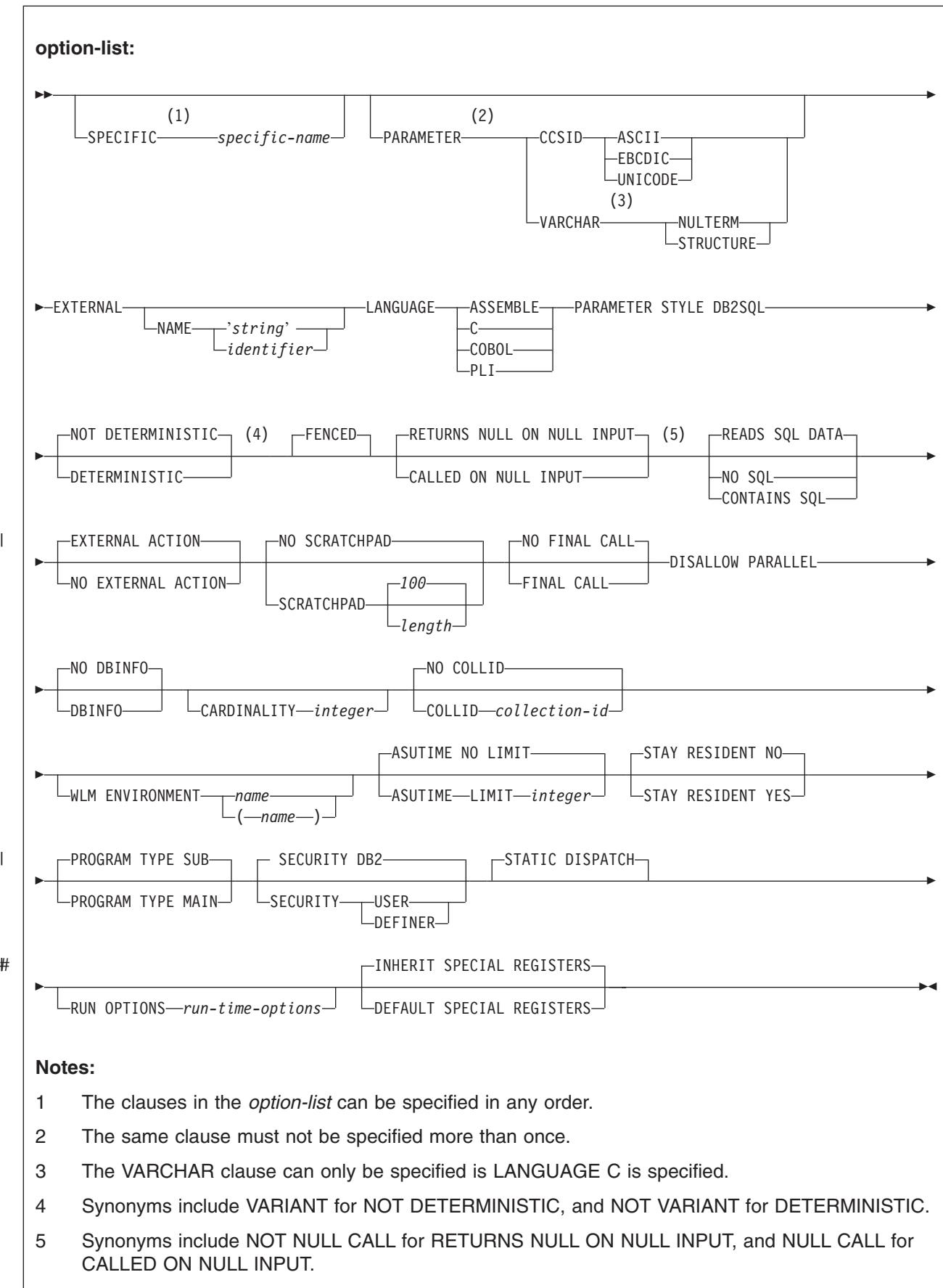
- 1 AS LOCATOR can be specified only for a LOB data type or a distinct type based on a LOB data type.

data-type:



built-in-data-type:

CREATE FUNCTION (external table)



Notes:

- 1 The clauses in the *option-list* can be specified in any order.
 - 2 The same clause must not be specified more than once.
 - 3 The VARCHAR clause can only be specified if LANGUAGE C is specified.
 - 4 Synonyms include VARIANT for NOT DETERMINISTIC, and NOT VARIANT for DETERMINISTIC.
 - 5 Synonyms include NOT NULL CALL for RETURNS NULL ON NULL INPUT, and NULL CALL for CALLED ON NULL INPUT.

Description

function-name

Names the user-defined function. The name is implicitly or explicitly qualified by a schema name. The combination of name, schema name, the number of parameters, and the data type of each parameter³⁰ (without regard for any length, precision, scale, subtype or encoding scheme attributes of the data type) must not identify a user-defined function that exists at the current server.

You can use the same name for more than one function if the function signature of each function is unique.

- The unqualified form of *function-name* is a long SQL identifier.

The name must not be any of the following system-reserved keywords even if you specify them as delimited identifiers:

ALL	LIKE	UNIQUE
AND	MATCH	UNKNOWN
ANY	NOT	=
BETWEEN	NULL	~=
DISTINCT	ONLY	<
EXCEPT	OR	≤
EXISTS	OVERLAPS	~<
FALSE	SIMILAR	>
FOR	SOME	≥
FROM	TABLE	~>
IN	TRUE	~>
IS	TYPE	

The unqualified function name is implicitly qualified with a schema name according to the following rules:

- If the statement is embedded in a program, the schema name is the authorization ID in the QUALIFIER bind option when the plan or package was created or last rebound. If QUALIFIER was not specified, the schema name is the owner of the plan or package.
- If the statement is dynamically prepared, the schema name is the SQL authorization ID in the CURRENT SQLID special register.
- The qualified form of *function-name* is a short SQL identifier (the schema name) followed by a period and a long SQL identifier.

The schema name must not begin with 'SYS' unless the schema name is 'SYSADM'.

The owner of the function is determined by how the CREATE FUNCTION statement is invoked:

- If the statement is embedded in a program, the owner is the authorization ID of the owner of the plan or package.
- If the statement is dynamically prepared, the owner is the SQL authorization ID in the CURRENT SQLID special register.

The owner is implicitly given the EXECUTE privilege with the GRANT option for the function.

(parameter-declaration,...)

#

Identifies the number of input parameters of the function, and specifies the data type of each parameter. All the parameters for a function are input parameters

30. If the function has more than 30 parameters, only the first 30 parameters are used to determine whether the function is unique.

CREATE FUNCTION (external table)

and are nullable. There must be one entry in the list for each parameter that the function expects to receive. Although not required, you can give each parameter a name.

A function can have no parameters. In this case, you must code an empty set of parentheses, for example:

```
CREATE FUNCTION WOOFER()
```

parameter-name

Specifies the name of the input parameter. The name is a long SQL identifier, and each name in the parameter list must not be the same as any other name. The same name cannot be used for a *parameter-name* and a column name.

data-type

Specifies the data type of the input parameter. The data type can be a built-in data type or a distinct type.

built-in-data-type

The data type of the input parameter is a built-in data type. You can use the same built-in data types as for the CREATE TABLE statement except LONG VARCHAR or LONG VARGRAPHIC. Use VARCHAR or VARGRAPHIC with an explicit length instead.

If you do not specify a specific value for the data types that have length, precision, or scale attributes (CHAR, GRAPHIC, DECIMAL, NUMERIC, FLOAT), the defaults are as follows:

CHAR	CHAR(1)
GRAPHIC	GRAPHIC(1)
DECIMAL	DECIMAL(5,0)
FLOAT	DOUBLE (length of 8)

For information on the data types, including the subtype of character data types (the FOR *subtype* DATA clause), see “built-in-data-type” on page 658.

For parameters with a string data type, the CCSID clause indicates whether the encoding scheme of the parameter value is ASCII, EBCDIC, or UNICODE. If you do not specify CCSID ASCII, CCSID EBCDIC, or CCSID UNICODE, the encoding scheme is the value of field DEF ENCODING SCHEME on installation panel DSNTIPF.

distinct-type-name

The data type of the input parameter is a distinct type. Any length, precision, scale, subtype, or encoding scheme attributes for the parameter are those of the source type of the distinct type.

If you specify the name of the distinct type without a schema name, DB2 resolves the schema name by searching the schemas in the SQL path.

Although parameters with a character data type have an implicitly or explicitly specified subtype (BIT, SBCS, or MIXED), the function program can receive character data of any subtype. Therefore, conversion of the input data to the subtype of the parameter might occur when the function is invoked. An error occurs if mixed data that actually contains DBCS characters is used as the value for an input parameter that is declared with an SBCS subtype.

Parameters with a datetime data type or a distinct type are passed to the function as a different data type:

- A datetime type parameter is passed as a character data type, and the data is passed in ISO format.
- The encoding scheme for a datetime type parameter is the same as the implicitly or explicitly specified encoding scheme of any character or graphic string parameters. If no character or graphic string parameters are passed, the encoding scheme is the value of field DEF ENCODING SCHEME on installation panel DSNTIPF.
- A distinct type parameter is passed as the source type of the distinct type.

AS LOCATOR

Specifies that a locator to the value of the parameter is passed to the function instead of the actual value. Specify AS LOCATOR only for parameters with a LOB data type or a distinct type that is based on a LOB data type. Passing locators instead of values can result in fewer bytes being passed to the function, especially when the value of the parameter is very large.

The AS LOCATOR clause has no effect on determining whether data types can be promoted, nor does it affect the function signature, which is used in function resolution.

TABLE LIKE *table-name* or *view-name* AS LOCATOR

Specifies that the parameter is a transition table. However, when the function is invoked, the actual values in the transition table are not passed to the function. A single value is passed instead. This single value is a locator to the table, which the function uses to access the columns of the transition table. A function with a table parameter can only be invoked from the triggered action of a trigger.

The use of TABLE LIKE provides an implicit definition of the transition table. It specifies that the transition table has the same number of columns as the identified table or view. The columns have the same data type, length, precision, scale, subtype, and encoding scheme as the identified table or view, as they are described in catalog tables SYSCOLUMNS and SYSTABLESPACE.

The *name* specified after TABLE LIKE must identify a table or view that exists at the current server. The name must not identify a declared temporary table. The name does not have to be the same name as the table that is associated with the transition table for the trigger. An unqualified table or view name is implicitly qualified according to the following rules:

- If the CREATE FUNCTION statement is embedded in a program, the implicit qualifier is the authorization ID in the QUALIFIER bind option when the plan or package was created or last rebound. If QUALIFIER was not used, the implicit qualifier is the owner of the plan or package.
- If the CREATE FUNCTION statement is dynamically prepared, the implicit qualifier is the SQL authorization ID in the CURRENT_SQLID special register.

When the function is invoked, the corresponding columns of the transition table identified by the table locator and the table or view identified in the TABLE LIKE clause must have the same definition. The data type, length, precision, scale, and encoding scheme of these columns must match

CREATE FUNCTION (external table)

exactly. The description of the table or view at the time the CREATE FUNCTION statement was executed is used.

Additionally, a character FOR BIT DATA column of the transition table cannot be passed as input for a table parameter for which the corresponding column of the table specified at the definition is not defined as character FOR BIT DATA. (The definition occurs with the CREATE FUNCTION statement.) Likewise, a character column of the transition table that is not FOR BIT DATA cannot be passed as input for a table parameter for which the corresponding column of the table specified at the definition is defined as character FOR BIT DATA.

For more information about using table locators, see *DB2 Application Programming and SQL Guide*.

RETURNS TABLE(*column-name data-type ...*)

Identifies that the output of the function is a table. The parentheses that follow the keyword enclose the list of names and data types of the columns of the table.

column-name

Specifies the name of the column. The name is a long identifier and must be unique within the RETURNS TABLE clause for the function.

data-type

Specifies the data type of the column. The column is nullable. The data type can be any built-in data type, except LONG VARCHAR or LONG VARGRAPHIC. The data type can also be any distinct type.

AS LOCATOR

Specifies that the function returns a locator to the value rather than the actual value. You can specify AS LOCATOR only for a LOB data type or a distinct type based on a LOB data type.

SPECIFIC *specific-name*

Specifies a unique name for the function. The name is implicitly or explicitly qualified with a schema name. The name, including the schema name, must not identify the specific name of another function that exists at the current server.

The unqualified form of *specific-name* is a long SQL identifier. The qualified form is a short SQL identifier (the schema name) followed by a period and a long SQL identifier.

If you do not specify a schema name, it is the same as the explicit or implicit schema name of the function name (*function-name*). If you specify a schema name, it must be the same as the explicit or implicit schema name of the function name.

If you do not specify the SPECIFIC clause, the default specific name is the name of the function. However, if the function name does not provide a unique specific name or if the function name is a single asterisk, DB2 generates a specific name in the form of:

SQLxxxxxxxxxxxx

where 'xxxxxxxxxxxx' is a string of 12 characters that make the name unique.

The specific name is stored in the SPECIFIC column of the SYSROUTINES catalog table. The specific name can be used to uniquely identify the function in several SQL statements (such as ALTER FUNCTION, COMMENT ON, DROP,

GRANT, and REVOKE) and in DB2 commands (START FUNCTION, STOP FUNCTION, and DISPLAY FUNCTION). However, the function cannot be invoked by its specific name.

PARAMETER CCSID or VARCHAR

CCSID

Indicates whether the encoding scheme for string parameters is ASCII, EBCDIC, or UNICODE. The default encoding scheme is the value specified in the CCSID clauses of the parameter list or RETURNS TABLE clause, or in the field DEF ENCODING SCHEME on installation panel DSNTIPF.

This clause provides a convenient way to specify the encoding scheme for *all* string parameters. If individual CCSID clauses are specified for individual parameters in addition to this PARAMETER CCSID clause, the value specified in *all* of the CCSID clauses must be the same value that is specified in this clause.

This clause also specifies the encoding scheme to be used for system-generated parameters of the routine such as message tokens and DBINFO.

VARCHAR

Specifies that the representation of the values of varying length character string-parameters, including, if applicable, the output of the function, for functions which specify LANGUAGE C.

This option can only be specified if LANGUAGE C is also specified or SQLCODE -628, SQLSTATE 42613 is returned.

NULTERM

Specifies that variable length character string parameters are represented in a NUL-terminated string form.

STRUCTURE

Specifies that variable length character string parameters are represented in a VARCHAR structure form.

Using the PARAMETER VARCHAR clause, there is no way to specify the VARCHAR form of an individual parameter as these is with PARAMETER CCSID. The PARAMETER VARCHAR clause only applies to parameters in a function's parameter list and in the RETURNS TABLE clause. It does not apply to system-generated parameters of the routine such as message tokens and DBINFO.

In a data sharing environment, you should not specify the PARAMETER VARCHAR clause until all members of the data sharing group support the clause. If some group members support this clause and others do not, and PARAMETER VARCHAR is specified in an external routine, the routine will encounter different parameter forms depending on which group member invokes the routine.

EXTERNAL

Specifies that the function being registered is based on code that is written in an external programming language and adheres to the documented linkage conventions and interface of that language.

If you do not specify the NAME clause, 'NAME *function-name*' is implicit. In this case, *function-name* must not be longer than 8 characters.

CREATE FUNCTION (external table)

NAME '*string*' or *identifier*

Identifies the name of the MVS load module that contains the user-written code that implements the logic of the function.

For other values of LANGUAGE, the name can be a string constant that is no longer than 8 characters or a short identifier. It must conform to the naming conventions for MVS load modules. Alphabetical extenders for national languages can be used as the first character and as subsequent characters in the load module name.

DB2 loads the load module when the function is invoked. The load module is created when the program that contains the function body is compiled and link-edited. The load module does not need to exist when the CREATE FUNCTION statement is executed. However, it must exist and be accessible by the current server when the function is invoked.

You can specify the EXTERNAL clause in one of the following ways:

EXTERNAL

EXTERNAL NAME PKJVSP1

EXTERNAL NAME 'PKJVSP1'

If you specify an external program name, you must use the NAME keyword. For example, this syntax is not valid:

EXTERNAL PKJVSP1

LANGUAGE

Specifies the application programming language in which the function program is written. All programs must be designed to run in IBM's Language Environment environment.

ASSEMBLE

The function is written in Assembler.

C The function is written in C or C++.

COBOL

The function is written in COBOL, including the object-oriented language extensions.

PLI

The function is written in PL/I.

PARAMETER STYLE DB2SQL

Specifies the linkage convention that the function program uses to receive input parameters from and pass return values to the invoking SQL statement.

DB2SQL indicates that parameters for indicator variables are associated with each input and return value to allow for null values. The parameters that are passed between the invoking SQL statement and the function include:

- The first *n* parameters are the input parameters that are specified for the function
- The next *m* parameters are the result columns of the function that are specified on the RETURNS TABLE clause
- *n* parameters for the indicator variables for the input parameters
- *m* parameters for the indicator variables of the result columns of the function that are specified on the RETURNS TABLE clause
- The SQLSTATE to be returned to DB2
- The qualified name of the function

#

- The specific name of the function
- The SQL diagnostic string to be returned to DB2
- The scratchpad, if SCRATCHPAD is specified
- The call type
- The DBINFO structure, if DBINFO is specified

For complete details about the structure of the parameter list that is passed, see *DB2 Application Programming and SQL Guide*.

NOT DETERMINISTIC or DETERMINISTIC

Specifies whether the function returns the same results for identical input arguments.

NOT DETERMINISTIC

The function might not return the same result for identical input arguments. The function depends on some state values that affect the results. DB2 uses this information to disable the merging of views and table expressions when processing SELECT, UPDATE, DELETE, or INSERT statements that reference this function. An example of a function that is not deterministic is one that generates random numbers, or any function that contains SQL statements.

NOT DETERMINISTIC is the default.

Some functions that are not deterministic can receive incorrect results if the function is executed by parallel tasks. Specify the DISALLOW PARALLEL clause for these functions.

DETERMINISTIC

The function always returns the same result for identical input arguments.

An example of a deterministic function is a function that calculates the square root of the input. DB2 uses this information to enable the merging of views and table expressions for SELECT, UPDATE, DELETE, or INSERT statements that reference this function. DETERMINISTIC is not the default. If applicable, specify DETERMINISTIC to prevent non-optimal access paths from being chosen for SQL statements that reference this function.

#

DB2 does not verify that the function program is consistent with the specification of DETERMINISTIC or NOT DETERMINISTIC.

FENCED

Specifies that the function runs in an external address space to prevent the function from corrupting DB2 storage.

FENCED is the default.

RETURNS NULL ON NULL INPUT or CALLED ON NULL INPUT

Specifies whether the function is called if any of the input arguments is null at execution time.

RETURNS NULL ON NULL INPUT

The function is not called if any of the input arguments is null. The result is an empty table, which is a table with no rows. RETURNS NULL ON INPUT is the default.

CALLED ON NULL INPUT

The function is called regardless of whether any of the input arguments is null, making the function responsible for testing for null argument values. The function can return an empty table, depending on its logic.

NO SQL, READS SQL DATA, or CONTAINS SQL

Indicates whether the function can execute any SQL statements and, if so, what

CREATE FUNCTION (external table)

type. DB2 verifies that the SQL issued by the function is consistent with this specification. See Table 72 on page 972 for a detailed list of the SQL statements that can be executed under each data access indication.

NO SQL

The function does not execute SQL statements.

READS SQL DATA

The function does not execute SQL statements that modify data. SQL statements that are not supported in any function return a different error.

READS SQL DATA is the default.

CONTAINS SQL

The function does not execute SQL statements that read or modify data. SQL statements that are not supported in any function return a different error.

EXTERNAL ACTION or NO EXTERNAL ACTION

Specifies whether the function takes an action that changes the state of an object that DB2 does not manage. An example of an external action is sending a message or writing a record to a file.

Because DB2 uses the RRS attachment for functions, DB2 can participate in two-phase commit with any other resource manager that uses RRS. For resource managers that do not use RRS, there is no coordination of commit or rollback operations on non-DB2 resources.

EXTERNAL ACTION

The function can take an action that changes the state of an object that DB2 does not manage.

Some SQL statements that invoke functions with external actions can result in incorrect results if parallel tasks execute the function. For example, if the function sends a note for each initial call to it, one note is sent for each parallel task instead of once for the function. Specify the DISALLOW PARALLEL clause for functions that do not work correctly with parallelism.

If you specify EXTERNAL ACTION, DB2:

- Materializes the views and table expressions in SELECT, UPDATE, DELETE or INSERT statements that refer to the function. This materialization can adversely affect the access paths that are chosen for the SQL statements that reference this function. Do not specify EXTERNAL ACTION if the function does not have an external action.
- Does not move the function from one task control block (TCB) to another between FETCH operations.
- Does not allow another function or stored procedure to use the TCB until the cursor is closed. This is also applicable for cursors declared WITH HOLD.

The only changes to resources made outside of DB2 that are under the control of commit and rollback operations are those changes made under RRS control.

EXTERNAL ACTION is the default.

NO EXTERNAL ACTION

The function does not take any action that changes the state of an object that DB2 does not manage. DB2 uses this information to enable the merging of views and table expressions for SELECT, UPDATE, DELETE, or

```
#  
#  
#  
#
```

```
#  
#  
#  
#
```

INSERT statements that reference this function. NO EXTERNAL ACTION is not the default. If applicable, specify NO EXTERNAL ACTION to prevent non-optimal access paths from being chosen for SQL statements that reference this function.

NO SCRATCHPAD or SCRATCHPAD

Specifies whether DB2 is to provide a scratchpad for the function. It is strongly recommended that functions be reentrant, and a scratchpad provides an area for the function to save information from one invocation to the next.

NO SCRATCHPAD

A scratchpad is not allocated and passed to the function. NO SCRATCHPAD is the default.

SCRATCHPAD *length*

When the function is invoked for the first time, DB2 allocates memory for a scratchpad. A scratchpad has the following characteristics:

- *length* must be between 1 and 32767. The default value is 100 bytes.
- DB2 initializes the scratchpad to all binary zeros (X'00"s).
- The scope of a scratchpad is the SQL statement. For each reference to the function in an SQL statement, there is one scratchpad. For example, assuming that function UDFX was defined with the SCRATCHPAD keyword, three scratchpads are allocated for the three references to UDFX in the following SQL statement:

```
SELECT A, UDFX(A) FROM TABLEB  
WHERE UDFX(A) > 103 OR UDFX(A) < 19;
```

If the function is run under parallel tasks, one scratchpad is allocated for each parallel task of each reference to the function in the SQL statement. This can lead to unpredictable results. For example, if a function uses the scratchpad to count the number of times that it is invoked, the count reflects the number of invocations done by the parallel task and not the SQL statement. Specify the DISALLOW PARALLEL clause for functions that will not work correctly with parallelism.

- The scratchpad is persistent. DB2 preserves its content from one invocation of the function to the next. Any changes that the function makes to the scratchpad on one call are still there on the next call. DB2 initializes the scratchpads when it begins to execute an SQL statement. DB2 does not reset scratchpads when a correlated subquery begins to execute.
- The scratchpad can be a central point for the system resources that the function acquires. If the function acquires system resources, specify FINAL CALL to ensure that DB2 calls the function one more time so that the function can free those system resources.

Each time the function invoked, DB2 passes an additional argument to the function that contains the address of the scratchpad.

If you specify SCRATCHPAD, DB2:

- Does not move the function from one task control block (TCB) to another between FETCH operations.
- Does not allow another function or stored procedure to use the TCB until the cursor is closed. This is also applicable for cursors declared WITH HOLD.

NO FINAL CALL or FINAL CALL

Specifies whether a *first call* and a *final call* are made to the function.

CREATE FUNCTION (external table)

NO FINAL CALL

A first call and final call are not made to the function. NO FINAL CALL is the default.

FINAL CALL

A first call and final call are made to the function in addition to one or more *open*, *fetch*, or *close calls*.

The types of calls are:

First call

A *first call* occurs only if the function was defined with FINAL CALL.

Before a first call, the scratchpad is set to binary zeros. Argument values are passed to the function, and the function might acquire memory or perform other one-time only resource initialization. However, the function should not return any data to DB2, but it can set return values for the SQL-state and diagnostic-message arguments.

Open call

An *open call* occurs unless the function returns an error. The scratchpad is set to binary zeros only if the function was defined with NO FINAL CALL. Argument values are passed to the function, and the function might perform any one-time initialization actions that are required. However, the function should not return any data to DB2.

Fetch call

A *fetch call* occurs unless the function returns an error during the first call or open call. Argument values are passed to the function, and DB2 expects the function to return a row of data or the end-of-table condition. If a scratchpad is also passed to the function, it remains untouched from the previous call.

Close call

A *close call* occurs unless the function returns an error during the first call, open call, or fetch call. No SQL-argument or SQL-argument-ind values are passed to the function, and if the function attempts to examine these values, unpredictable results may occur. If a scratchpad is also passed to the function, it remains untouched from the previous call.

The function should not return any data to DB2, but it can set return values for the SQL-state and diagnostic-message arguments. Also on close call, a function that is defined with NO FINAL CALL should release any system resources that it acquired. (A function that is defined with FINAL CALL should release any acquired resources on the final call.)

Final

The *final call* balances the first call, and like the first call, occurs only if the function was defined with FINAL CALL. The function can set return values for the SQL-state and diagnostic-message arguments. The function should also release any system resources that it acquired. A final call occurs at these times:

- *End of statement*: When the cursor is closed for cursor-oriented statements, or the execution of the statement has completed.
- *End of transaction*: When normal end of statement processing does not occur. For example, the logic of an application, for some reason, bypasses closing the cursor.

If a commit operation occurs while a cursor defined as WITH HOLD is open, a final call is made when the cursor is closed or the application ends. If a commit occurs at the end of a parallel task, a final call is made regardless of whether a cursor defined as WITH HOLD is open.

If a commit, rollback, or abort operation causes the final call, the function cannot issue any SQL statements when it is invoked.

DISALLOW PARALLEL

Specifies that DB2 does not consider parallelism for the function.

NO DBINFO or DBINFO

Specifies whether specific information that is known by DB2 is passed to the function when it is invoked.

NO DBINFO

No additional information is passed. NO DBINFO is the default.

DBINFO

An additional argument is passed when the function is invoked. The argument is a structure that contains information such as the application run-time authorization ID, the schema name, the name of a table or column that the function might be inserting into or updating, and identification of the database server that invoked the function. For details about the argument and its structure, see *DB2 Application Programming and SQL Guide*.

CARDINALITY *integer*

Specifies an estimate of the expected number of rows that the function returns. The number is used for optimization purposes. The value of *integer* must range from 0 to 2147483647.

If you do not specify CARDINALITY, DB2 assumes a finite value. The finite value is the same value that DB2 assumes for tables for which the RUNSTATS utility has not gathered statistics.

If a function has an infinite cardinality—the function never returns the “end-of-table” condition and always returns a row, then a query that requires the “end-of-table” to work correctly will need to be interrupted. Thus, avoid using such functions in queries that involve GROUP BY and ORDER BY.

NO COLLID or COLLID *collection-id*

Identifies the package collection that is to be used when the function is executed. This is the package collection into which the DBRM that is associated with the function program is bound.

NO COLLID

The package collection for the function is the same as the package collection of the program that invokes the function. If a trigger invokes the function, the collection of the trigger package is used. If the invoking program does not use a package, the package collection is the value of the CURRENT PACKAGESET special register.

NO COLLID is the default.

COLLID *collection-id*

The name of the package collection that is to be used when the external is executed.

WLM ENVIRONMENT

Identifies the MVS workload manager (WLM) application environment in which the function is to run. The *name* of the WLM environment is a long identifier.

CREATE FUNCTION (external table)

If you do not specify WLM ENVIRONMENT, the function runs in the WLM-established stored procedure address space that is specified at installation time.

name

The WLM environment in which the function must run. If another user-defined function or a stored procedure calls the function and that calling routine is running in an address space that is not associated with the WLM environment, DB2 routes the function request to a different MVS address space.

*name,**

When an SQL application program directly invokes the function, the WLM environment in which the function runs.

If another user-defined function or a stored procedure calls the function, the function runs in same environment that the calling routine uses. In this case, authorization to run the function in the WLM environment is not checked because the authorization of the calling routine suffices.

Users must have the appropriate authorization to execute functions in the specified WLM environment. For an example of a RACF command that provides this authorization, see “Running external functions in WLM environments” on page 551.

ASUTIME

Specifies the total amount of processor time, in CPU service units, that a single invocation of the function can run. The value is unrelated to the ASUTIME column of the resource limit specification table.

When you are debugging a function, setting a limit can be helpful if the function gets caught in a loop. For information on service units, see *OS/390 MVS Initialization and Tuning Guide*.

NO LIMIT

There is no limit on the service units. NO LIMIT is the default.

LIMIT *integer*

The limit on the service units is a positive integer in the range of 1 to 2G. If the function uses more service units than the specified value, DB2 cancels the function.

STAY RESIDENT

Specifies whether the load module for the function is to remain resident in memory when the function ends.

NO

The load module is deleted from memory after the function ends. Use NO for non-reentrant functions. NO is the default.

YES

The load module remains resident in memory after the function ends. Use YES for reentrant functions.

PROGRAM TYPE

Specifies whether the function program runs as a main routine or a subroutine.

SUB

The function runs as a subroutine. SUB is the default.

MAIN

The function runs as a main routine.

SECURITY

Specifies how the function interacts with an external security product, such as RACF, to control access to non-SQL resources.

DB2

The function does not require an external security environment. If the function accesses resources that an external security product protects, the access is performed using the authorization ID that is associated with the WLM-established stored procedure address space.

DB2 is the default.

USER

An external security environment should be established for the function. If the function accesses resources that the external security product protects, the access is performed using the primary authorization ID of the process that invoked the function.

DEFINER

An external security environment should be established for the function. If the function accesses resources that the external security product protects, the access is performed using the authorization ID of the owner of the function.

STATIC DISPATCH

At function resolution time, DB2 chooses a function based on the static (or declared) types of the function parameters. STATIC DISPATCH is the default.

RUN OPTIONS *run-time-options*

Specifies the Language Environment run-time options to be used for the function. You must specify *run-time-options* as a character string that is no longer than 254 bytes. If you do not specify RUN OPTIONS or pass an empty string, DB2 does not pass any run-time options to Language Environment, and Language Environment uses its installation defaults.

For a description of the Language Environment run-time options, see *OS/390 Language Environment for OS/390 & VM Programming Reference*.

INHERIT SPECIAL REGISTERS

Indicates that the values of special registers are inherited according to the rules listed in the table for characteristics of special registers in a user-defined function in Table 19 on page 93.

DEFAULT SPECIAL REGISTERS

Indicates that special registers are initialized to the default values, as indicated by the rules in the table for characteristics of special registers in a user-defined function in Table 19 on page 93.

Notes

See “Notes” on page 549 for information about:

- Choosing data types for parameters
- Specifying the encoding scheme for parameters
- Determining the uniqueness of functions in a schema
- Character string representation considerations
- Overriding a built-in function
- Running external functions in WLM environments
- Specifying non-deterministic or external action functions in the definition of a scrollable cursor

#

CREATE FUNCTION (external table)

Example

The following example registers a table function written to return a row consisting of a single document identifier column for each known document in a text management system. The first parameter matches a given subject area and the second parameter contains a given string.

Within the context of a single session, the table function always returns the same table; therefore, it is defined as DETERMINISTIC. In addition, the DISALLOW PARALLEL keyword is added because table functions cannot operate in parallel.

Although the size of the output for DOCMATCH is highly variable, CARDINALITY 20 is a representative value and is specified to help DB2.

```
CREATE FUNCTION DOCMATCH (VARCHAR(30), VARCHAR(255))
    RETURNS TABLE (DOC_ID CHAR(16))
    EXTERNAL NAME ABC
    LANGUAGE C
    PARAMETER STYLE DB2SQL
    NO SQL
    DETERMINISTIC
    NO EXTERNAL ACTION
    FENCED
    SCRATCHPAD
    FINAL CALL
    DISALLOW PARALLEL
    CARDINALITY 20;
```

CREATE FUNCTION (sourced)

This CREATE FUNCTION statement registers a user-defined function that is based on an existing scalar or column function with a database server.

Invocation

This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is implicitly or explicitly specified.

Authorization

The privilege set defined below must include at least one of the following:

- The CREATEIN privilege for the schema or all schemas
- SYSADM or SYSCTRL authority

The authorization ID that matches the schema name implicitly has the CREATEIN privilege on the schema.

Privilege set: If the statement is embedded in an application program, the privilege set is the privileges that are held by the authorization ID of the owner of the plan or package.

If the statement is dynamically prepared, the privilege set is the privileges that are held by the SQL authorization ID of the process. The specified function name can include a schema name (a qualifier). However, if the schema name is not the same as the SQL authorization ID, one of the following conditions must be met:

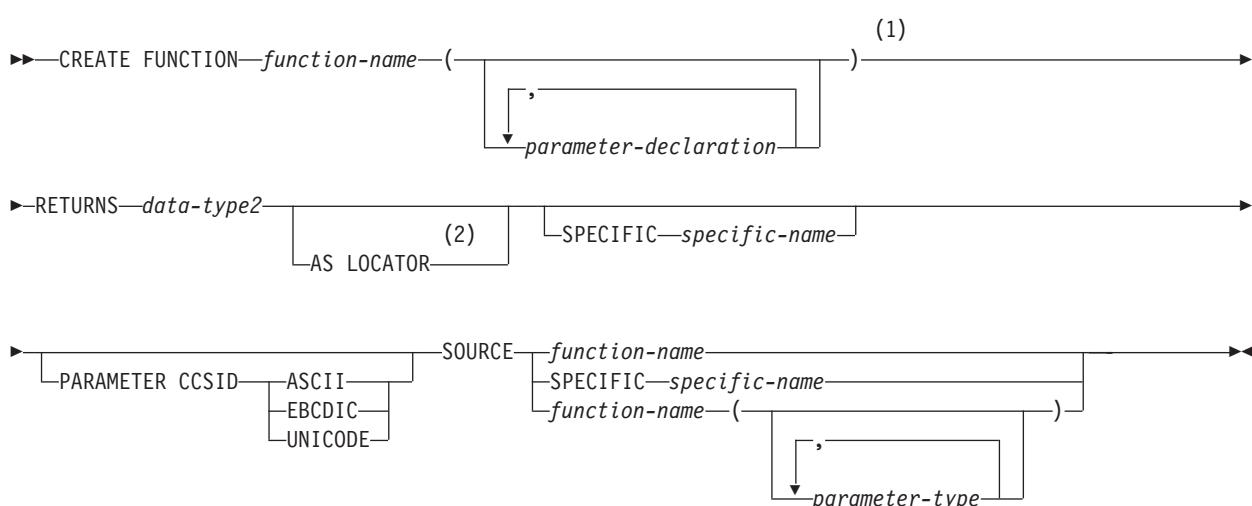
- The privilege set includes SYSADM or SYSCTRL authority.
- The SQL authorization ID of the process has the CREATEIN privilege on the schema.

Additional privileges are required for the source function, and other privileges are also needed if the function uses a table as a parameter, or refers to a distinct type. These privileges are:

- The EXECUTE privilege for the function that the SOURCE clause references.
- The SELECT privilege on any table that is an input parameter to the function.
- The USAGE privilege on each distinct type that the function references.

CREATE FUNCTION (sourced)

Syntax



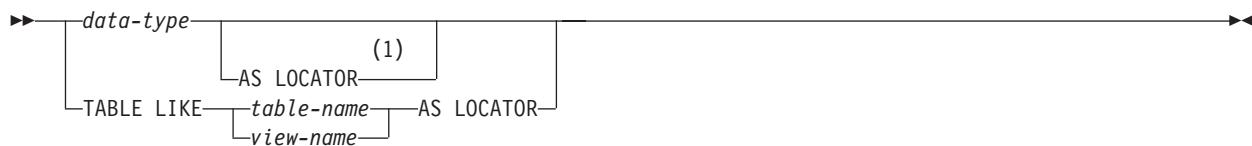
Notes:

- 1 RETURNS, SPECIFIC, and SOURCE can be specified in any order.
 - 2 AS LOCATOR can be specified only for a LOB data type or a distinct type based on a LOB data type.

parameter-declaration:



parameter-type:



Notes:

- 1 AS LOCATOR can be specified only for a LOB data type or a distinct type based on a LOB data type.

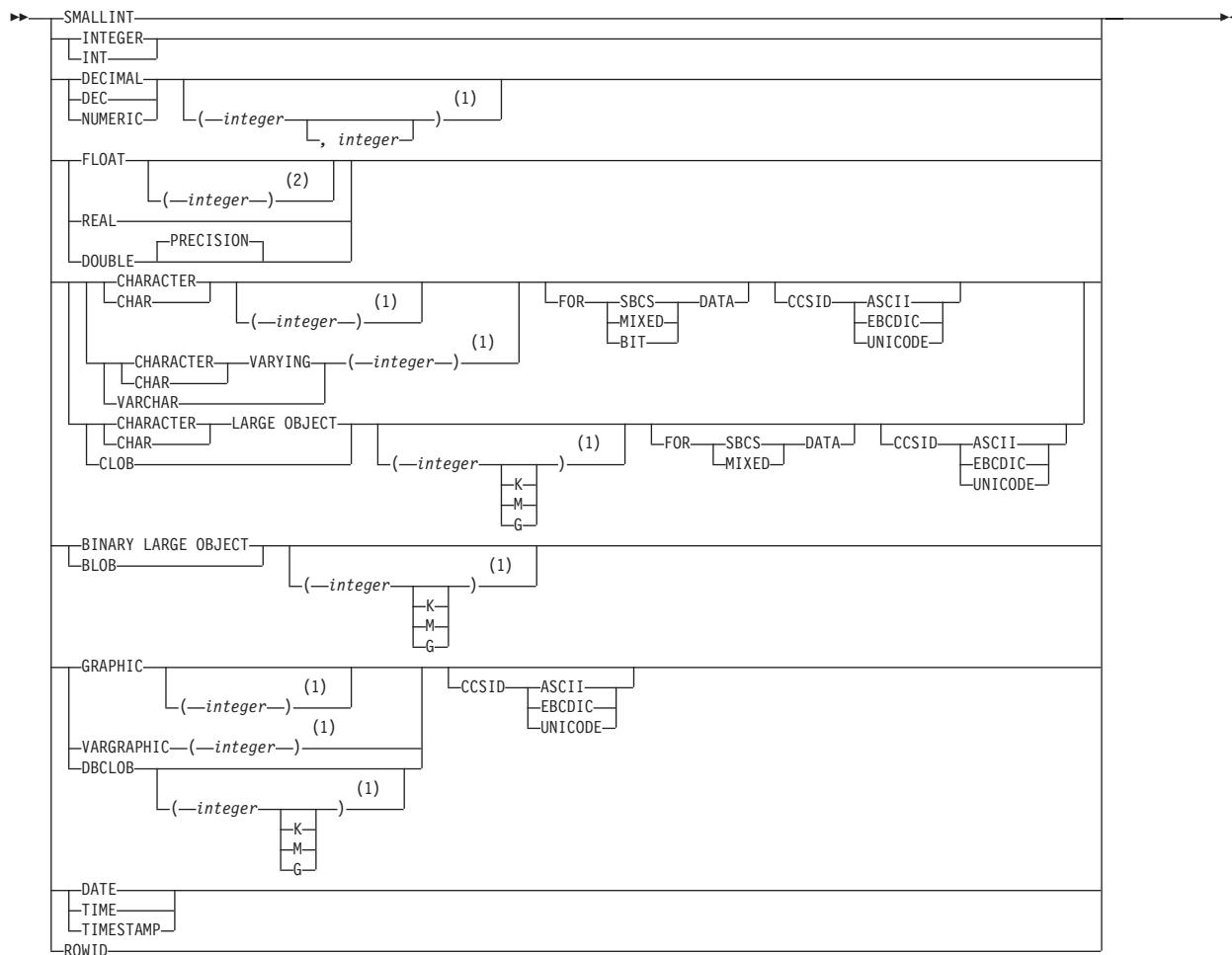
data-type:

►►—*built-in-data-type*—►►

 └—*distinct-type-name*—┘

CREATE FUNCTION (sourced)

built-in-data-type:



Notes:

- 1 Coding specific values for the length, precision, or scale attributes of a data type for a parameter in the SOURCE clause is optional. Empty parentheses, (), can be used instead to indicate that DB2 is to ignore the attributes when determining whether data types match. However, if the length, precision, or scale attributes are specified, the value must exactly match the value that was specified when the source function was created.
- 2 Coding a specific value is optional. If a value is specified, it does not have to match the value that was specified when the source function was created because matching is based on data type (REAL or DOUBLE). $1 \leq \text{integer} \leq 21$ indicates REAL and $22 \leq \text{integer} \leq 53$ indicates DOUBLE. Empty parentheses cannot be used.

Description

function-name

Names the user-defined function. The name is implicitly or explicitly qualified by a schema name. The combination of name, schema name, the number of

parameters, and the data type each parameter³¹ (without regard for any length, precision, scale, subtype, or encoding scheme attributes of the data type) must not identify a user-defined function that exists at the current server.

If the function is sourced on an existing function to enable the use of the existing function with a distinct type, the name can be the same name as the existing function. In general, more than one function can have the same name if the function signature of each function is unique.

- The unqualified form of *function-name* is a long SQL identifier.

The name must not be any of the following system-reserved keywords even if you specify them as delimited identifiers:

ALL	LIKE	UNIQUE
AND	MATCH	UNKNOWN
ANY	NOT	=
BETWEEN	NULL	~=
DISTINCT	ONLY	<
EXCEPT	OR	~=
EXISTS	OVERLAPS	<~
FALSE	SIMILAR	>
FOR	SOME	~=
FROM	TABLE	~>
IN	TRUE	<>
IS	TYPE	

The unqualified function name is implicitly qualified with a schema name according to the following rules:

- If the statement is embedded in a program, the schema name is the authorization ID in the QUALIFIER bind option when the plan or package was created or last rebound. If QUALIFIER was not specified, the schema name is the owner of the plan or package.
- If the statement is dynamically prepared, the schema name is the SQL authorization ID in the CURRENT SQLID special register.
- The qualified form of *function-name* is a short SQL identifier (the schema name) followed by a period and a long SQL identifier.

The schema name must not begin with 'SYS' unless the schema name is 'SYSADM'.

The owner of the function is determined by how the CREATE FUNCTION statement is invoked:

- If the statement is embedded in a program, the owner is the authorization ID of the owner of the plan or package.
- If the statement is dynamically prepared, the owner is the SQL authorization ID in the CURRENT SQLID special register.

The owner is implicitly given the EXECUTE privilege with the GRANT option for the function.

(*parameter-declaration*,...)

Specifies the number of input parameters of the function and the data type of each parameter. All the parameters for a function are input parameters and are nullable. There must be one entry in the list for each parameter that the function expects to receive. Although not required, you can give each parameter a name.

31. If the function has more than 30 parameters, only the first 30 parameters are used to determine whether the function is unique.

CREATE FUNCTION (sourced)

A function can have no parameters. In this case, you must code an empty set of parentheses, for example:

```
CREATE FUNCTION WOOFER()
```

parameter-name

Specifies the name of the input parameter. The name is a long SQL identifier, and each name in the parameter list must not be the same as any other name.

data-type

Specifies the data type of the input parameter. The data type can be a built-in data type or a distinct type.

built-in-data-type

The data type of the input parameter is a built-in data type. You can use the same built-in data types as for the CREATE TABLE statement except LONG VARCHAR or LONG VARGRAPHIC. Use VARCHAR or VARGRAPHIC with an explicit length instead.

If you do not specify a specific value for the data types that have length, precision, or scale attributes (CHAR, GRAPHIC, DECIMAL, NUMERIC, FLOAT), the defaults are as follows:

CHAR	CHAR(1)
GRAPHIC	GRAPHIC(1)
DECIMAL	DECIMAL(5,0)
FLOAT	DOUBLE (length of 8)

For information on the data types, including the subtype of character data types (the FOR *subtype* DATA clause), see “built-in-data-type” on page 658.

For parameters with a string data type, the CCSID clause indicates whether the encoding scheme of the parameter value is ASCII, EBCDIC, or UNICODE. If you do not specify CCSID ASCII, CCSID EBCDIC, or CCSID UNICODE, the encoding scheme is the value of field DEF ENCODING SCHEME on installation panel DSNTIPF.

distinct-type-name

The data type of the input parameter is a distinct type. Any length, precision, scale, subtype, or encoding scheme attributes for the parameter are those of the source type of the distinct type.

The implicitly or explicitly specified encoding scheme of all the parameters with a string data type must be the same—either all ASCII, all EBCDIC, or all UNICODE.

Although parameters with a character data type have an implicitly or explicitly specified subtype (BIT, SBCS, or MIXED), the function program can receive character data of any subtype. Therefore, conversion of the input data to the subtype of the parameter might occur when the function is invoked.

Parameters with a datetime data type or a distinct type are passed to the function as a different data type:

- A datetime type parameter is passed as a character data type, and the data is passed in ISO format.

The encoding scheme for a datetime type parameter is the same as the implicitly or explicitly specified encoding scheme of any character or

graphic string parameters. If no character or graphic string parameters are passed, the encoding scheme is the value of field DEF ENCODING SCHEME on installation panel DSNTIPF.

- A distinct type parameter is passed as the source type of the distinct type.

You can specify any built-in data type or distinct type that matches or can be cast to the data type of the corresponding parameter of the source function (the function that is identified in the SOURCE clause). (For information on casting data types, see “Casting between data types” on page 62.) Length, precision, or scale attributes do not have to be specified for data types with these attributes. When specifying data types with these attributes, follow these rules:

- An empty set of parentheses can be used to indicate that the length, precision, or scale is the same as the source function.
- If length, precision, or scale is not explicitly specified, and empty parentheses are not specified, the default length of the data type is implied. For example:

CHAR	CHAR(1)
GRAPHIC	GRAPHIC(1)
DECIMAL	DECIMAL(5,0)
FLOAT	DOUBLE (length of 8)

For more information on default lengths of data types, see “built-in-data-type” on page 658.

AS LOCATOR

Specifies that a locator to the value of the parameter is passed to the function instead of the actual value. Specify AS LOCATOR only for parameters with a LOB data type or a distinct type based on a LOB data type. Passing locators instead of values can result in fewer bytes being passed to the function, especially when the value of the parameter is very large.

The AS LOCATOR clause has no effect on determining whether data types can be promoted, nor does it affect the function signature, which is used in function resolution.

TABLE LIKE *table-name* or *view-name* AS LOCATOR

Specifies that the parameter is a transition table. However, when the function is invoked, the actual values in the transition table are not passed to the function. A single value is passed instead. This single value is a locator to the table, which the function uses to access the columns of the transition table. A function with a table parameter can only be invoked from the triggered action of a trigger.

The use of TABLE LIKE provides an implicit definition of the transition table. It specifies that the transition table has the same number of columns as the identified table or view. The columns have the same data type, length, precision, scale, subtype, and encoding scheme as the identified table or view, as they are described in catalog tables SYSCOLUMNS and SYSTABLESPACE.

The *name* specified after TABLE LIKE must identify a table or view that exists at the current server. The name must not identify a declared temporary table. The name does not have to be the same name as the

CREATE FUNCTION (sourced)

table that is associated with the transition table for the trigger. An unqualified table or view name is implicitly qualified according to the following rules:

- If the CREATE FUNCTION statement is embedded in a program, the implicit qualifier is the authorization ID in the QUALIFIER bind option when the plan or package was created or last rebound. If QUALIFIER was not used, the implicit qualifier is the owner of the plan or package.
- If the CREATE FUNCTION statement is dynamically prepared, the implicit qualifier is the SQL authorization ID in the CURRENT_SQLID special register.

When the function is invoked, the corresponding columns of the transition table identified by the table locator and the table or view identified in the TABLE LIKE clause must have the same definition. The data type, length, precision, scale, and encoding scheme of these columns must match exactly. The description of the table or view at the time the CREATE FUNCTION statement was executed is used.

Additionally, a character FOR BIT DATA column of the transition table cannot be passed as input for a table parameter for which the corresponding column of the table specified at the definition is not defined as character FOR BIT DATA. (The definition occurs with the CREATE FUNCTION statement.) Likewise, a character column of the transition table that is not FOR BIT DATA cannot be passed as input for a table parameter for which the corresponding column of the table specified at the definition is defined as character FOR BIT DATA.

For more information about using table locators, see *DB2 Application Programming and SQL Guide*.

RETURNS

Identifies the output of the function.

data-type2

Specifies the data type of the output. The output is nullable.

You can specify any built-in data type or distinct type that can be cast to from the data type of the source function's result. To specify a LONG VARCHAR or LONG VARGRAPHIC, use VARCHAR or VARGRAPHIC with an explicit length instead. (For information on casting data types, see “Casting between data types” on page 62.) For additional rules that apply to the data type that you can specify, see “Rules for creating sourced functions” on page 583.

AS LOCATOR

Specifies that the function returns a locator to the value rather than the actual value. You can specify AS LOCATOR only if the output from the function has a LOB data type or a distinct type based on a LOB data type.

SPECIFIC *specific-name*

Provides a unique name for the function. The name is implicitly or explicitly qualified with a schema name. The name, including the schema name, must not identify the specific name of another function that exists at the current server.

The unqualified form of *specific-name* is a long SQL identifier. The qualified form is a short SQL identifier (the schema name) followed by a period and a long SQL identifier.

If you do not specify a schema name, it is the same as the explicit or implicit schema name of the function name (*function-name*). If you specify a schema name, it must be the same as the explicit or implicit schema name of the function name.

If you do not specify the SPECIFIC clause, the default specific name is the name of the function. However, if the function name does not provide a unique specific name or if the function name is a single asterisk, DB2 generates a specific name in the form of:

SQLxxxxxxxxxxxx

where 'xxxxxxxxxxxx' is a string of 12 characters that make the name unique.

The specific name is stored in the SPECIFIC column of the SYSROUTINES catalog table. The specific name can be used to uniquely identify the function in several SQL statements (such as ALTER FUNCTION, COMMENT ON, DROP, GRANT, and REVOKE) and in DB2 commands (START FUNCTION, STOP FUNCTION, and DISPLAY FUNCTION). However, the function cannot be invoked by its specific name.

PARAMETER CCSID

Indicates whether the encoding scheme for string parameters is ASCII, EBCDIC, or UNICODE. The default encoding scheme is the value specified in the CCSID clauses of the parameter list or RETURNS clause, or in the field DEF ENCODING SCHEME on installation panel DSNTIPF.

This clause provides a convenient way to specify the encoding scheme for *all* string parameters. If individual CCSID clauses are specified for individual parameters in addition to this PARAMETER CCSID clause, the value specified in *all* of the CCSID clauses must be the same value that is specified in this clause.

This clause also specifies the encoding scheme to be used for system-generated parameters of the routine such as message tokens and DBINFO.

SOURCE

Specifies that the function that you are registering is a sourced function. A *sourced function* is implemented by another function (the *source function*). The source function can be any previously created user-defined function except an external table function. The source function can also be any built-in function except the following (if a particular syntax is shown, only the indicated form cannot be specified):

- COUNT(*)
- COUNT_BIG(*)
- CHAR(*datetime-expression, second-argument*) where *second-argument* is ISO, USA, EUR, JIS, or LOCAL
- COALESCE
- NULLIF
- RAISE-ERROR
- STRIP
- VALUE

If you base the sourced function directly or indirectly on an external scalar function, the sourced function inherits the attributes of the external scalar function. This can involve several layers of sourced functions. For example, assume that function A is sourced on function B, which in turn is sourced on

CREATE FUNCTION (sourced)

function C. Function C is an external scalar function. Functions A and B inherit all of the attributes that are specified on the EXTERNAL clause of the CREATE FUNCTION statement for function C.

EXECUTE authority is required on the source function.

To specify a built-in function as the source function, use the last syntax variation in the following list, *function-name* (*parameter-type*, ...).

function-name

Identifies the function to be used as the source function by its function name. A schema name implicitly or explicitly qualifies the name. Only one function with this name must exist in the schema.

If you specify an unqualified *function-name*, DB2 uses the SQL path of the authorization ID (the value of the CURRENT PATH special register) to locate the function. The first schema that has only one function with this name on which the user has EXECUTE authority is selected. DB2 returns an error if it cannot find a function or encounters a schema that has more than one function with this name.

If you specify a qualified *function-name*, DB2 returns an error if there is no function with this name in the named schema or more than one function with this name exists in the schema.

SPECIFIC *specific-name*

Identifies the function to be used as the source function by its specific name. A schema name implicitly or explicitly qualifies the name.

If you specify an unqualified *specific-name*, DB2 uses the SQL path of the authorization ID (the value of the CURRENT PATH special register) to locate the schema. DB2 searches the SQL path and selects the first schema that contains a function with this specific name for which the user has EXECUTE authority. DB2 returns an error if it cannot find a function with the specific name in one of the schemas in the SQL path.

If you specify a qualified *specific-name*, DB2 searches the named schema for the function. DB2 returns an error if it cannot find a function with the specific name.

function-name (*parameter-type*,...)

Identifies the function to be used as the source function by its function signature. You must use this form of the syntax if the source function is a built-in function. You cannot use this form of the syntax if the source function was defined with a table parameter (the LIKE TABLE was specified in the CREATE FUNCTION statement to indicate that one of the input parameters is a transition table). Instead, identify the function with its function name, if unique, or with its specific name.

DB2 does not use function resolution to select the source function because the function signature uniquely identifies the function.

If *function-name()* is specified, the function that is identified must have zero parameters.

function-name

Identifies the function name of the source function. If you specify an unqualified name, DB2 searches the schemas of the SQL path; otherwise, DB2 searches the named schema.

parameter-type,...

Provides a list of data types, separated by commas, that must match

the data types of the parameters of the source function. DB2 uses the number of data types and the logical concatenation of the data types to identify the source function.

For data types that have length, precision, or scale attributes, you can either specify a value for the attribute or use a set of empty parentheses (with the noted exceptions):

- Empty parentheses indicate that DB2 is to ignore the attribute when determining whether the data types match. For example, DEC() will match a parameter whose data type is DEC(7,2).
FLOAT cannot be specified with empty parentheses because its parameter value indicates different data types (REAL or DOUBLE).
- If you use a specific value for a length, precision, or scale attribute, the value must exactly match the value that was specified (implicitly or explicitly) in the CREATE FUNCTION statement for the source function. For example, DECIMAL(7,4) does not match a parameter whose data type is DECIMAL(7,2). Coding specific values for length, precision, scale, subtype, and encoding scheme attributes ensures that the source function you intend to use is used.

The specific value for FLOAT(*n*) does not have exactly match the defined value of the source function because 1<=n= 21 indicates REAL and 22<=n<=53 indicates DOUBLE. Matching is based on whether the data type is REAL or DOUBLE.

- If length, precision, or scale is not explicitly specified, and empty parentheses are not specified, the default length of the data type is implied. For example:

CHAR	CHAR(1)
GRAPHIC	GRAPHIC(1)
DECIMAL	DECIMAL(5,0)
FLOAT	DOUBLE (length of 8)

The implicit length must exactly match the value that was specified (implicitly or explicitly) in the CREATE FUNCTION statement. For a complete list of the default lengths of data types, see “CREATE TABLE” on page 653.

For either empty parentheses or specific values, the synonyms for data types are considered a match. For example, DEC and NUMERIC will match.

If you omit the FOR DATA or CCSID clause for data types with a subtype or encoding scheme attribute, DB2 is to ignore the attribute when determining whether the data types match. An exception to ignoring the attribute is FOR BIT DATA. A character FOR BIT DATA parameter of the new function cannot correspond to a parameter of the source function that is not defined as character FOR BIT DATA. Likewise, a character parameter of the new function that is not FOR BIT DATA cannot correspond to a parameter of the source function that is defined as character FOR BIT DATA.

If no function with the specified signature exists in the explicitly or implicitly specified schema, an error occurs.

The number of input parameters in the function that is being created must be the same as the number of parameters in the source function.

CREATE FUNCTION (sourced)

If the data type of each input parameter is not the same as or castable to the corresponding parameter of the source function, an error occurs. The data type of the final result of the source function must match or be castable to the result of the sourced function.

Notes

Choosing data types for parameters: When you choose the data types of the input and output parameters for your function, consider the rules of promotion that can affect the values of the parameters. (See “Promotion of data types” on page 61). For example, a constant that is one of the input arguments to the function might have a built-in data type that is different from the data type that the function expects, and more significantly, might not be promotable to that expected data type. Based on the rules of promotion, we recommend using the following data types for parameters:

- INTEGER instead of SMALLINT
- DOUBLE instead of REAL
- VARCHAR instead of CHAR
- VARGRAPHIC instead of GRAPHIC

For portability of functions across platforms that are not DB2 for OS/390 and z/OS, do not use the following data types, which might have different representations on different platforms:

- FLOAT. Use DOUBLE or REAL instead.
- NUMERIC. Use DECIMAL instead.

Specifying the encoding scheme for parameters: The implicitly or explicitly specified encoding scheme of all the parameters with a string data type (both input and output parameters) must be the same—either all ASCII, all EBCDIC, or all UNICODE.

Determining the uniqueness of functions in a schema: At the current server, the function signature of each function, which is the qualified function name combined with the number and data types of the input parameters, must be unique. This means that two different schemas can each contain a function with the same name that have the same data types for all of their corresponding data types. However, a schema must not contain two functions with the same name that have the same data types for all of their corresponding data types.

When determining whether corresponding data types match, DB2 does not consider any length, precision, scale, subtype or encoding scheme attributes in the comparison. DB2 considers the synonyms of data types (DECIMAL and NUMERIC, REAL and FLOAT, and DOUBLE and FLOAT) a match. Therefore, CHAR(8) and CHAR(35) are considered to be the same, as are DECIMAL(11,2), DECIMAL(4,3), and NUMERIC(4,2).

Assume that the following statements are executed to create four functions in the same schema. The second and fourth statements fail because they create functions that are duplicates of the functions that the first and third statements created.

```
CREATE FUNCTION PART (INT, CHAR(15)) ...
CREATE FUNCTION PART (INTEGER, CHAR(40)) ...

CREATE FUNCTION ANGLE (DECIMAL(12,2)) ...
CREATE FUNCTION ANGLE (DEC(10,7)) ...
```

Rules for creating sourced functions: For the discussion in this section, assume that the function that is being created is named NEWF and the source function is named SOURCEF. Consider the following rules when creating a sourced function:

- The unqualified names of the sourced function and source function can be different (NEWF and SOURCEF).
- The number of input parameters for NEWF and SOURCEF must be the same; otherwise, an error occurs when the CREATE FUNCTION statement is executed.
- When specifying the input parameters and output for NEWF, you can specify a value for the precision, scale, subtype, or encoding scheme for a data type with any of these attributes or use empty parentheses.

Empty parentheses, such as VARCHAR(), indicate that the value of the attribute is the same as the attribute for the corresponding parameter of SOURCEF, or that is determined by data type promotion. If you specify any values for the attributes, DB2 checks the values against the corresponding input parameters and returned output of SOURCEF as described next.

- When the CREATE FUNCTION statement is executed, DB2 checks the input parameters of NEWF against those of SOURCEF. The data type of each input parameter of NEWF function must be either the same as, or promotable to, the data type of the corresponding parameter of SOURCEF; otherwise, an error occurs. (For information on the promotion of data types, see “Casting between data types” on page 62.)

This checking does not guarantee that an error will not occur when NEWF is invoked. For example, an argument that matches the data type and length or precision attributes of a NEWF parameter might not be promotable if the corresponding SOURCEF parameter has a shorter length or less precision. In general, do not define the parameters of a sourced function with length or precision attributes that are greater than the attributes of the corresponding parameters of the source function.

- When the CREATE FUNCTION statement is executed, DB2 checks the data type identified in the RETURNS clause of NEWF against the data type that SOURCEF returns. The data type that SOURCEF returns must be either the same as, or promotable to, the RETURNS data type of NEWF; otherwise, an error occurs.

This checking does not guarantee that an error will not occur when NEWF is invoked. For example, the value of a result that matches the data type and length or precision attributes of those specified for SOURCEF's result might not be promotable if the RETURNS data type of NEWF has a shorter length or less precision. Consider the possible effects of defining the RETURNS data type of a sourced function with length or precision attributes that are less than the attributes defined for the data type returned by source function.

Scrollable cursors specified with user-defined functions: A row can be fetched more than once with a scrollable cursor. Therefore, if a scrollable cursor is defined with a non-deterministic function in the select list of the cursor, a row can be fetched multiple times with different results for each fetch. (However, the value of a non-deterministic function in the WHERE clause of a scrollable cursor is captured when the cursor is opened and remains unchanged until the cursor is closed.) Similarly, if a scrollable cursor is defined with a user-defined function with external action, the action is executed with every fetch.

CREATE FUNCTION (sourced)

Examples

Example 1: Assume that you created a distinct type HATSIZE, which you based on the built-in data type INTEGER. You want to have an AVG function to compute the average hat size of different departments. Create a sourced function that is based on built-in function AVG.

```
CREATE FUNCTION AVE (HATSIZE) RETURNS HATSIZE  
    SOURCE SYSIBM.AVG (INTEGER);
```

When you created distinct type HATSIZE, two cast functions were generated, which allow HATSIZE to be cast to INTEGER for the argument and INTEGER to be cast to HATSIZE for the result of the function.

Example 2: After Smith registered the external scalar function CENTER in his schema, you decide that you want to use this function, but you want it to accept two INTEGER arguments instead of one INTEGER argument and one FLOAT argument. Create a sourced function that is based on CENTER.

```
CREATE FUNCTION MYCENTER (INTEGER, INTEGER)  
    RETURNS FLOAT  
    SOURCE SMITH.CENTER (INTEGER, FLOAT);
```

CREATE FUNCTION (SQL scalar)

This statement is used to define a user-defined SQL scalar function. A scalar function returns a single value each time it is invoked. Specifying a function is generally valid wherever an SQL expression is valid.

Invocation

This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is implicitly or explicitly specified.

Authorization

The privilege set defined below must include at least one of the following:

- The CREATEIN privilege for the schema or all schemas
- SYSADM or SYSCTRL authority

The authorization ID that matches the schema name implicitly has the CREATEIN privilege on the schema.

Privilege set: If the statement is embedded in an application program, the privilege set is the privileges that are held by the authorization ID of the owner of the plan or package.

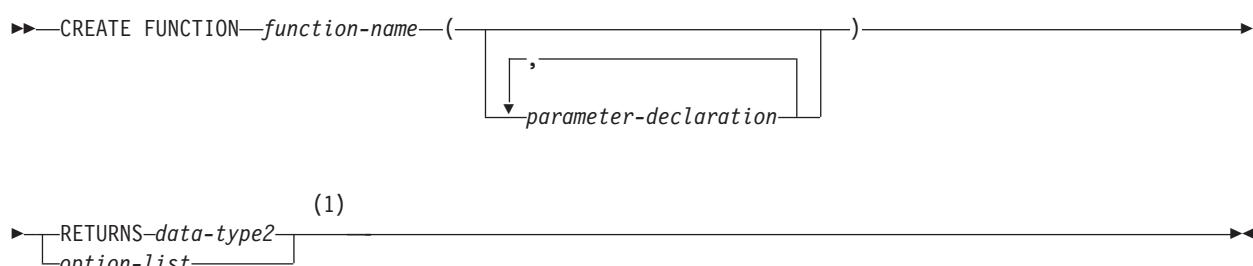
If the statement is dynamically prepared, the privilege set is the privileges that are held by the SQL authorization ID of the process. The specified function name can include a schema name (a qualifier). However, if the schema name is not the same as the SQL authorization ID, one of the following conditions must be met:

- The privilege set includes SYSADM or SYSCTRL authority.
- The SQL authorization ID of the process has the CREATEIN privilege on the schema.

Additional privileges are required if the function uses a table as a parameter or refers to a distinct type. These privileges are:

- The SELECT privilege on any table that is an input parameter to the function.
- The USAGE privilege on each distinct type that the function references.

Syntax



Notes:

- 1 The RETURNS clause and the clauses that follow in the *option-list* can be specified in any order. However, the same clause cannot be specified more than once.

CREATE FUNCTION (SQL scalar)

parameter-declaration:

►—parameter-name——(1)——data-type1

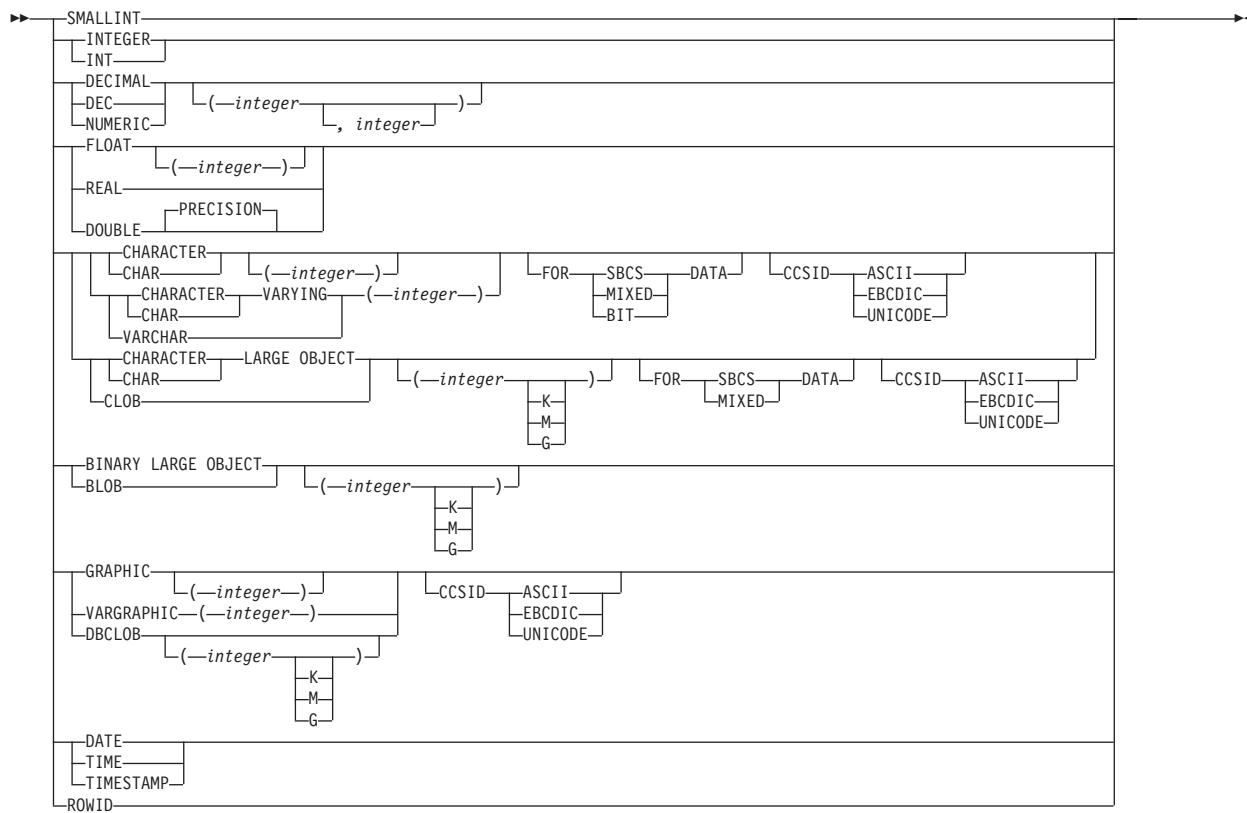
Notes:

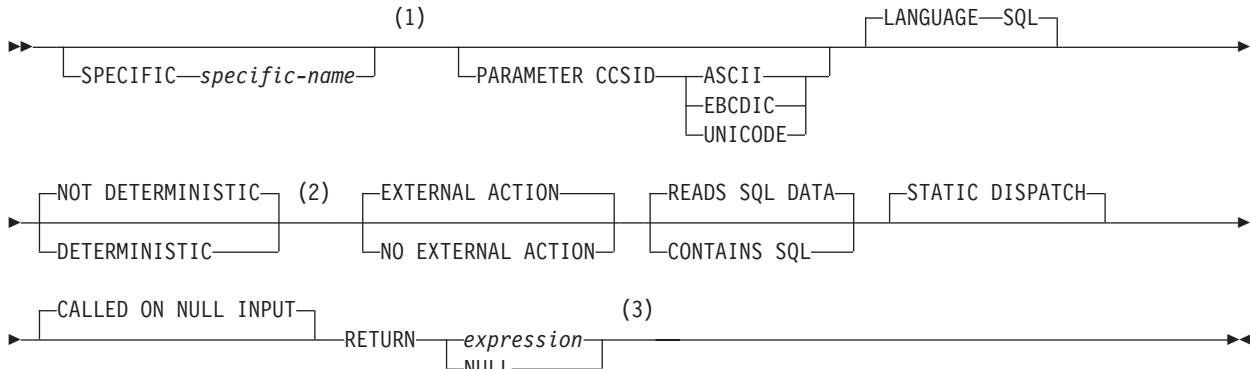
- 1 Note that the *parameter-name* is required for SQL functions.

data-type:

► *built-in-data-type*
 └ *distinct-type-name* ┘

built-in-data-type:



option-list:**Notes:**

- 1 This clause and the other clauses in the *option-list* can be specified in any order. However, the same clause cannot be specified more than once.
- 2 Synonyms for this clause include VARIANT for NOT DETERMINISTIC, and NOT VARIANT for DETERMINISTIC.
- 3 For an SQL function, the RETURN clause is required, but it can be specified anywhere within the *option-list*.

Description

function-name

Names the user-defined function. The name is implicitly or explicitly qualified by a schema name. The combination of name, schema name, the number of parameters, and the data type of each parameter³² (without regard for any length, precision, scale, subtype or encoding scheme attributes of the data type) must not identify a user-defined function that exists at the current server.

You can use the same name for more than one function if the function signature of each function is unique.

- The unqualified form of *function-name* is a long SQL identifier.

The name must not be any of the following system-reserved keywords even if you specify them as delimited identifiers:

ALL	LIKE	UNIQUE
AND	MATCH	UNKNOWN
ANY	NOT	=
BETWEEN	NULL	~=
DISTINCT	ONLY	<
EXCEPT	OR	≤
EXISTS	OVERLAPS	~<
FALSE	SIMILAR	>
FOR	SOME	≥
FROM	TABLE	~>
IN	TRUE	~>
IS	TYPE	

32. If the function has more than 30 parameters, only the first 30 parameters are used to determine whether the function is unique.

CREATE FUNCTION (SQL scalar)

The unqualified function name is implicitly qualified with a schema name according to the following rules:

- If the statement is embedded in a program, the schema name is the authorization ID in the QUALIFIER bind option when the plan or package was created or last rebound. If QUALIFIER was not specified, the schema name is the owner of the plan or package.
- If the statement is dynamically prepared, the schema name is the SQL authorization ID in the CURRENT SQLID special register.
- The qualified form of *function-name* is a short SQL identifier (the schema name) followed by a period and a long SQL identifier.

The schema name must not begin with 'SYS' unless the schema name is 'SYSADM'.

The owner of the function is determined by how the CREATE FUNCTION statement is invoked:

- If the statement is embedded in a program, the owner is the authorization ID of the owner of the plan or package.
- If the statement is dynamically prepared, the owner is the SQL authorization ID in the CURRENT SQLID special register.

The owner is implicitly given the EXECUTE privilege with the GRANT option for the function.

(parameter-declaration,...)

Identifies the number of input parameters of the function, and specifies the name and data type of each parameter. All the parameters for a function are input parameters and are nullable. There must be one entry in the list for each parameter that the function expects to receive.

A function can have no parameters. In this case, you must code an empty set of parentheses, for example:

CREATE FUNCTION WOOFER()

parameter-name

Specifies the name of the input parameter. The name is a long SQL identifier, and each name in the parameter list must not be the same as any other name.

data-type

Specifies the data type of the input parameter. The data type can be a built-in data type or a distinct type.

built-in-data-type

The data type of the input parameter is a built-in data type. You can use the same built-in data types as for the CREATE TABLE statement except LONG VARCHAR or LONG VARGRAPHIC. Use VARCHAR or VARGRAPHIC with an explicit length instead.

If you do not specify a specific value for the data types that have length, precision, or scale attributes (CHAR, GRAPHIC, DECIMAL, NUMERIC, FLOAT), the defaults are as follows:

CHAR	CHAR(1)
GRAPHIC	GRAPHIC(1)
DECIMAL	DECIMAL(5,0)
FLOAT	DOUBLE (length of 8)

For information on the data types, including the subtype of character data types (the FOR *subtype* DATA clause), see “built-in-data-type” on page 658.

For parameters with a string data type, the CCSID clause indicates whether the encoding scheme of the parameter value is ASCII, EBCDIC, or UNICODE. If you do not specify CCSID ASCII, CCSID EBCDIC, or CCSID UNICODE, the encoding scheme is the value of field DEF ENCODING SCHEME on installation panel DSNTIPF.

distinct-type-name

The data type of the input parameter is a distinct type. Any length, precision, scale, subtype, or encoding scheme attributes for the parameter are those of the source type of the distinct type.

If you specify the name of the distinct type without a schema name, DB2 resolves the schema name by searching the schemas in the SQL path.

The implicitly or explicitly specified encoding scheme of all the parameters with a string data type must be the same—either all ASCII, all EBCDIC, or all UNICODE.

Although parameters with a character data type have an implicitly or explicitly specified subtype (BIT, SBCS, or MIXED), the function program can receive character data of any subtype. Therefore, conversion of the input data to the subtype of the parameter might occur when the function is invoked. An error occurs if mixed data that actually contains DBCS characters is used as the value for an input parameter that is declared with an SBCS subtype.

Parameters with a datetime data type or a distinct type are passed to the function as a different data type:

- A datetime type parameter is passed as a character data type, and the data is passed in ISO format.

The encoding scheme for a datetime type parameter is the same as the implicitly or explicitly specified encoding scheme of any character or graphic string parameters. If no character or graphic string parameters are passed, the encoding scheme is the value of field DEF ENCODING SCHEME on installation panel DSNTIPF.

- A distinct type parameter is passed as the source type of the distinct type.

RETURNS

Identifies the output of the function. Consider this clause in conjunction with the optional CAST FROM clause.

data-type2

Specifies the data type of the output. The output is nullable.

The same considerations that apply to the data type of input parameter, as described under “*data-type*” on page 588, apply to the data type of the output of the function.

SPECIFIC *specific-name*

Specifies a unique name for the function. The name is implicitly or explicitly qualified with a schema name. The name, including the schema name, must not identify the specific name of another function that exists at the current server.

CREATE FUNCTION (SQL scalar)

The unqualified form of *specific-name* is a long SQL identifier. The qualified form is a short SQL identifier (the schema name) followed by a period and a long SQL identifier.

If you do not specify a schema name, it is the same as the explicit or implicit schema name of the function name (*function-name*). If you specify a schema name, it must be the same as the explicit or implicit schema name of the function name.

If you do not specify the SPECIFIC clause, the default specific name is the name of the function. However, if the function name does not provide a unique specific name or if the function name is a single asterisk, DB2 generates a specific name in the form of:

SQLxxxxxxxxxxxx

where 'xxxxxxxxxxxx' is a string of 12 characters that make the name unique.

The specific name is stored in the SPECIFIC column of the SYSROUTINES catalog table. The specific name can be used to uniquely identify the function in several SQL statements (such as ALTER FUNCTION, COMMENT ON, DROP, GRANT, and REVOKE) and must be used in DB2 commands (START FUNCTION, STOP FUNCTION, and DISPLAY FUNCTION). However, the function cannot be invoked by its specific name.

PARAMETER CCSID

Indicates whether the encoding scheme for string parameters is ASCII, EBCDIC, or UNICODE. The default encoding scheme is the value specified in the CCSID clauses of the parameter list or RETURNS clause, or in the field DEF ENCODING SCHEME on installation panel DSNTIPF.

This clause provides a convenient way to specify the encoding scheme for *all* string parameters. If individual CCSID clauses are specified for individual parameters in addition to this PARAMETER CCSID clause, the value specified in *all* of the CCSID clauses must be the same value that is specified in this clause.

This clause also specifies the encoding scheme to be used for system-generated parameters of the routine such as message tokens and DBINFO.

LANGUAGE SQL

Specifies the application programming language in which the stored function is written. The value of the function is written as DB2 SQL within the *expression* of the RETURN clause. LANGUAGE SQL is the default.

NOT DETERMINISTIC or DETERMINISTIC

Specifies whether the function returns the same results for identical input arguments.

NOT DETERMINISTIC

The function might not return the same result for identical input arguments. The function depends on some state values that affect the results. DB2 uses this information to disable the merging of views and table expressions when processing SELECT, UPDATE, DELETE, or INSERT statements that reference this function. An example of a function that is not deterministic is one that generates random numbers.

NOT DETERMINISTIC must be specified explicitly or implicitly if the function program accesses a special register or invokes another non-deterministic function. NOT DETERMINISTIC is the default.

DETERMINISTIC

The function always returns the same result for identical input arguments. An example of a deterministic function is a function that calculates the square root of the input. DB2 uses this information to enable the merging of views and table expressions for SELECT, UPDATE, DELETE, or INSERT statements that reference this function. DETERMINISTIC is not the default. If applicable, specify DETERMINISTIC to prevent non-optimal access paths from being chosen for SQL statements that reference this function.

DB2 does not verify that the function program is consistent with the specification of DETERMINISTIC or NOT DETERMINISTIC.

EXTERNAL ACTION or NO EXTERNAL ACTION

Specifies whether the function takes an action that changes the state of an object that DB2 does not manage. An example of an external action is sending a message or writing a record to a file.

EXTERNAL ACTION

The function can take an action that changes the state of an object that DB2 does not manage.

Some SQL statements that invoke functions with external actions can result in incorrect results if parallel tasks execute the function. For example, if the function sends a note for each initial call to it, one note is sent for each parallel task instead of once for the function. Specify the DISALLOW PARALLEL clause for functions that do not work correctly with parallelism.

If you specify EXTERNAL ACTION, then DB2:

- Materializes the views and table expressions in SELECT, UPDATE, DELETE or INSERT statements that refer to the function. This materialization can adversely affect the access paths that are chosen for the SQL statements that reference this function. Do not specify EXTERNAL ACTION if the function does not have an external action.
- Does not move the function from one task control block (TCB) to another between FETCH operations.
- Does not allow another function or stored procedure to use the TCB until the cursor is closed. This is also applicable for cursors declared WITH HOLD.

The only changes to resources made outside of DB2 that are under the control of commit and rollback operations are those changes made under RRS control.

EXTERNAL ACTION must be specified implicitly or explicitly specified if the SQL routine body invokes a function that is defined with EXTERNAL ACTION. EXTERNAL ACTION is the default.

NO EXTERNAL ACTION

The function does not take any action that changes the state of an object that DB2 does not manage. DB2 uses this information to enable the merging of views and table expressions for SELECT, UPDATE, DELETE, or INSERT statements that reference this function. NO EXTERNAL ACTION is not the default. If applicable, specify NO EXTERNAL ACTION to prevent non-optimal access paths from being chosen for SQL statements that reference this function.

DB2 does not verify that the function program is consistent with the specification of EXTERNAL ACTION or NO EXTERNAL ACTION.

CREATE FUNCTION (SQL scalar)

READS SQL DATA or CONTAINS SQL

Indicates whether the function can execute any SQL statements and, if so, what type. DB2 verifies that the SQL issued by the function is consistent with this specification. See Table 72 on page 972 for a detailed list of the SQL statements that can be executed under each data access indication.

READS SQL DATA

The function does not execute SQL statements that modify data. SQL statements that are not supported in any function return a different error.

READS SQL DATA is the default.

CONTAINS SQL

The function does not execute SQL statements that read or modify data. SQL statements that are not supported in any function return a different error.

STATIC DISPATCH

At function resolution time, DB2 chooses a function based on the static (or declared) types of the function parameters. STATIC DISPATCH is the default.

CALLED ON NULL INPUT

The function is called regardless of whether any of the input arguments is null, making the function responsible for testing for null arguments. The function can return null. CALLED ON NULL INPUT is the default.

RETURN

Specifies the return value of the function. The *expression* (or NULL) is the body of the function. Parameter names can be referenced in the RETURN clause. Parameter names can be qualified by the function name to avoid ambiguous reference.

expression

Specifies the *expression* to be returned for the function. The data type of the expression result must be assignable to the data type defined in the RETURNS clause of the function.

The *expression* can contain one or more of the following:

- Function (either user-defined or built-in)
- Expression enclosed in parentheses
- Constant
- Special register
- Labeled duration
- CASE expression
- CAST specification

The *expression* cannot include a column name or a host variable. See “Expressions” on page 111 for information on expressions.

NULL

Specifies that the function returns null for the data type defined in the RETURNS clause.

Notes

Choosing data types for parameters: When you choose the data types of the input and output parameters for your function, consider the rules of promotion that can affect the values of the parameters. (See “Promotion of data types” on page 61). For example, a constant that is one of the input arguments to the function might have a built-in data type that is different from the data type that the function

expects, and more significantly, might not be promotable to that expected data type. Based on the rules of promotion, using the following data types for parameters is recommended:

- INTEGER instead of SMALLINT
- DOUBLE instead of REAL
- VARCHAR instead of CHAR
- VARGRAPHIC instead of GRAPHIC

For portability of functions across platforms that are not DB2 for OS/390 and z/OS, do not use the following data types, which might have different representations on different platforms:

- FLOAT. Use DOUBLE or REAL instead.
- NUMERIC. Use DECIMAL instead.

Specifying the encoding scheme for parameters: The implicitly or explicitly specified encoding scheme of all the parameters with a string data type (both input and output parameters) must be the same—either all ASCII or all EBCDIC.

Determining the uniqueness of functions in a schema: At the current server, the function signature of each function, which is the qualified function name combined with the number and data types of the input parameters, must be unique. If the function has more than 30 input parameters, only the data types of the first 30 are used to determine uniqueness. This means that two different schemas can each contain a function with the same name that have the same data types for all of their corresponding data types. However, a single schema must not contain multiple functions with the same name that have the same data types for all of their corresponding data types.

When determining whether corresponding data types match, DB2 does not consider any length, precision, scale, subtype or encoding scheme attributes in the comparison. DB2 considers the synonyms of data types (DECIMAL and NUMERIC, REAL and FLOAT, and DOUBLE and FLOAT) a match. Therefore, CHAR(8) and CHAR(35) are considered to be the same, as are DECIMAL(11,2), DECIMAL(4,3), and NUMERIC(4,2).

Assume that the following statements are executed to create four functions in the same schema. The second and fourth statements fail because they create functions that are duplicates of the functions that the first and third statements created.

```
CREATE FUNCTION PART (A INT, B CHAR(15)) ...
CREATE FUNCTION PART (A INTEGER, B CHAR(40)) ...

CREATE FUNCTION ANGLE (A DECIMAL(12,2)) ...
CREATE FUNCTION ANGLE (A DEC(10,7)) ...
```

Overriding a built-in function: Giving an SQL function the same name as a built-in function is not a recommended practice unless you are trying to change the functionality of the built-in function.

If you do intend to create an SQL function with the same name as a built-in function, be careful to maintain the uniqueness of its function signature. If your function has the same name and data types of the corresponding parameters of the built-in function but implements different logic, DB2 might choose the wrong function when the function is invoked with an unqualified function name. Thus, the application might fail, or perhaps even worse, run successfully but provide an inappropriate result.

CREATE FUNCTION (SQL scalar)

Resolution of function invocations: DB2 resolves function invocations inside the body of the function according to the SQL path that is in effect for the CREATE FUNCTION statement. This SQL path does not change after the function is created.

Referencing date and time special registers: If an SQL function contains multiple references to any of the date or time special registers, all references return the same value. Further, this value is the same value returned by the register invocation in the statement that invoked the function.

Self-referencing function: The body of an SQL function (that is, the *expression* or NULL in the RETURN clause of the CREATE FUNCTION statement) cannot contain a recursive invocation of itself or to another function that invokes it, because such a function would not exist to be referenced.

Scrollable cursors specified with user-defined functions: A row can be fetched more than once with a scrollable cursor. Therefore, if a scrollable cursor is defined with a non-deterministic function in the select list of the cursor, a row can be fetched multiple times with different results for each fetch. (However, the value of a non-deterministic function in the WHERE clause of a scrollable cursor is captured when the cursor is opened and remains unchanged until the cursor is closed.) Similarly, if a scrollable cursor is defined with a user-defined function with external action, the action is executed with every fetch.

Examples

Example 1: Define a scalar function that returns the tangent of a value using existing SIN and COS built-in functions:

```
CREATE FUNCTION TAN (X DOUBLE)
  RETURNS DOUBLE
  LANGUAGE SQL
  CONTAINS SQL
  NO EXTERNAL ACTION
  DETERMINISTIC
  RETURN SIN(X)/COS(X);
```

CREATE GLOBAL TEMPORARY TABLE

The CREATE GLOBAL TEMPORARY TABLE statement creates a description of a temporary table at the current server.

Invocation

This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared.

Authorization

The privilege set that is defined below must include at least one of the following:

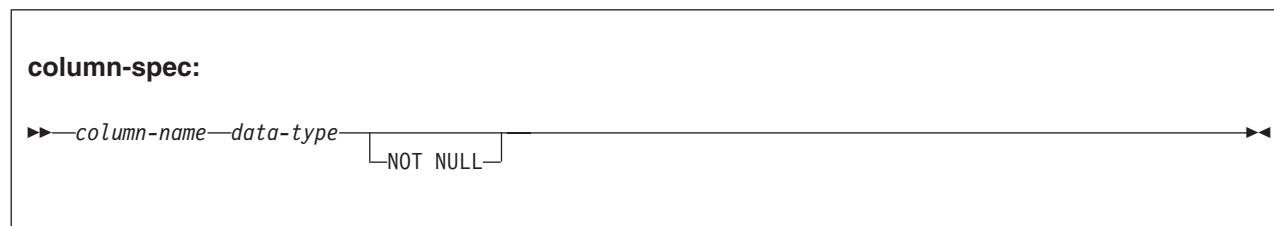
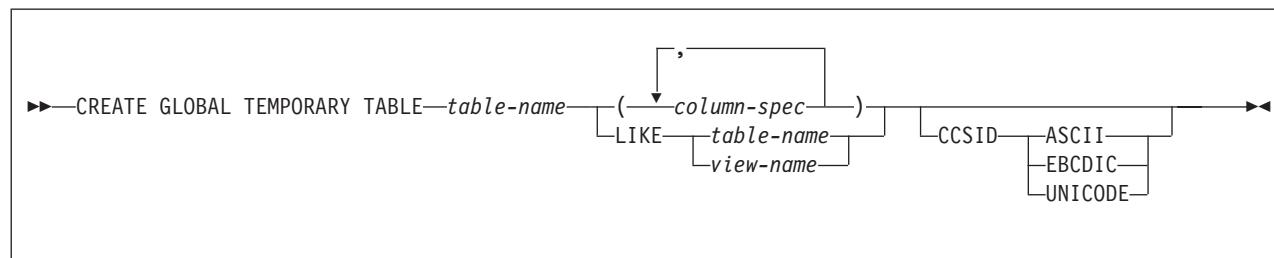
- The CREATETMTAB system privilege
- The CREATETAB database privilege for any database
- DBADM, DBCTRL, or DBMAINT authority for any database
- SYSADM or SYSCTRL authority

However, DABADM, DBCTRL, or DBMAINT authority is not sufficient authority if you are creating a temporary table for someone else and the table qualifier is not your authorization ID.

Additional privileges might be required when the data type of a column is a distinct type or the LIKE clause is specified. See the description of *distinct-type* and LIKE for the details.

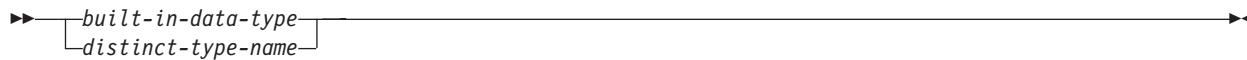
Privilege set: The privilege set is the same as the privilege set for the CREATE TABLE statement. See “Privilege Set” on page 653 for details.

Syntax

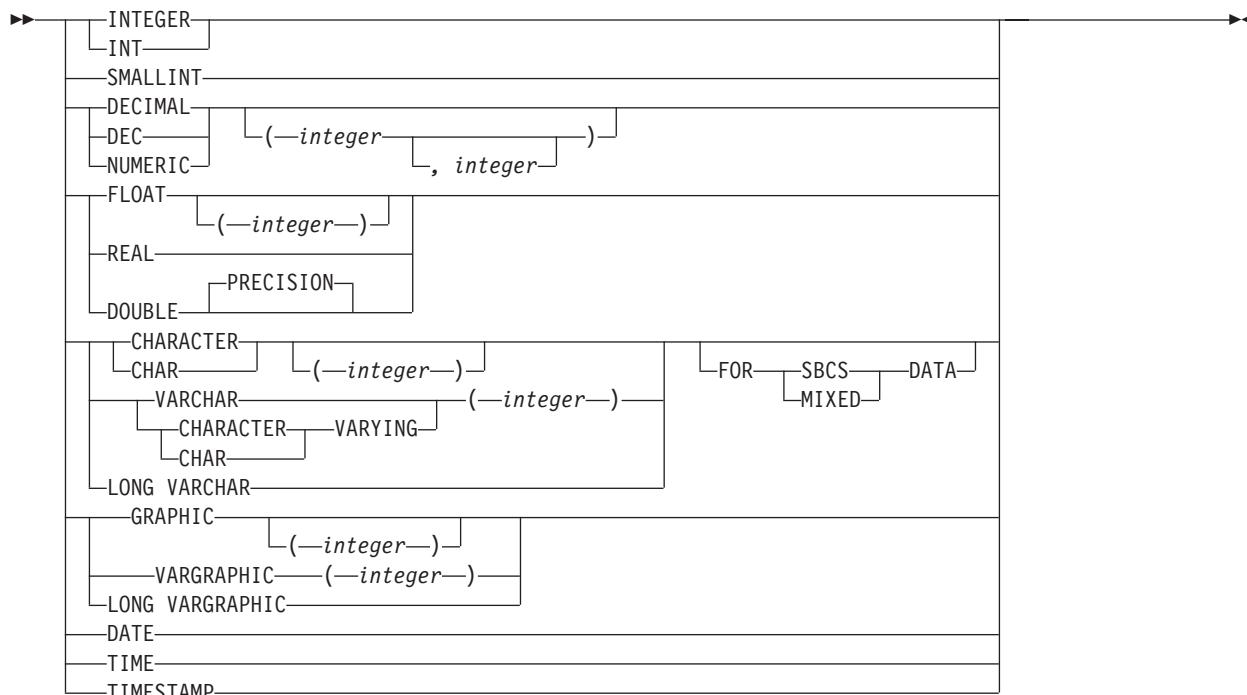


CREATE GLOBAL TEMPORARY TABLE

data-type:



built-in-data-type:



Description

table-name

Names the temporary table. The name, including the implicit or explicit qualifier, must not identify a table, view, alias, synonym, or temporary table that exists at the database server.

The qualification rules for *table-name* are the same as for *table-name* in the CREATE TABLE statement. (See “*table-name*” on page 658.)

The owner acquires ALL PRIVILEGES on the table WITH GRANT OPTION and the authority to drop the table.

column-spec

Defines the attributes of a column for each instance of the table. The number of columns defined must not exceed 750. The maximum record size must not exceed 32714 bytes. The maximum row size must not exceed 32706 bytes (8 bytes less than the maximum record size).

column-name

Names the column. The name must not be qualified and must not be the same as the name of another column in the table.

data-type

Specifies the data type of the column. The data type can be a built-in data type or a distinct type.

built-in-data-type

Any built-in data type that can be specified for the CREATE TABLE statement with the exception that you cannot define a temporary table with a LOB or ROWID column.

For more information on and the rules that apply to the data types, including the subtype of character data types (the FOR *subtype* DATA clause, see “*built-in-data-type*” on page “*built-in-data-type*” on page 658.

distinct-type

Any distinct type except one that is based on a LOB or ROWID data type. The privilege set must implicitly or explicitly include the USAGE privilege on the distinct type.

NOT NULL

Specifies that the column cannot contain nulls. Omission of NOT NULL indicates that the column can contain nulls.

LIKE *table-name* or *view-name*

Specifies that the columns of the table have exactly the same name and description as the columns of the identified table or view. The name specified after LIKE must identify a table, view, or temporary table that exists at the current server. The privilege set must implicitly or explicitly include the SELECT privilege on the identified table or view.

This clause is similar to the LIKE clause on CREATE TABLE, but it has the following differences:

- If any column of the identified table or view has an attribute value that is not allowed for a column in a temporary table, that attribute value is ignored. The corresponding column in the new temporary table has the default value for that attribute unless otherwise indicated.
- If any column of the identified table or view allows a default value other than null, then that default value is ignored and the corresponding column in the new temporary table has no default value. A default value other than null is not allowed for any column in a temporary table.

CCSID *encoding-scheme*

Specifies the encoding scheme for string data stored in the table.

ASCII Specifies that the data must be encoded by using the ASCII CCSIDs of the server.

An error occurs if a valid ASCII CCSID has not been specified for the installation.

EBCDIC

Specifies that data must be encoded by using the EBCDIC CCSIDs of the server.

An error occurs if a valid EBCDIC CCSID has not been specified for the installation.

CREATE GLOBAL TEMPORARY TABLE

UNICODE

Specifies that data must be encoded by using the CCSIDs of the server for Unicode.

An error occurs if a valid CCSID for Unicode has not been specified for the installation.

Usually, each encoding scheme requires only a single CCSID. Additional CCSIDs are needed when mixed, graphic, or Unicode data is used. An error occurs if CCSIDs have not been defined.

For the creation of temporary tables, the CCSID clause can be specified whether or not the LIKE clause is specified. If the CCSID clause is specified, the encoding scheme of the new table is the scheme that is specified in the CCSID clause. If the CCSID clause is not specified, the encoding scheme of the new table is the same as the scheme for the table specified in the LIKE clause.

Notes

Instantiation and termination: Let T be a temporary table defined at the current server and let P denote an application process:

- An empty instance of T is created as a result of the first implicit or explicit reference to T in an OPEN, SELECT INTO, INSERT, or DELETE operation that is executed by any program in P.
- Any program in P can reference T and any reference to T by a program in P is a reference to that instance of T.

When a commit operation terminates a unit of work in P and no program in P has an open WITH HOLD cursor that is dependent on T, the commit includes the operation DELETE FROM T.

- When a rollback operation terminates a unit of work in P, the rollback includes the operation DELETE FROM T.
- When the connection to the database server at which an instance of T was created terminates, the instance of T is destroyed. However, the definition of T remains. A DROP TABLE statement must be executed to drop the definition of T.

Restrictions and extensions: Let T denote a temporary table:

- Columns of T cannot have default values other than null.
- A column of T cannot have a LOB or ROWID data type (or a distinct type based on one).
- T cannot have unique constraints, referential constraints, or check constraints.
- T cannot be defined as the parent in a referential constraint.
- T cannot be referenced in:
 - A CREATE INDEX statement.
 - A LOCK TABLE statement.
 - As the object of an UPDATE statement in which the object is T or a view of T. However, you can reference T in the WHERE clause of an UPDATE statement.
 - DB2 utility commands.
- As with all tables stored in a work file, query parallelism cannot be considered for any query that references T.
- If T is referenced in the *subselect* of a CREATE VIEW statement, you cannot specify a WITH CHECK OPTION clause in the CREATE VIEW statement.

- ALTER TABLE T is valid only if the statement is used to add a column to T. Any column that you add to T must have a default value of null.

When you alter T, any plans and packages that refer to the table are invalidated, and DB2 automatically rebinds the plans and packages the next time they are run.

- DELETE FROM T or a *view of T* is valid only if the statement does not include a WHERE or WHERE CURRENT OF clause. In addition, DELETE FROM *view of T* is valid only if the view was created (CREATE VIEW) without the WHERE clause. A DELETE FROM statement deletes all the rows from the table or view.
- You can refer to T in the FROM clause of any subselect. If you refer to T in the first FROM clause of a select-statement, you cannot specify a FOR UPDATE clause.
- You cannot use a DROP DATABASE statement to implicitly drop T. To drop T, reference T in a DROP TABLE statement.
- A temporary table instantiated by an SQL statement using a three-part table name can be accessed by another SQL statement using the same name in the same application process for as long as the DB2 connection which established the instantiation is not terminated.
- GRANT ALL PRIVILEGES ON T is valid, but you cannot grant specific privileges on T.

Of the ALL privileges, only the ALTER, INSERT, DELETE, and SELECT privileges can actually be used on T.

- REVOKE ALL PRIVILEGES ON T is valid, but you cannot revoke specific privileges from T.
- A COMMIT operation deletes all rows of every temporary table of the application process, but the rows of T are not deleted if any program in the application process has an open WITH HOLD cursor that is dependent on T. In addition, if RELEASE(COMMIT) is in effect and no open WITH HOLD cursors are dependent on T, all logical work files for T are also deleted.
- A ROLLBACK operation deletes all rows and all logical work files of every temporary table of the application process.
- You can reuse threads when using a temporary table, and a logical work file for a temporary table name remains available until deallocation. A new logical work file is not allocated for that temporary table name when the thread is reused.
- You can refer to T in the following statements:

ALTER FUNCTION	CREATE PROCEDURE	DECLARE TABLE
ALTER PROCEDURE	CREATE SYNONYM	DROP TABLE
COMMENT ON	CREATE TABLE LIKE	INSERT
CREATE ALIAS	CREATE VIEW	LABEL ON
CREATE FUNCTION	DESCRIBE TABLE	SELECT INTO

Examples

Example 1: Create a temporary table, CURRENTMAP. Name two columns, CODE and MEANING, both of which cannot contain nulls. CODE contains numeric data and MEANING has character data. Assuming a value of NO for the field MIXED DATA on installation panel DSNTIPF, column MEANING has a subtype of SBCS:

```
CREATE GLOBAL TEMPORARY TABLE CURRENTMAP
  (CODE INTEGER NOT NULL, MEANING VARCHAR(254) NOT NULL);
```

Example 2: Create a temporary table, EMP:

CREATE GLOBAL TEMPORARY TABLE

```
CREATE GLOBAL TEMPORARY TABLE EMP
  (TMPDEPTNO  CHAR(3)      NOT NULL,
   TMPDEPTNAME VARCHAR(36)  NOT NULL,
   TMPMGRNO    CHAR(6)       ,
   TMPLOCATION CHAR(16)      );
```

CREATE INDEX

The CREATE INDEX statement creates a partitioning or nonpartitioning index and an index space at the current server. The columns included in the key of the index are columns of a table at the current server.

Invocation

This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is implicitly or explicitly specified.

Authorization

The privilege set that is defined below must include at least one of the following:

- The INDEX privilege on the table
- Ownership of the table
- DBADM authority for the database that contains the table
- SYSADM or SYSCTRL authority

Additional privileges might be required, as explained in the description of the BUFFERPOOL and USING STOGROUP clauses.

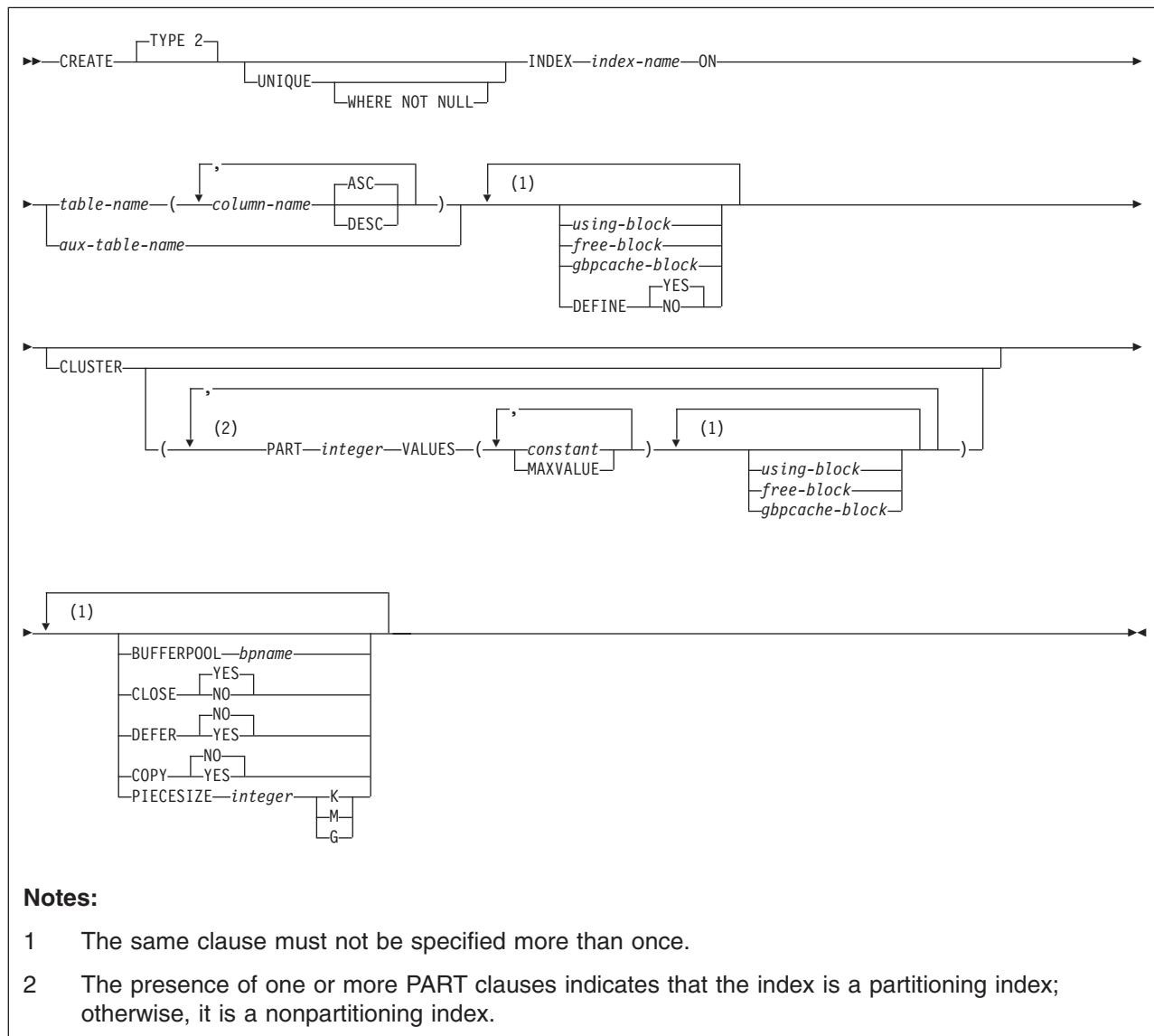
Privilege set: If the statement is embedded in an application program, the privilege set is the privileges that are held by the authorization ID of the owner of the plan or package. If the specified index name includes a qualifier that is not the same as this authorization ID, the privilege set must include SYSADM or SYSCTRL authority, or DBADM or DBCTRL authority for the database.

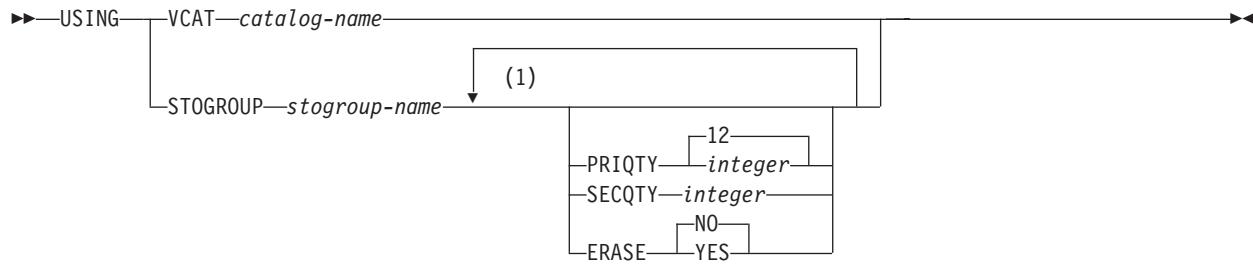
If the statement is dynamically prepared, the privilege set is the privileges that are held by the SQL authorization ID of the process. However, if the specified index name includes a qualifier that is not the same as this authorization ID, the following rules apply:

- If the privilege set includes SYSADM or SYSCTRL authority, or DBADM or DBCTRL authority for the database, any qualifier is valid.
- If the privilege set includes none of these authorities, the qualifier is valid only if it is the same as one of the authorization IDs of the process and the privilege set that are held by that authorization ID includes all privileges needed to create the index. This is an exception to the rule that the privilege set is the privileges that are held by the SQL authorization ID of the process.

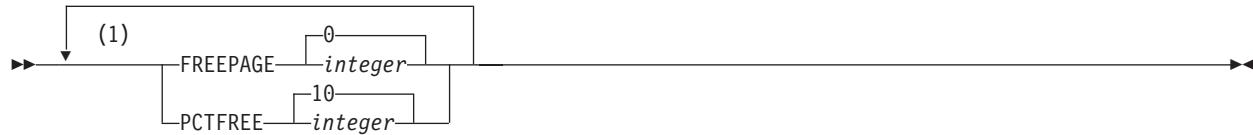
CREATE INDEX

Syntax



using-block:**Notes:**

- 1 The same clause must not be specified more than once.

free-block:**Notes:**

- 1 The same clause must not be specified more than once.

gbpcache-block:

Description

TYPE 2

Specifies a type 2 index. The TYPE 2 clause is not required. A type 2 index is always created.

UNIQUE

Prevents the table from containing two or more rows with the same value of the index key. If any column of the key can contain null values, the meaning of "the same value" is determined by the use or omission of the option WHERE NOT NULL:

- If WHERE NOT NULL is omitted, any two null values are taken to be equal. For example, if the key is a single column, that column can contain no more than one null value.

CREATE INDEX

- If WHERE NOT NULL is used, any two null values are taken to be unequal. If the key is a single column, that column can contain any number of null values, though its other values must be unique.

Unless DEFER YES is specified, the uniqueness constraint is also checked during the execution of the CREATE INDEX statement. If the table already contains rows with duplicate key values, the index is not created. Refer to Part 2 of *DB2 Application Programming and SQL Guide* for more information about using the REBUILD INDEX utility when duplicate keys exist for an index defined with UNIQUE and DEFER YES.

A table requires a unique index if you use the UNIQUE or PRIMARY KEY clause in the CREATE TABLE statement. DB2 implicitly creates those unique indexes if the CREATE TABLE statement is processed by the schema processor; otherwise, you must explicitly create them. If any of the unique indexes that must be explicitly defined do not exist, the definition of the table is incomplete, and the following rules apply:

- Let K denote a key for which a required unique index does not exist and let n denote the number of unique indexes that remain to be created before the definition of the table is complete. (For a new table that has no indexes, K is its primary key or any of the keys defined in the CREATE TABLE statement as UNIQUE and n is the number of such keys. After the definition of a table is complete, an index cannot be dropped if it is enforcing a primary key or unique key.)
- The creation of the unique index reduces n by one if the index key is identical to K. The keys are identical only if they have the same columns in the same order.
- If n is now zero, the creation of the index completes the definition of the table.
- If K is a primary key, the description of the index indicates that it is a primary index. If K is not a primary key, the description of the index indicates that it enforces the uniqueness of a key defined as UNIQUE in the CREATE TABLE statement.

A table also requires a unique index if there is a ROWID column that is defined as GENERATED BY DEFAULT.

INDEX *index-name*

Names the index. The name must not identify an index that exists at the current server.

The associated index space also has a name. That name appears as a qualifier in the names of data sets defined for the index. If the data sets are managed by the user, the name is the same as the second (or only) part of *index-name*. If this identifier consists of more than eight characters, only the first eight are used. The name of the index space must be unique among the names of the index spaces and table spaces of the database for the identified table. If the data sets are defined by DB2, then DB2 derives a unique name.

The qualification rules for an index name depend on the type of table as follows:

- *Index on a base table or auxiliary table.* If the index name is unqualified and the statement is embedded in an application program, the owner of the index is the authorization ID that serves as the implicit qualifier for unqualified object names. This is the authorization ID in the QUALIFIER operand when the plan or package was created or last rebound. If QUALIFIER was not used, the owner of the index is the owner of the package or plan.

If the index name is unqualified and the statement is dynamically prepared, the SQL authorization ID is the owner of the index.

- *Index on a declared temporary table.* The qualifier, if explicitly specified, must be SESSION. If the index name is unqualified, DB2 uses SESSION as the implicit qualifier.

ON *table-name* or *aux-table-name*

Identifies the table on which the index is created. The name can identify a base table, a declared temporary table, or an auxiliary table.

table-name

Identifies the base table or declared temporary table on which the index is created. The name must identify a table that exists at the current server. (The name of a declared temporary table must be qualified with SESSION.) The name must not identify a created temporary table.

(column-name,...)

Specifies the columns of the index key.

Each *column-name* must identify a column of the table. Do not specify more than 64 columns, the same column more than once, or a LOB column (or a column with a distinct type that is based on a LOB data type). Do not qualify *column-name*.

The sum of the length attributes of the columns must not be greater than $255 - n$, where n is the number of columns that can contain null values.

ASC Puts the index entries in ascending order by the column. ASC is the default.

DESC Puts the index entries in descending order by the column.

aux-table-name

Identifies the auxiliary table on which the index is created. The name must identify an auxiliary table that exists at the current server. If the auxiliary table already has an index, do not create another one. An auxiliary table can only have one index.

Do not specify any columns for the index key. The key value is implicitly defined as a unique 19-byte value that is system generated.

If qualified, *table-name* or *aux-table-name* can be a two-part or three-part name. If a three-part name is used, the first part must match the value of the field DB2 LOCATION NAME of installation panel DSNTIPR at the current server. (If the current server is not the local DB2, this name is not necessarily the name in the CURRENT SERVER special register.) Whether the name is two-part or three-part, the authorization ID that qualifies the name is the owner of the index.

The table space that contains the named table must be available to DB2 so that its data sets can be opened. If the table space is EA-enabled, the data sets for the index must be defined to belong to a DFSMS data class that has the extended format and addressability attributes.

using-block

The components of the USING clause are discussed below, first for nonpartitioning indexes and then for partitioning indexes.

Using Clause for Nonpartitioning Indexes

For nonpartitioning indexes, the USING clause indicates whether the data sets for the index are to be managed by the user or managed by DB2. If DB2 definition is specified, the clause also gives space allocation parameters (PRIQTY and SECQTY) and an erase rule (ERASE).

If you omit USING, the data sets will be managed by DB2 on volumes listed in the default storage group of the table's database. That default storage group must exist. With no USING clause, PRIQTY, SECQTY, and ERASE assume their default values.

VCAT *catalog-name*

Specifies that the first data set for the index is managed by the user, and that following data sets, if needed, are also managed by the user.

The data sets defined for the index are linear VSAM data sets cataloged in an integrated catalog facility catalog identified by *catalog-name*. Because *catalog-name* is a short identifier, an alias must be used if the catalog name is longer than eight characters.

Conventions for index data set names are given in Part 2 (Volume 1) of *DB2 Administration Guide*. *catalog-name* is the first qualifier for each data set name.

One or more DB2 subsystems could share integrated catalog facility catalogs with the current server. To avoid the chance of having one of those subsystems attempt to assign the same name to different data sets, select a value for *catalog-name* that is not used by the other DB2 subsystems.

STOGROUP *stogroup-name*

Specifies that DB2 will define and manage the data sets for the index. Each data set will be defined on a volume listed in the identified storage group. The values specified (or the defaults) for PRIQTY and SECQTY determine the primary and secondary allocations for the data set. If $\text{PRIQTY} + 118 \times \text{SECQTY}$ is 2 gigabytes or greater, more than one data set could eventually be used, but only the first is defined during execution of this statement.

To use USING STOGROUP, the privilege set must include SYSADM authority, SYSCTRL authority, or the USE privilege for that storage group. Moreover, *stogroup-name* must identify a storage group that exists at the current server and includes in its description at least one volume serial number. The description can indicate that the choice of volumes will be left to Storage Management Subsystem (SMS). Each volume specified in the storage group must be accessible to MVS for dynamic allocation of the data set, and all these volumes must be of the same device type.

The integrated catalog facility catalog used for the storage group must **not** contain an entry for the first data set of the index. If the catalog is password protected, the description of the storage group must include a valid password.

The storage group supplies the data set name. The first level qualifier is also the name of, or an alias for, the integrated catalog facility catalog on

which the data set is to be cataloged. The naming convention for the data set is the same as if the data set is managed by the user.

PRIQTY integer

Specifies the minimum primary space allocation for a DB2-managed data set. The primary space allocation is at least *n* kilobytes, where *n* is:

- | | |
|----------------|--|
| 12 | If <i>integer</i> is less than 12 or PRIQTY is omitted |
| <i>integer</i> | If <i>integer</i> is between 12 and 4194304 |
| 4194304 | If <i>integer</i> is greater than 4194304 |

If neither the PRIQTY nor SECQTY value is specified and the IXQTY subsystem parameter is nonzero, the IXQTY value is used for the primary and the secondary quantities.

DB2 specifies the primary space allocation to access method services using the smallest multiple of 4KB not less than *n*. The allocated space can be greater than the amount of space requested by DB2. For example, it could be the smallest number of tracks that will accommodate the space requested. To more closely estimate the actual amount of storage, see the description of the DEFINE CLUSTER command in *DFSMS/MVS: Access Method Services for the Integrated Catalog*.

When determining a suitable value for PRIQTY, be aware that two of the pages of the primary space are used by DB2 for purposes other than storing index entries.

SECQTY integer

Specifies the minimum secondary space allocation for a DB2-managed data set. The secondary space allocation is at least *n* kilobytes, where *n* is:

- | | |
|----------------|---|
| 12 | If SECQTY and PRIQTY are omitted |
| 4194304 | If <i>integer</i> is greater than 4194304 |
| <i>integer</i> | If <i>integer</i> is not greater than 4194304 |

If *integer* is 0, no data set for the index can be extended. If you specify PRIQTY and do not specify SECQTY, the default for SECQTY is either 10% of PRIQTY or 3 times the index page size (4KB), whichever is larger. If neither the PRIQTY nor SECQTY value is specified and the IXQTY subsystem parameter is nonzero, the IXQTY value is used for the primary and the secondary quantities.

DB2 specifies the secondary space allocation to access method services using the smallest multiple of 4KB not less than *n*. The allocated space can be greater than the amount of space requested by DB2. For example, it could be the smallest number of tracks that will accommodate the space requested. To more closely estimate the actual amount of storage, see the description of the DEFINE CLUSTER command in *DFSMS/MVS: Access Method Services for the Integrated Catalog*.

ERASE

Indicates whether the DB2-managed data sets are to be erased when they are deleted during the execution of a utility or an SQL statement that drops the index. Refer to *DFSMS/MVS: Access Method Services for the Integrated Catalog* for more information.

NO

Does not erase the data sets. Operations involving data set deletion

#

CREATE INDEX

will perform better than ERASE YES. However, the data is still accessible, though not through DB2. This is the default.

YES

Erases the data sets. As a security measure, DB2 overwrites all data in the data sets with zeros before they are deleted.

USING Clause for Partitioning Indexes:

If the index is partitioning, there is a PART clause for each partition. Within a PART clause, a USING clause is optional. If a USING clause is present, it applies to that partition in the same way that a USING clause for a nonpartitioning index applies to the entire index.

When a USING block is absent from a PART clause, the USING clause parameters for the partition depend on whether a USING clause is specified before the PART clauses.

- If the USING clause is specified, it applies to every PART clause that does not include a USING clause.
- If the USING clause is not specified, the following defaults apply to the partition:
 - Data sets are managed by DB2
 - The default storage group for the database is used
 - A value of 12 is used for PRIQTY and SECQTY
 - A value of NO is used for ERASE

VCAT *catalog-name*

Specifies a user-managed data set with a name that starts with the specified catalog name. You must specify the catalog name in the form of a short identifier. Thus, you must specify an alias if the name of the integrated catalog facility catalog is longer than eight characters.

If *n* is the number of the partition, the identified integrated catalog facility catalog must already contain an entry for the *n*th data set of the index, conforming to the DB2 naming convention for data sets set forth in Part 2 (Volume 1) of *DB2 Administration Guide*.

One or more DB2 subsystems could share integrated catalog facility catalogs with the current server. To avoid the chance of having one of those subsystems attempt to assign the same name to different data sets, select a value for *catalog-name* that is not used by the other DB2 subsystems.

DB2 assumes one and only one data set for each partition.

Do not specify VCAT for an index on a declared temporary table.

|

STOGROUP *stogroup-name*

If USING STOGROUP is used, explicitly or by default, for a partition *n*, DB2 defines the data set for the partition during the execution of the CREATE INDEX statement, using space from the named storage group. The privilege set must include SYSADM authority, SYSCTRL authority, or the USE privilege for that storage group. The integrated catalog facility catalog used for the storage group must NOT contain an entry for the *n*th data set of the index.

stogroup-name must identify a storage group that exists at the current server and the privilege set must include SYSADM authority, SYSCTRL authority, or the USE privilege for the storage group.

If you omit PRIQTY, SECQTY, or ERASE from a USING STOGROUP clause for some partition, their values are given by the next USING STOGROUP clause that governs that partition: either a USING clause that

is not in any PART clause, or a default USING clause. DB2 assumes one and only one data set for each partition.

End of using-block

free-block

FREEPAGE *integer*

Specifies how often to leave a page of free space when index entries are created as the result of executing a DB2 utility or when creating an index for a table with existing rows. One free page is left for every *integer* pages. The value of *integer* can range from 0 to 255. The default is 0, leaving no free pages.

| Do not specify FREEPAGE for an index on a declared temporary table.

PCTFREE *integer*

Determines the percentage of free space to leave in each nonleaf page and leaf page when entries are added to the index or index partition as the result of executing a DB2 utility or when creating an index for a table with existing rows. The first entry in a page is loaded without restriction. When additional entries are placed in a nonleaf or leaf page, the percentage of free space is at least as great as *integer*.

The value of *integer* can range from 0 to 99, however, if a value greater than 10 is specified, only 10 percent of free space will be left in nonleaf pages. The default is 10.

| Do not specify PCTFREE for an index on a declared temporary table.

If the index is partitioning, the values of FREEPAGE and PCTFREE for a particular partition are given by the first of these choices that applies:

- The values of FREEPAGE and PCTFREE given in the PART clause for that partition. Do not use more than one *free-block* in any PART clause.
- The values given in a *free-block* that is not in any PART clause.
- The default values FREEPAGE 0 and PCTFREE 10.

End of free-block

gbpcache-block

GBPCACHE

In a data sharing environment, specifies what index pages are written to the group buffer pool. In a non-data-sharing environment, the option is ignored unless the index is on a declared temporary table. Do not specify GBPCACHE for an index on a declared temporary table in either environment (data sharing or non-data-sharing).

CHANGED

When there is inter-DB2 R/W interest on the index or partition, updated pages are written to the group buffer pool. When there is no inter-DB2 R/W interest, the group buffer pool is not used. Inter-DB2 R/W interest exists when more than one member in the data sharing group has the index or partition open, and at least one member has it open for update. GBPCACHE CHANGED is the default.

CREATE INDEX

If the index is in a group buffer pool that is defined as GBPCACHE(NO), CHANGED is ignored and no pages are cached to the group buffer pool.

ALL

Indicates that pages are to be cached in the group buffer pool as they are read in from DASD.

Exception: In the case of a single updating DB2 when no other DB2s have any interest in the page set, no pages are cached in the group buffer pool.

Hiperpools are not used for indexes or partitions that are defined with GBPCACHE ALL.

If the index is in a group buffer pool that is defined as GBPCACHE(NO), ALL is ignored and no pages are cached to the group buffer pool.

NONE

Indicates that no pages are to be cached to the group buffer pool. DB2 uses the group buffer pool only for cross-validation.

If the index is partitioning, the value of GBPCACHE for a particular partition is given by the first of these choices that applies:

1. The value of GBPCACHE given in the PART clause for that partition. Do not use more than one *gbpcache-block* in any PART clause.
2. The value given in a *gbpcache-block* that is not in any PART clause.
3. The default value is CHANGED.

End of *gbpcache-block*

DEFINE

Specifies when the underlying data sets for the index are physically created.

YES

The data sets are created when the index is created (the CREATE INDEX statement is executed). YES is the default.

NO

The data sets are not created until data is inserted into the index. DEFINE NO is applicable only for DB2-managed data sets (USING STOGROUP is specified). DEFINE NO is ignored for user-managed data sets (USING VCAT is specified). DB2 uses the SPACE column in catalog table SYSINDEXPART to record the status of the data sets (undefined or allocated). DEFINE NO is also ignored if the index is being created on a table that is not empty, or on an auxiliary table.

Do not specify DEFINE NO for an index on a declared temporary table. Do not use DEFINE NO on an index if you use a program outside of DB2 to propagate data into a table on which that index is defined. The DB2 catalog stores information about whether the data sets for an index space have been allocated. If you use DEFINE NO on an index of a table and data is then propagated into the table from a program that is outside of DB2, the index space data sets are allocated, but the DB2 catalog will not reflect this fact. As a result, DB2 acts as if the data sets for the index space have not yet been allocated. The resulting inconsistency causes DB2 to deny application programs access to the data until the inconsistency is resolved.

Use DEFINE NO especially when performance of the CREATE INDEX statement is important or DASD resource is constrained.

CLUSTER

Specifies that the index is the cluster index of the table. Do not use CLUSTER if the index is for an auxiliary table, or if CLUSTER was used in the definition of an existing index on the table. If you do not use CLUSTER, the index is not a cluster index unless it is the first index defined on the table in a nonpartitioned table space. In this case, the first index implicitly serves as the cluster index until CLUSTER is used in the definition of another index on the table.

The implicit or explicit clustering index is ignored when data is inserted into a table space that is defined with MEMBER CLUSTER. Instead of using cluster order, DB2 chooses where to locate the data based on available space. The MEMBER CLUSTER attribute only affects data that is inserted with the INSERT statement; data is always loaded and reorganized in cluster order.

PART *integer*

A PART clause specifies the highest value of the index key in one partition of a partitioning index. In this context, highest means highest in the sorting sequences of the index columns. In a column defined as *ascending* (ASC), highest and lowest have their usual meanings. In a column defined as *descending* (DESC), the lowest actual value is highest in the sorting sequence.

If you use CLUSTER, and the table is contained in a partitioned table space, you must use exactly one PART clause for each partition (defined with NUMPARTS on CREATE TABLESPACE). If there are p partitions, the value of *integer* must range from 1 through p .

The length of the highest value of a partition (also called the limit key) is the same as the length of the partitioning index.

#

VALUES(*constant* or **MAXVALUE**....)

You must use at least one value (*constant* or MAXVALUE) after VALUES in each PART clause. You can use as many values as there are columns in the key. The concatenation of all the values is the highest value of the key in the corresponding partition of the index.

constant

Specifies a constant value with a data type that is the same as its corresponding column. The precision and scale of a decimal constant must not be greater than the precision and scale of its corresponding column. If a string constant is longer or shorter than required by the length attribute of its column, the constant is either truncated or padded on the right to the required length. If the column is ascending, the padding character is X'FF'; if the column is descending, the padding character is X'00'.

#

MAXVALUE

Specifies a value that is greater than the greatest possible value for the data type of the column to which it corresponds (the value is all X'FF', regardless of whether the column is ascending or descending). If all of the columns in the partitioning key are ascending, a constant cannot be specified following MAXVALUE; once MAXVALUE is specified, all subsequent columns must be specified with MAXVALUE.

#

The key values are subject to the following rules:

- The first value corresponds to the first column of the key, the second value to the second column, and so on.
 - If a key includes a ROWID column (or a column with a distinct type that is sourced on a ROWID data type), the values of the ROWID column are

CREATE INDEX

```
# assumed to be in the range of X'000...00' to X'FFF...FF'. Only the first 17
# bytes of the value that is specified for the corresponding ROWID column
# are considered.
#
# • Using fewer values than there are columns in the key has the same
#   effect as using the highest possible values for all omitted columns.
#
# • The highest value of the key in any partition must be lower than the
#   highest value of the key in the next partition.
#
# • The length attributes of the specified index columns must not be greater
#   than 255. This limit is subject to additional restriction that the combination
#   is imposed by the number of partitions and the limit key length cannot
#   exceed 65394:
#
#   (number of partitions) * (106 + limit key size in bytes) < 65394.
#
# • The highest value of the key in the last partition depends on how the
#   table space is defined.

# For table spaces that are created without the LARGE or DSSIZE option,
# the values that you specify after VALUES are not enforced. The highest
# value of the key that can be placed in the table is the highest possible
# value of the key.

# For table spaces that are created with the LARGE or DSSIZE options,
# the values that you specify after VALUES are enforced. The value that is
# specified is the highest value of the key that can be placed in the table.
# Any key values greater than the value specified for the last partition are
# out of range.

# When you define a table space with DSSIZE, you automatically give the
# same size to all indexes that point to that table space.
```

BUFFERPOOL *bpname*

Identifies the buffer pool to be used for the index. The *bpname* must identify an activated 4KB buffer pool and the privilege set must include SYSADM or SYSCTRL authority or the USE privilege for the buffer pool.

The default is the default buffer pool for indexes in the database.

See “Naming conventions” on page 34 for more details about *bpname*. See Chapter 2 of *DB2 Command Reference* for a description of active and inactive buffer pools.

CLOSE

Specifies whether or not the data set is eligible to be closed when the index is not being used and the limit on the number of open data sets is reached.

YES

Eligible for closing. This is the default unless the index is on a declared temporary table.

NO

Not eligible for closing.

If DSMAX is reached and there are no CLOSE YES page sets to close, CLOSE NO page sets will be closed.

For an index on a declared temporary table, DB2 uses CLOSE NO regardless of the value specified.

DEFER

Indicates whether the index is built during the execution of the CREATE INDEX statement. Regardless of the option specified, the description of the index and its index space is added to the catalog. If the table is empty and DEFER YES is

specified, the index is neither built nor placed in a rebuild pending status. Refer to Part 2 (Volume 1) of *DB2 Administration Guide* for more information about using DEFER. Do not specify DEFER for an index on a declared temporary table or an auxiliary table.

NO

The index is built. This is the default.

YES

The index is not built. If the table is populated, the index is placed in a rebuild pending status and a warning message is issued; the index must be rebuilt by the REBUILD INDEX utility.

COPY

Indicates whether the COPY utility is allowed for the index. Do not specify COPY for an index on a declared temporary table.

NO

Does not allow full image or concurrent copies or the use of the RECOVER utility on the index. NO is the default.

YES

Allows full image or concurrent copies and the use of the RECOVER utility on the index.

PIECESIZE *integer*

Specifies the maximum addressability of each piece (data set) for a nonpartitioning index. The subsequent keyword K, M, or G, indicates the units of the value specified in *integer*.

- K** Indicates that the *integer* value is to be multiplied by 1 024 to specify the maximum piece size in bytes. The integer must be a power of two between 256 and 67 108 864.
- M** Indicates that the *integer* value is to be multiplied by 1 048 576 to specify the maximum piece size in bytes. The integer must be a power of two between 1 and 65 536.
- G** Indicates that the *integer* value is to be multiplied by 1 073 741 824 to specify the maximum piece size in bytes. The integer must be a power of two between 1 and 64.

Table 45 on page 614 shows the valid values for piece size, which depend on the size of the table space.

CREATE INDEX

Table 45. Valid values of PIECESIZE clause

K units	M units	G units	Size attribute of table space
256 K	-	-	-
512 K	-	-	-
1024 K	1 M	-	-
2048 K	2 M	-	-
4096 K	4 M	-	-
8192 K	8 M	-	-
16384 K	16 M	-	-
32768 K	32 M	-	-
65536 K	64 M	-	-
131072 K	128 M	-	-
262144 K	256 M	-	-
524288 K	512 M	-	-
1048576 K	1024 M	1 G	-
2097152 K	2048 M	2 G	-
4194304 K	4096 M	4 G	LARGE, DSSIZE 4 G (or greater)
8388608 K	8192 M	8 G	DSSIZE 8 G (or greater)
16777216 K	16384 M	16 G	DSSIZE 16 G (or greater)
33554432 K	32768 M	32 G	DSSIZE 32 G (or greater)
67108864 K	65536 M	64 G	DSSIZE 64 G

As only a specification of the maximum amount of data that a piece can hold and not the actual allocation of storage, PIECESIZE has no effect on primary and secondary space allocation.

The default for piece size is 2 G (2 GB) for indexes that are backed by table spaces that were created without the LARGE or DSSIZE option, and 4 G (4 GB) for indexes that are backed by table spaces that were created with the LARGE or DSSIZE option.

Later, if you change the PIECESIZE value with the ALTER INDEX statement, be aware of the effect on the index, which is put into REBUILD-pending status. See “PIECESIZE” on page 417.

Notes

If DEFER NO is implicitly or explicitly specified, the CREATE INDEX statement cannot be executed while a DB2 utility has control of the table space that contains the identified table.

If the identified table already contains data and if the index build is not deferred, CREATE INDEX creates the index entries for it. If the table does not yet contain data, CREATE INDEX creates a description of the index; the index entries are created when data is inserted into the table.

There are no restrictions on the use of ASC or DESC for the columns of a parent key or foreign key. An index on a foreign key does not have to have the same ascending and descending attributes as the index of the corresponding parent key.

EBCDIC, ASCII, and UNICODE encoding schemes for an index: An index has the same encoding scheme as its associated table.

Choosing a value for PIECESIZE: To choose a value for PIECESIZE, divide the size of the nonpartitioning index by the number of data sets that you want. For example, to ensure that you have 5 data sets for the nonpartitioning index, and your nonpartitioning index is 10 MB (and not likely to grow much), specify PIECESIZE 2 M. If your nonpartitioning index is likely to grow, choose a larger value.

Remember that 32 pieces is the limit if the underlying table space is not defined as LARGE (or as DSSIZE 4G or greater) and that the limit is 254 if the table space is defined as LARGE (or as DSSIZE 4G or greater).

Keep the PIECESIZE value in mind when you are choosing values for primary and secondary quantities. Ideally, the value of your primary quantity plus the secondary quantities should be evenly divisible into PIECESIZE.

Dropping an index: Partitioning indexes can only be dropped by dropping the associated table space. Nonpartitioning indexes that are not indexes on auxiliary tables can be dropped simply by dropping the indexes. An empty index on an auxiliary table can be explicitly dropped; a populated index can be dropped only by dropping other objects. For details, see “Dropping an index on an auxiliary table and an auxiliary table” on page 771.

If the index is a unique index that enforces a primary key, unique key, or referential constraint, the constraint must be dropped before the index is dropped. See “DROP” on page 763.

Creating indexes on DB2 catalog tables: For details on creating indexes on catalog tables, see “SQL statements allowed on the catalog” on page 1011.

EA-enabled index data sets: If an index is created for an EA-enabled table space, the data sets for the index must be set up to belong to a DFSMS data class that has the extended format and extended addressability attributes.

Examples

Example 1: Create a unique index, named DSN8710.XDEPT1, on table DSN8710.DEPT. Index entries are to be in ascending order by the single column DEPTNO. DB2 is to define the data sets for the index, using storage group DSN8G710. Each data set (piece) should hold 1 megabyte of data at most. Use 512 kilobytes as the primary space allocation for each data set and 64 kilobytes as the secondary space allocation. These specifications enable each data set to be extended up to 8 times before a new data set is used—512KB + (8*64KB)= 1024KB.

The data sets can be closed when no one is using the index and do not need to be erased if the index is dropped.

```
CREATE UNIQUE INDEX DSN8710.XDEPT1
  ON DSN8710.DEPT
    (DEPTNO ASC)
  USING STOGROUP DSN8G710
    PRIQTY 512
    SECQTY 64
    ERASE NO
  BUFFERPOOL BP1
  CLOSE YES
  PIECESIZE 1 M;
```

For the above example, the underlying data sets for the index will be created immediately, which is the default (DEFINE YES). Assuming that table DSN8710.DEPT is empty, if you wanted to defer the creation of the data sets until data is first inserted into the index, you would specify DEFINE NO instead of accepting the default behavior.

Example 2: Create a cluster index, named XEMP2, on table EMP in database DSN8710. Put the entries in ascending order by column EMPNO. Let DB2 define

CREATE INDEX

the data sets for each partition using storage group DSN8G710. Make the primary space allocation be 36 kilobytes, and allow DB2 to use the default value for SECQTY, which for this example is 12 kilobytes (3 times 4KB). If the index is dropped, the data sets need not be erased.

There are to be 4 partitions, with index entries divided among them as follows:

- Partition 1: entries up to H99
- Partition 2: entries above H99 up to P99
- Partition 3: entries above P99 up to Z99
- Partition 4: entries above Z99

Associate the index with buffer pool BP1 and allow the data sets to be closed when no one is using the index. Enable the use of the COPY utility for full image or concurrent copies and the RECOVER utility.

```
CREATE INDEX DSN8710.XEMP2
  ON DSN8710.EMP
    (EMPNO ASC)
  USING STOGROUP DSN8G710
    PRIQTY 36
    ERASE NO
    CLUSTER
      (PART 1 VALUES('H99'),
       PART 2 VALUES('P99'),
       PART 3 VALUES('Z99'),
       PART 4 VALUES('999'))
  BUFFERPOOL BP1
  CLOSE YES
  COPY YES;
```

Example 3: Create a nonpartitioning index, named DSN8710.XDEPT1, on table DSN8710.DEPT. Put the entries in ascending order by column DEPTNO. Assume that the data sets are managed by the user with catalog name DSNCAT and each data set (piece) is to hold 1 gigabyte of data at most before the next data set is used.

```
CREATE UNIQUE INDEX DSN8710.XDEPT1
  ON DSN8710.DEPT
    (DEPTNO ASC)
  USING VCAT DSNCAT
  PIECESIZE 1048576 K;
```

Example 4: Assume that a column named EMP_PHOTO with a data type of BLOB(110K) was added to the sample employee table for each employee's photo and auxiliary table EMP_PHOTO_ATAB was created in LOB table space DSN8D71A.PHOTOLTS to store the BLOB data for the column. Create an index named XPHOTO on the auxiliary table. The data sets are to be user-managed with catalog name DSNCAT.

```
CREATE UNIQUE INDEX DSN8710.XPHOTO
  ON DSN8710.EMP_PHOTO_ATAB
  USING VCAT DSNCAT
  COPY YES;
```

In this example, no columns are specified for the key because auxiliary indexes have implicitly generated keys.

CREATE PROCEDURE (external)

The CREATE PROCEDURE statement defines an external stored procedure.

Invocation

This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is specified implicitly or explicitly.

Authorization

The privilege set that is defined below must include at least one of the following:

- The CREATEIN privilege for the schema or all schemas
- SYSADM or SYSCTRL authority

The authorization ID that matches the schema name implicitly has the CREATEIN privilege on the schema.

Privilege set: If the statement is embedded in an application program, the privilege set is the privileges that are held by the authorization ID of the owner of the plan or package.

If the statement is dynamically prepared, the privilege set is the privileges that are held by the SQL authorization ID of the process. The specified procedure name can include a schema name (a qualifier). However, if the schema name is not the same as the SQL authorization ID, one of the following conditions must be met:

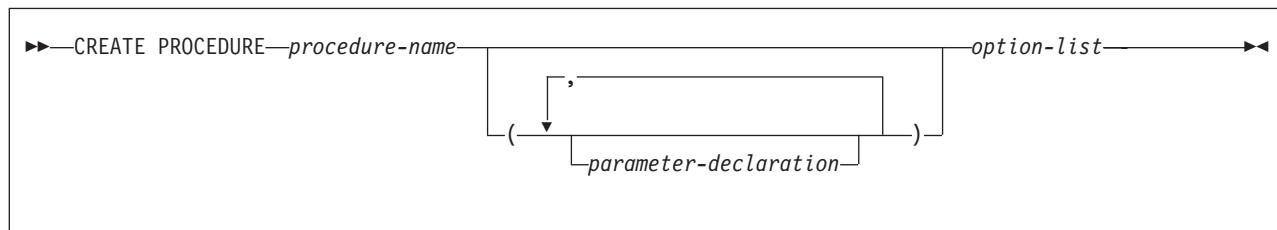
- The privilege set includes SYSADM or SYSCTRL authority.
- The SQL authorization ID of the process has the CREATEIN privilege on the schema.

The authorization ID that is used to create the stored procedure must have authority to create programs that are to be run either in the DB2-established stored procedure address space or the specified workload manager (WLM) environment. In addition, if the stored procedure uses a distinct type as a parameter, this authorization ID must have the USAGE privilege on each distinct type that is a parameter.

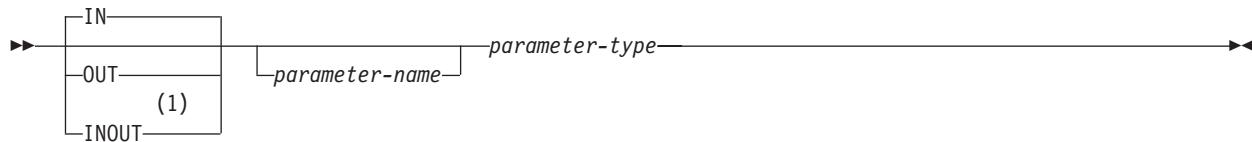
| When LANGUAGE is JAVA and a *jar-name* is specified in the EXTERNAL NAME clause, the privilege set must include USAGE on the JAR, the Java ARchive file.
|

CREATE PROCEDURE (external)

Syntax



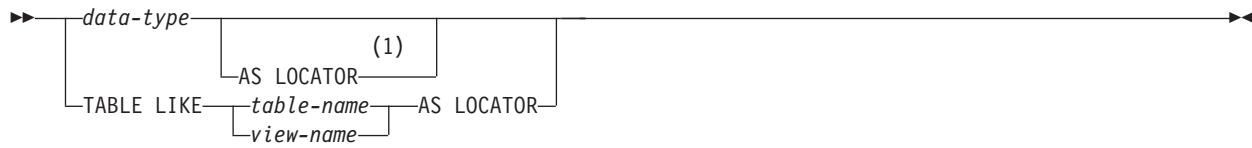
parameter-declaration:



Notes:

- 1 For a REXX stored procedure, only one parameter can have type OUT or INOUT. That parameter must be declared last.

parameter-type:

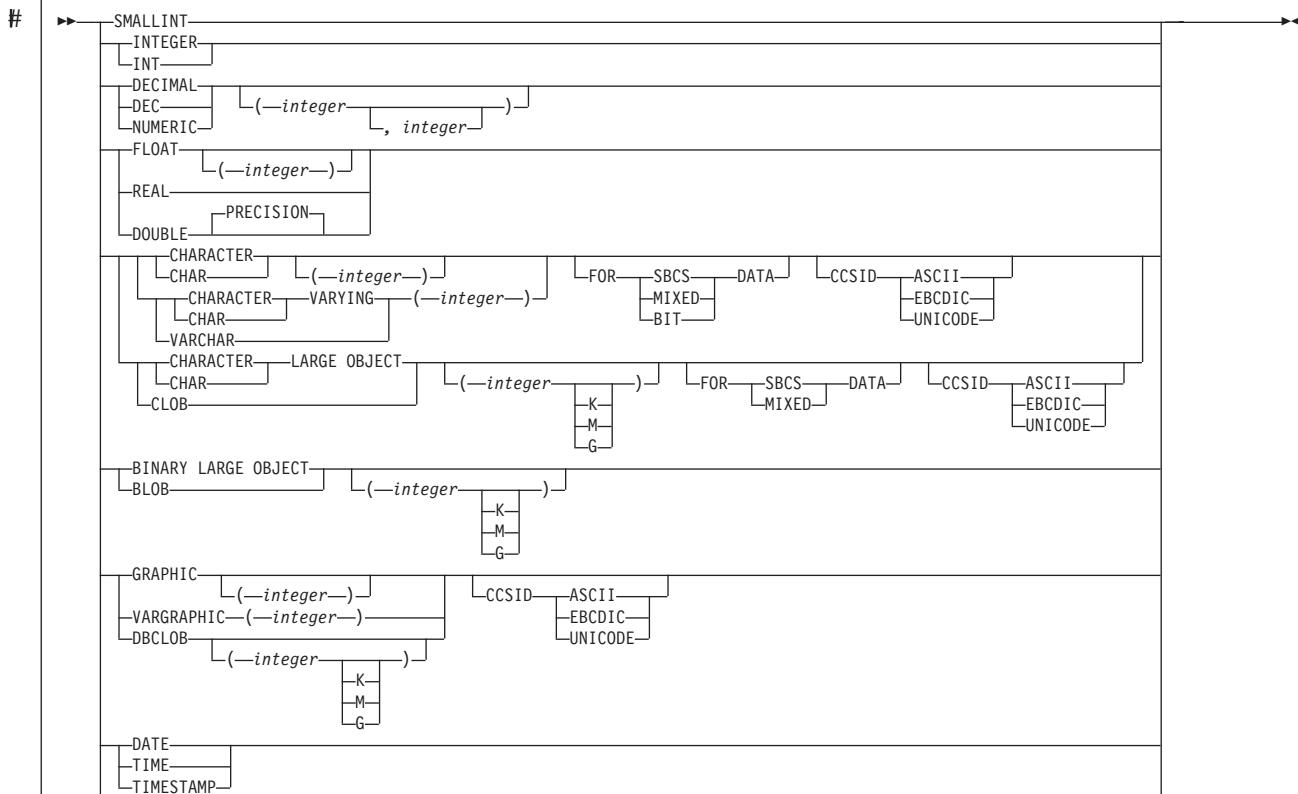


Notes:

- 1 AS LOCATOR can be specified only for a LOB data type or a distinct type based on a LOB data type.

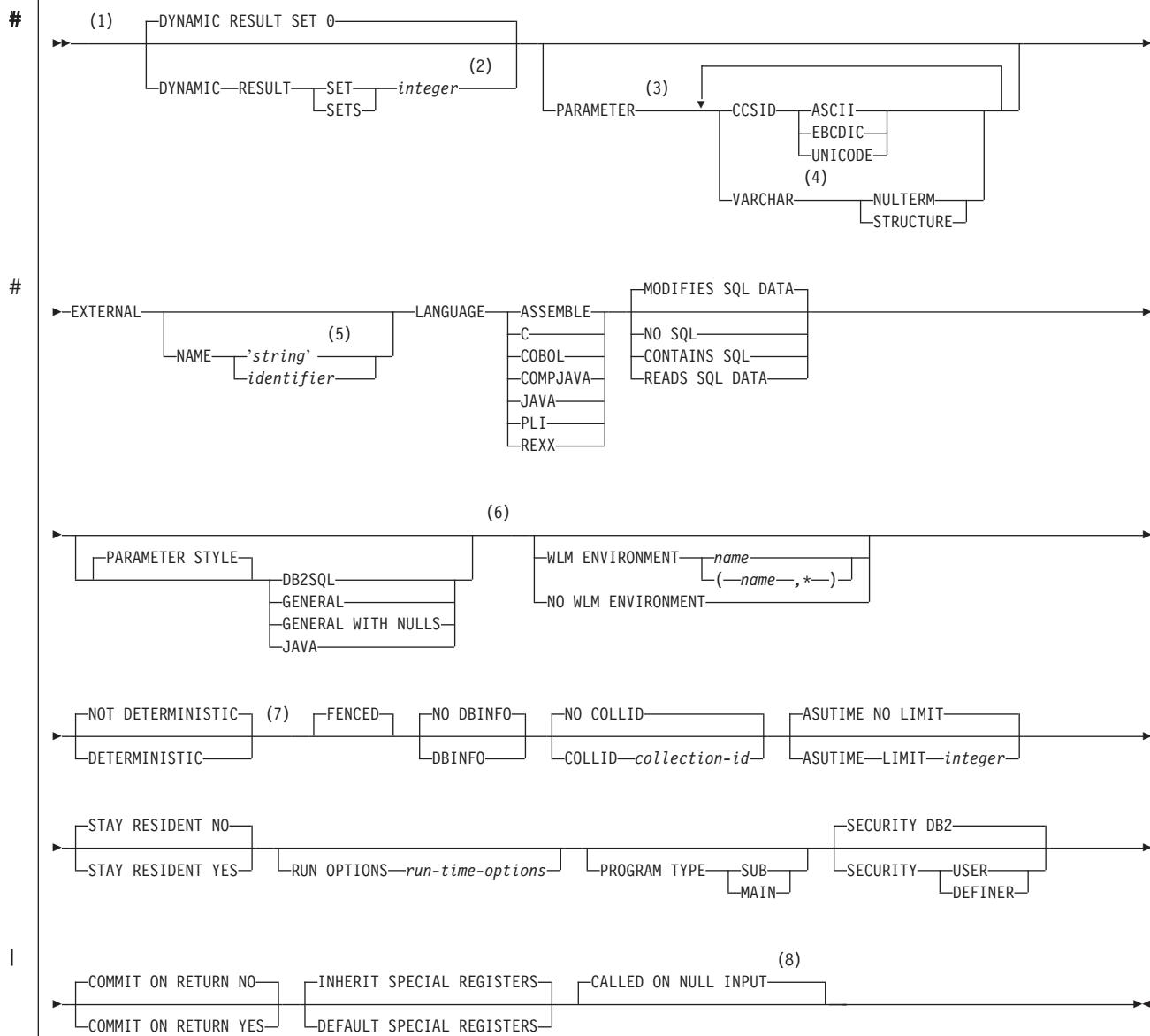
data-type:



built-in-data-type:

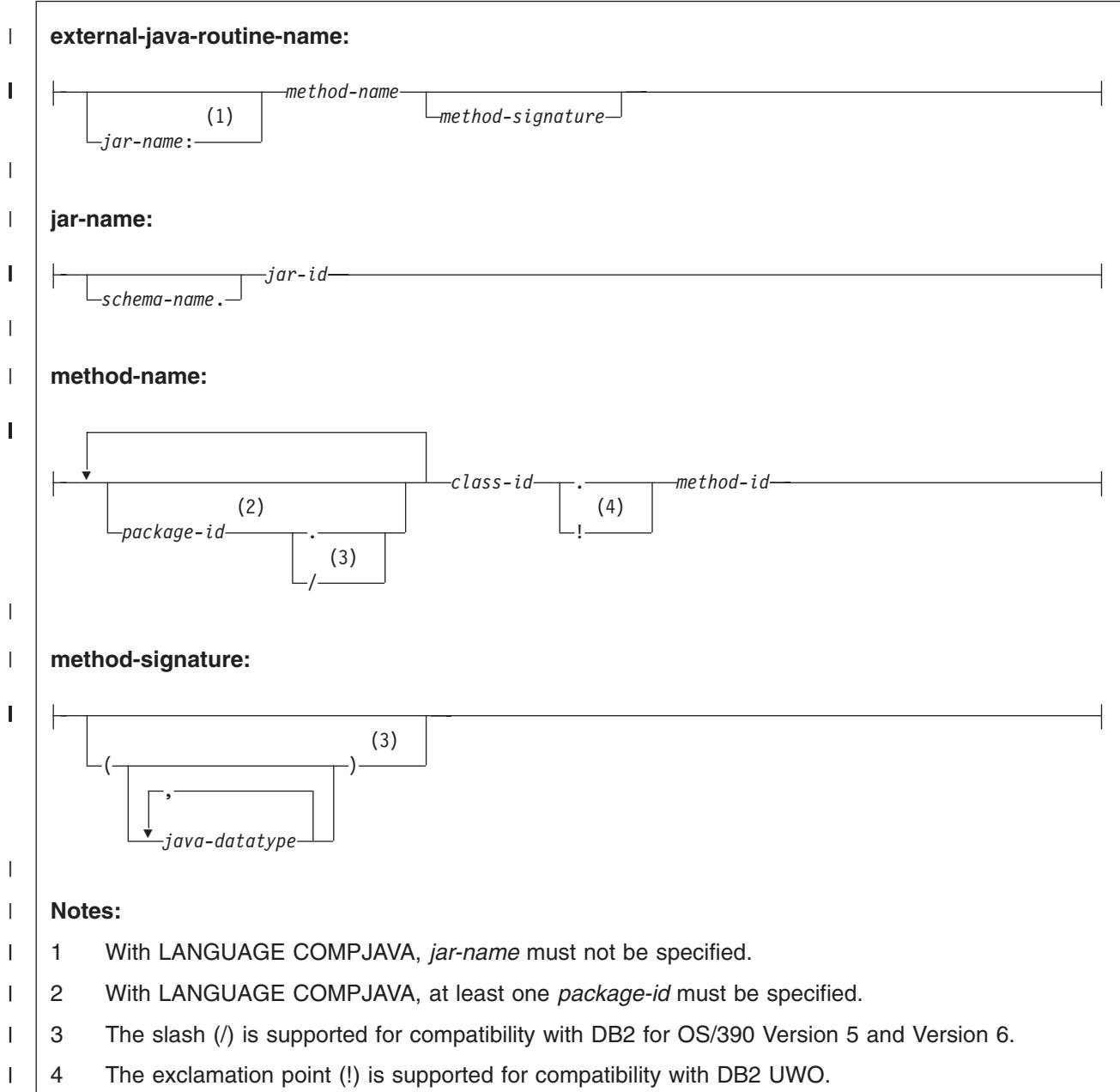
CREATE PROCEDURE (external)

option-list:



Notes:

- 1 The clauses in the *option list* can be specified in any order.
- 2 Synonyms include RESULT SET(S) for DYNAMIC RESULT SET(S).
- 3 The same clause must not be specified more than once.
- 4 The VARCHAR clause can only be specified if LANGUAGE C is specified.
- 5 With LANGUAGE COMPJAVA or JAVA, use a valid *external-java-routine-name*.
- 6 Synonyms include STANDARD CALL, SIMPLE CALL, and SIMPLE CALL WITH NULLS.
- 7 Synonyms include VARIANT for NOT DETERMINISTIC and NOT VARIANT for DETERMINISTIC.
- 8 Synonyms include NULL CALL for CALLED ON NULL INPUT.



Description

procedure-name

Names the stored procedure. The name cannot be a single asterisk even if you specify it as a delimited identifier ("*").

The name is implicitly or explicitly qualified by a schema. The name, including the implicit or explicit qualifier, must not identify an existing stored procedure at the current server.

- The unqualified form of *procedure-name* is a long SQL identifier. The unqualified name is implicitly qualified with a schema name according to the following rules:

CREATE PROCEDURE (external)

If the statement is embedded in a program, the schema name is the authorization ID in the QUALIFIER bind option when the plan or package was created or last rebound. If QUALIFIER was not specified, the schema name is the owner of the plan or package.

If the statement is dynamically prepared, the schema name is the SQL authorization ID in the CURRENT SQLID special register.

- The qualified form of *procedure-name* is a short SQL identifier (the schema name) followed by a period and a long SQL identifier.

The schema name must not begin with 'SYS' unless the schema name is 'SYSPROC' or 'SYSADM'. The schema name can be SYSIBM if the privilege set includes SYSADM or SYSCTRL authority.

The owner of the procedure is determined by how the CREATE PROCEDURE statement is invoked:

- If the statement is embedded in a program, the owner is the authorization ID of the owner of the plan or package.
- If the statement is dynamically prepared, the owner is the SQL authorization ID in the CURRENT SQLID special register.

The owner is implicitly given the EXECUTE privilege with the GRANT option for the procedure.

(parameter-declaration,...)

Specifies the number of parameters of the stored procedure and the data type of each parameter. All of the parameters are nullable. A parameter for a stored procedure can be used only for input, only for output, or for both input and output. If an error is returned by the procedure, OUT parameters are undefined, and INOUT parameters are unchanged. Although not required, you can give each parameter a name.

IN Identifies the parameter as an input parameter to the stored procedure. The parameter does not contain a value when the stored procedure returns control to the calling SQL application.

IN is the default.

OUT

Identifies the parameter as an output parameter that is returned by the stored procedure.

INOUT

Identifies the parameter as both an input and output parameter for the stored procedure.

parameter-name

Names the parameter. *parameter-name* is a long identifier.

data-type

Specifies the data type of the parameter. The data type can be a built-in data type or a distinct type.

built-in-data-type

The data type of the parameter is a built-in data type. You can use the same built-in data types as for the CREATE TABLE statement except LONG VARCHAR or LONG VARGRAPHIC. Use VARCHAR or VARGRAPHIC with an explicit length instead.

If you do not specify a specific value for the data types that have length, precision, or scale attributes (CHAR, GRAPHIC, DECIMAL, NUMERIC, FLOAT), the defaults are as follows:

CHAR	CHAR(1)
GRAPHIC	GRAPHIC(1)
DECIMAL	DECIMAL(5,0)
FLOAT	DOUBLE (length of 8)

For more information on the data types, including the subtype of character data types (the FOR *subtype* DATA clause), see “built-in-data-type” on page 658.

For parameters with a string data type, the CCSID clause indicates whether the encoding scheme of the parameter value is ASCII, EBCDIC, or UNICODE. If you do not specify CCSID ASCII, CCSID EBCDIC, or CCSID UNICODE, the encoding scheme is the value of field DEF ENCODING SCHEME on installation panel DSNTIPF.

distinct-type-name

The data type of the input parameter is a distinct type. Any length, precision, scale, subtype, or encoding scheme attributes for the parameter are those of the source type of the distinct type.

If you specify the name of the distinct type without a schema name, DB2 resolves the schema name by searching the schemas in the SQL path.

Although an input parameter with a character data type has an implicitly or explicitly specified subtype (BIT, SBCS, or MIXED), the value that is actually passed in the input argument on the CALL statement can have any subtype. Therefore, conversion of the input data to the subtype of the parameter might occur when the procedure is called. With ASCII or EBCDIC, an error occurs if mixed data that actually contains DBCS characters is used as the value for an input parameter that is declared with an SBCS subtype.

Parameters with a datetime data type or a distinct type are passed to the function as a different data type:

- A datetime type parameter is passed as a character data type, and the data is passed in ISO format.

The encoding scheme for a datetime type parameter is the same as the implicitly or explicitly specified encoding scheme of any character or graphic string parameters. If no character or graphic string parameters are passed, the encoding scheme is the value of field DEF ENCODING SCHEME on installation panel DSNTIPF.

- A distinct type parameter is passed as the source type of the distinct type.

AS LOCATOR

Specifies that a locator to the value of the parameter is passed to the procedure instead of the actual value. Specify AS LOCATOR only for parameters with a LOB data type or a distinct type based on a LOB data type. Passing locators instead of values can result in fewer bytes being passed to the procedure, especially when the value of the parameter is very large.

The AS LOCATOR clause has no effect on determining whether data types can be promoted.

CREATE PROCEDURE (external)

TABLE LIKE *table-name* or *view-name* AS LOCATOR

Specifies that the parameter is a transition table. However, when the procedure is called, the actual values in the transition table are not passed to the stored procedure. A single value is passed instead. This single value is a locator to the table, which the procedure uses to access the columns of the transition table. A procedure with a table parameter can only be invoked from the triggered action of a trigger.

The use of TABLE LIKE provides an implicit definition of the transition table. It specifies that the transition table has the same number of columns as the identified table or view. The columns have the same data type, length, precision, scale, subtype, and encoding scheme as the identified table or view, as they are described in catalog tables SYSCOLUMNS and SYSTABLESPACES.

The *name* specified after TABLE LIKE must identify a table or view that exists at the current server. The name must not identify a declared temporary table. The name does not have to be the same name as the table that is associated with the transition table for the trigger. An unqualified table or view name is implicitly qualified according to the following rules:

- If the CREATE PROCEDURE statement is embedded in a program, the implicit qualifier is the authorization ID in the QUALIFIER bind option when the plan or package was created or last rebound. If QUALIFIER was not used, the implicit qualifier is the owner of the plan or package.
- If the CREATE PROCEDURE statement is dynamically prepared, the implicit qualifier is the SQL authorization ID in the CURRENT SQLID special register.

When the procedure is called, the corresponding columns of the transition table identified by the table locator and the table or view identified in the TABLE LIKE clause must have the same definition. The data type, length, precision, scale, and encoding scheme of these columns must match exactly. The description of the table or view at the time the CREATE PROCEDURE statement was executed is used.

Additionally, a character FOR BIT DATA column of the transition table cannot be passed as input for a table parameter for which the corresponding column of the table specified at the definition is not defined as character FOR BIT DATA. (The definition occurs with the CREATE PROCEDURE statement.) Likewise, a character column of the transition table that is not FOR BIT DATA cannot be passed as input for a table parameter for which the corresponding column of the table specified at the definition is defined as character FOR BIT DATA.

For more information about using table locators, see *DB2 Application Programming and SQL Guide*.

DYNAMIC RESULT SET *integer* or DYNAMIC RESULT SETS *integer*

Specifies the maximum number of query result sets that the stored procedure can return. The default is DYNAMIC RESULT SETS 0, which indicates that there are no result sets. The value must be between 0 and 32767.

PARAMETER CCSID or VARCHAR

Specifies the encoding scheme for string parameters, and in the case of LANGUAGE C, specifies the representation of variable length string parameters.

CCSID

Indicates whether the encoding scheme for string parameters is ASCII, EBCDIC, or UNICODE. The default encoding scheme is the value specified

in the CCSID clauses of the parameter list or in the field DEF ENCODING SCHEME on installation panel DSNTIPF.

This clause provides a convenient way to specify the encoding scheme for
all string parameters. If individual CCSID clauses are specified for individual
parameters in addition to this PARAMETER CCSID clause, the value
specified in all of the CCSID clauses must be the same value that is
specified in this clause.

This clause also specifies the encoding scheme to be used for
system-generated parameters of the routine such as message tokens and
DBINFO.

VARCHAR

Specifies that the representation of the values of varying length character
string-parameters for procedures that specify LANGUAGE C.

This option can only be specified if LANGUAGE C is also specified or
SQLCODE -628, SQLSTATE 42613 is returned.

NULTERM

Specifies that variable length character string parameters are
represented in a NUL-terminated string form.

STRUCTURE

Specifies that variable length character string parameters are
represented in a VARCHAR structure form.

Using the PARAMETER VARCHAR clause, there is no way to specify the
VARCHAR form of an individual parameter as these is with PARAMETER
CCSID. The PARAMETER VARCHAR clause only applies to parameters in
a procedure's parameter list and in the RETURNS clause. It does not apply
to system-generated parameters of the routine such as message tokens
and DBINFO.

In a data sharing environment, you should not specify the PARAMETER
VARCHAR clause until all members of the data sharing group support the
clause. If some group members support this clause and others do not, and
PARAMETER VARCHAR is specified in an external routine, the routine will
encounter different parameter forms depending on which group member
invokes the routine.

EXTERNAL

Specifies the program that runs when the procedure name is specified in a CALL statement.

The program does not need to exist when the CREATE PROCEDURE statement is executed. However, it must exist and be accessible by the current server when a CALL statement to the stored procedure is issued.

You can specify the EXTERNAL clause in one of the following ways:

EXTERNAL

EXTERNAL NAME PKJVSP1

EXTERNAL NAME 'PKJVSP1'

If you specify an external program name, you must use the NAME keyword. For example, this syntax is not valid:

EXTERNAL PKJVSP1

CREATE PROCEDURE (external)

NAME '*string*' or *identifier*

Identifies the user-written code that implements the stored procedure.

If LANGUAGE is COMPJAVA or JAVA, '*string*' must be specified and enclosed in single quotation marks, with no extraneous blanks within the single quotation marks. It must specify a valid *external-java-routine-name*. If multiple '*string*'s are specified, the total length of all of them must not be greater than 1305 bytes and they must be separated by a space or a line break. Do not specify a JAR for a JAVA procedure for which NO SQL is also specified.

An *external-java-routine-name* contains the following parts:

jar-name

Identifies the name given to the JAR when it was installed in the database. The name contains *jar-id*, which can optionally be qualified with a schema. Examples are "myJar" and "mySchema.myJar." The unqualified *jar-id* is implicitly qualified with a schema name according to the following rules:

- If the statement is embedded in a program, the schema name is the authorization ID in the QUALIFIER bind option when the package or plan was created or last rebound. If the QUALIFIER was not specified, the schema name is the owner of the package or plan.
- If the statement is dynamically prepared, the schema name is the SQL authorization ID in the CURRENT SQLID special register.

If *jar-name* is specified, it must exist when the CREATE PROCEDURE statement is processed. With LANGUAGE COMPJAVA, a *jar-name* must not be specified.

If *jar-name* is not specified, the procedure is loaded from the class file directly instead of being loaded from a JAR file. DB2 for DB2 for OS/390 and z/OS searches the directories in the CLASSPATH associated with the WLM Environment. Environmental variables for Java routines are specified in a data set identified in a JAVAENV DD card on the JCL used to start the address space for a WLM-managed stored procedure.

method-name

Identifies the name of the method and must not be longer than 254 bytes. Its package, class, and method IDs are specific to Java and as such are not limited to 18 bytes. In addition, the rules for what these can contain are not necessarily the same as the rules for an SQL ordinary identifier.

package-id

Identifies the package list that the class identifier is part of. If the class is part of a package, the method name must include the complete package prefix, such as "myPacks.StoredProcs." The Java virtual machine looks in the directory "/myPacks/StoredProcs/" for the classes. With LANGUAGE COMPJAVA, at least one *package-id* must be specified.

class-id

Identifies the class identifier of the Java object.

method-id

Identifies the method identifier with the Java class to be invoked.

method-signature

Identifies a list of zero or more Java data types for the parameter list and must not be longer than 1024 bytes. Specify the *method-signature* if the procedure involves any input or output parameters that can be NULL. When the stored procedure being created is called, DB2 searches for a Java method with the exact *method-signature*. The number of *java-datatype* elements specified indicates how many parameters that the Java method must have.

A Java procedure can have no parameters. In this case, you code an empty set of parentheses for *method-signature*. If a Java *method-signature* is not specified, DB2 searches for a Java method with a signature derived from the default JDBC types associated with the SQL types specified in the parameter list of the CREATE PROCEDURE statement.

For other values of LANGUAGE, the name can be a string constant that is no longer than 8 characters or a short identifier. It must conform to the naming conventions for MVS load modules. Alphabetical extenders for national languages can be used as the first character and as subsequent characters in the load module name.

If you do not specify the NAME clause, *NAME procedure-name* is implicit. In this case procedure-name must not be longer than 8 characters, and LANGUAGE must not be COMPJAVA or JAVA.

LANGUAGE

Specifies the application programming language in which the stored procedure is written. Assembler, C, COBOL, and PL/I programs must be designed to run in IBM's Language Environment.

ASSEMBLE

The stored procedure is written in Assembler.

C The stored procedure is written in C or C++.**COBOL**

The stored procedure is written in COBOL, including the OO-COBOL language extensions.

COMPJAVA

The stored procedure is written in Java and the Java byte code has been bound into a PDSE member using the Visual Age for Java ET/390 byte code binder. When LANGUAGE COMPJAVA is specified, the EXTERNAL NAME clause must also be specified with a valid *external-java-routine-name*.

JAVA

The stored procedure is written in Java byte code and is executed in the OS/390 Java Virtual Machine. When LANGUAGE JAVA is specified, the EXTERNAL NAME clause must be specified with a valid *external-java-routine-name* and PARAMETER STYLE must be specified with JAVA.

Do not specify LANGUAGE JAVA when DBINFO, NO WLM ENVIRONMENT, PROGRAM TYPE MAIN, or RUN OPTIONS is specified.

PLI

The stored procedure is written in PL/I.

CREATE PROCEDURE (external)

REXX

The stored procedure is written in REXX. Do not specify LANGUAGE REXX when PARAMETER STYLE DB2SQL or NO WLM ENVIRONMENT is specified. When REXX is specified, the procedure must use PARAMETER STYLE GENERAL or GENERAL WITH NULLS.

NO SQL, MODIFIES SQL DATA, READS SQL DATA, or CONTAINS SQL DATA

Indicates whether the stored procedure can execute any SQL statements and, if so, what type. See Table 72 on page 972 for a detailed list of the SQL statements that can be executed under each data access indication.

NO SQL

The stored procedure cannot execute any SQL statements. Do not specify NO SQL for a JAVA procedure that uses a JAR.

MODIFIES SQL DATA

The stored procedure can execute any SQL statement except those statements that are not supported in any stored procedure.

MODIFIES SQL DATA is the default.

READS SQL DATA

The stored procedure cannot execute SQL statements that modify data. SQL statements that are not supported in any stored procedure return a different error.

CONTAINS SQL

The stored procedure cannot execute any SQL statements that read or modify data. SQL statements that are not supported in any stored procedure return a different error.

PARAMETER STYLE

Identifies the linkage convention used to pass parameters to the stored procedure. All of the linkage conventions provide arguments to the stored procedure that contain the parameters specified on the CALL statement. Some of the linkage conventions pass additional arguments to the stored procedure that provide more information to the stored procedure. For more information on linkage conventions, see *DB2 Application Programming and SQL Guide*.

DB2SQL

In addition to the parameters on the CALL statement, the following arguments are also passed to the stored procedure:

- A null indicator for each parameter on the CALL statement
- The SQLSTATE to be returned to DB2
- The qualified name of the stored procedure
- The specific name of the stored procedure
- The SQL diagnostic string to be returned to DB2

If DBINFO is specified, an additional parameter, the DB2INFO structure, might also be passed.

DB2SQL is the default unless LANGUAGE is COMPJAVA. Do not specify DB2SQL when LANGUAGE REXX is specified.

GENERAL

Only the parameters on the CALL statement are passed to the stored procedure. The parameters cannot be null.

GENERAL WITH NULLS

In addition to the parameters on the CALL statement, another argument is also passed to the stored procedure. The additional argument contains a

vector of null indicators for each of the parameters on the CALL statement that enables the stored procedure to accept or return null parameter values.

JAVA

The stored procedure uses a convention for passing parameters that conforms to the Java and SQLJ specifications. PARAMETER JAVA can be specified only if LANGUAGE is COMPJAVA or JAVA. If LANGUAGE is COMPJAVA, PARAMETER STYLE defaults to JAVA. JAVA must be specified for PARAMETER STYLE when LANGUAGE is JAVA.

INOUT and OUT parameters are passed as single-entry arrays. The INOUT and OUT parameters are declared in the Java method as single-element arrays of the Java type.

For REXX stored procedures (LANGUAGE REXX), GENERAL and GENERAL WITH NULLS are the only valid values for PARAMETER STYLE; therefore, specify one of these values and do not allow PARAMETER STYLE to default to DB2SQL.

WLM ENVIRONMENT

Identifies the MVS workload manager (WLM) environment in which the stored procedure is to run when the DB2 stored procedure address space is WLM-established. The *name* of the WLM environment is a long identifier.

If you do not specify WLM ENVIRONMENT, the stored procedure runs in the default WLM-established stored procedure address space specified at installation time.

name

The WLM environment in which the stored procedure must run. If another stored procedure or a user-defined function calls the stored procedure and that calling routine is running in an address space that is not associated with the specified WLM environment, DB2 routes the stored procedure request to a different MVS address space.

(*name*, *)

When an SQL application program directly calls a stored procedure, the WLM environment in which the stored procedure runs.

If another stored procedure or a user-defined function calls the stored procedure, the stored procedure runs in the same WLM environment that the calling routine uses.

To define a stored procedure that is to run in a specified WLM environment, you must have appropriate authority for the WLM environment. For an example of a RACF command that provides this authorization, see “Running stored procedures” on page 634.

NO WLM ENVIRONMENT

Indicates that the stored procedure is to run in the DB2-established stored procedure address space.

Do not specify NO WLM ENVIRONMENT if you implicitly or explicitly define the stored procedure with any of the following clauses or parameters:

- The PROGRAM TYPE SUB clause
- The SECURITY USER or SECURITY DEFINER clause
- The LANGUAGE REXX, LANGUAGE COMPJAVA, or JAVA clause
- Parameters with a LOB data type or a distinct type based on a LOB data type

CREATE PROCEDURE (external)

To define a stored procedure that is to run in the DB2-established stored procedure address space, you must have appropriate authority for the address space. For an example of a RACF command that provides this authorization, see “Running stored procedures” on page 634.

NOT DETERMINISTIC or DETERMINISTIC

Specifies whether the stored procedure returns the same result from successive calls with identical input arguments.

NOT DETERMINISTIC

The stored procedure might not return the same result from successive calls with identical input arguments. NOT DETERMINISTIC is the default.

DETERMINISTIC

The stored procedure returns the same result from successive calls with identical input arguments.

DB2 does not verify that the stored procedure code is consistent with the specification of DETERMINISTIC or NOT DETERMINISTIC.

FENCED

Specifies that the stored procedure runs in an external address space to prevent user programs from corrupting DB2 storage.

FENCED is the default.

NO DBINFO or DBINFO

Specifies whether specific information known by DB2 is passed to the stored procedure when it is invoked.

NO DBINFO

Additional information is not passed. NO DBINFO is the default.

DBINFO

An additional argument is passed when the stored procedure is invoked. The argument is a structure that contains information such as the name of the current server, the application run-time authorization ID, and identification of the version and release of the database manager that invoked the procedure. For details about the argument and its structure, see *DB2 Application Programming and SQL Guide*.

DBINFO can be specified only if PARAMETER STYLE DB2SQL is specified.

NO COLLID or COLLID *collection-id*

Identifies the package collection that is to be used when the stored procedure is executed. This is the package collection into which the DBRM that is associated with the stored procedure is bound.

NO COLLID

The package collection for the stored procedure is the same as the package collection of the calling program. If the calling program does not use a package, the package collection is set to the value of special register CURRENT PACKAGESET.

NO COLLID is the default.

COLLID *collection-id*

The package collection for the stored procedure is the one specified.

For REXX stored procedures, *collection-id* can be DSNREXRR, DSNREXRS, DSNREXCR, or DSNREXCS.

#

ASUTIME

Specifies the total amount of processor time, in CPU service units, that a single invocation of a stored procedure can run. The value is unrelated to the ASUTIME column of the resource limit specification table.

When you are debugging a stored procedure, setting a limit can be helpful in case the stored procedure gets caught in a loop. For information on service units, see *OS/390 MVS Initialization and Tuning Guide*.

NO LIMIT

There is no limit on the service units. NO LIMIT is the default.

LIMIT *integer*

The limit on the service units is a positive *integer* in the range of 1 to 2G. If the stored procedure uses more service units than the specified value, DB2 cancels the stored procedure.

STAY RESIDENT

Specifies whether the stored procedure load module is to remain resident in memory when the stored procedure ends.

NO

The load module is deleted from memory after the stored procedure ends. NO is the default.

YES

The load module remains resident in memory after the stored procedure ends.

RUN OPTIONS *run-time-options*

Specifies the Language Environment run-time options to be used for the stored procedure. For a REXX stored procedure, specifies the Language Environment run-time options to be passed to the REXX language interface to DB2. You must specify *run-time-options* as a character string that is no longer than 254 bytes. If you do not specify RUN OPTIONS or pass an empty string, DB2 does not pass any run-time options to Language Environment, and Language Environment uses its installation defaults.

|
| Do not specify RUN OPTIONS when LANGUAGE COMPJAVA or LANGUAGE JAVA is specified.

| For a description of the Language Environment run-time options, see *OS/390 Language Environment for OS/390 & VM Programming Reference*.

PROGRAM TYPE

Specifies whether the stored procedure runs as a main routine or a subroutine. The default is always PROGRAM TYPE SUB when LANGUAGE JAVA is specified.

SUB

The stored procedure runs as a subroutine.

With LANGUAGE JAVA, PROGRAM TYPE SUB is the only valid option. Do
not specify PROGRAM TYPE SUB when NO WLM ENVIRONMENT is
specified.

MAIN

| The stored procedure runs as a main routine. With LANGUAGE REXX,
| PROGRAM TYPE MAIN is always in effect.

The default for PROGRAM TYPE is:
• MAIN with LANGUAGE REXX

CREATE PROCEDURE (external)

- ```
• SUB with LANGUAGE JAVA
• For other languages, the default depends on the value of the CURRENT
RULES special register:
– MAIN when the value is DB2
– SUB when the value is STD
```

### **SECURITY**

Specifies how the stored procedure interacts with an external security product, such as RACF, to control access to non-SQL resources.

#### **DB2**

The stored procedure does not require a special external security environment. If the stored procedure accesses resources that an external security product protects, the access is performed using the authorization ID associated with the stored procedure address space. DB2 is the default. SECURITY DB2 is the only valid choice when NO WLM ENVIRONMENT is specified.

#### **USER**

An external security environment should be established for the stored procedure. If the stored procedure accesses resources that the external security product protects, the access is performed using the authorization ID of the user who invoked the stored procedure. Do not specify SECURITY USER when NO WLM ENVIRONMENT is specified.

#### **DEFINER**

An external security environment should be established for the stored procedure. If the stored procedure accesses resources that the external security product protects, the access is performed using the authorization ID of the owner of the stored procedure. Do not specify SECURITY DEFINER when NO WLM ENVIRONMENT is specified.

### **COMMIT ON RETURN**

Indicates whether DB2 commits the transaction immediately on return from the stored procedure.

#### **NO**

DB2 does not issue a commit when the stored procedure returns. NO is the default.

#### **YES**

DB2 issues a commit when the stored procedure returns if the following statements are true:

- The SQLCODE that is returned by the CALL statement is not negative.
- The stored procedure is not in a must abort state.

The commit operation includes the work that is performed by the calling application process and the stored procedure.

If the stored procedure returns result sets, the cursors that are associated with the result sets must have been defined as WITH HOLD to be usable after the commit.

### **INHERIT SPECIAL REGISTERS**

Indicates that the values of special registers are inherited according to the rules listed in the table for characteristics of special registers in a stored procedure function in Table 19 on page 93.

### **DEFAULT SPECIAL REGISTERS**

Indicates that special registers are initialized to the default values, as

indicated by the rules in the table for characteristics of special registers in a stored procedure function in Table 19 on page 93.

#### CALLED ON NULL INPUT

Specifies that the stored procedure will be called even if any of the input arguments is null, making the procedure responsible for testing for null argument values. The result is the null value. CALLED ON NULL INPUT is the default.

### Notes

**Choosing data types for parameters:** When you choose the data types of the parameters for your stored procedure, consider the rules of promotion that can affect the values of the parameters. (See “Promotion of data types” on page 61). For example, a constant that is one of the input arguments to the stored procedure might have a built-in data type that is different from the data type that the procedure expects, and more significantly, might not be promotable to that expected data type. Based on the rules of promotion, using the following data types for parameters is recommended:

- INTEGER instead of SMALLINT
- DOUBLE instead of REAL
- VARCHAR instead of CHAR
- VARGRAPHIC instead of GRAPHIC

For portability of functions across platforms that are not DB2 for OS/390 and z/OS, do not use the following data types, which might have different representations on different platforms:

- FLOAT. Use DOUBLE or REAL instead.
- NUMERIC. Use DECIMAL instead.

**Specifying the encoding scheme for parameters:** The implicitly or explicitly specified encoding scheme of all the parameters with a string data type (both input and output parameters) must be the same—either all ASCII, all EBCDIC, or all UNICODE.

**Character string representation considerations:** The PARAMETER VARCHAR clause is specific to LANGUAGE C functions because of the native use of NUL-terminated strings in C. VARCHAR structure representation is useful when character string data is known to contain embedded NUL-terminators. It is also useful when it cannot be guaranteed that character string data does not contain embedded NUL-terminators.

PARAMETER VARCHAR does not apply to fixed length character strings, VARCHAR FOR BIT DATA, CLOB, DBCLOB, or implicitly generated parameters. The clause does not apply to VARCHAR FOR BIT DATA because BIT DATA can contain X'00' characters, and its value representation starts with length information. It does not apply to LOB data because a LOB value representation starts with length information.

PARAMETER VARCHAR does not apply to optional parameters that are implicitly provided to an external procedure. For example, a CREATE PROCEDURE statement for LANGUAGE C must also specify PARAMETER STYLE SQL, which returns an SQLSTATE NUL-terminated character string; that SQLSTATE will not be represented in VARCHAR structured form. Likewise, none of the parameters that represent the qualified name of the procedure, the specific name of the procedure, or the SQL diagnostic string that is returned to the database manager will be represented in VARCHAR structured form.

## CREATE PROCEDURE (external)

**Running stored procedures:** You can use the WLM ENVIRONMENT clause to identify the MVS address space in which a stored procedure is to run. Using different WLM environments lets you isolate one group of programs from another. For example, you might choose to isolate programs based on security requirements and place all payroll applications in one WLM environment because those applications deal with sensitive data, such as employee salaries.

If you use NO WLM ENVIRONMENT, the stored procedure will run in the DB2-established stored procedure address space, where there is no ability to isolate one group of programs from another.

Regardless of where the stored procedure is to run, DB2 invokes RACF to determine whether you have appropriate authorization. You must have authorization to issue CREATE PROCEDURE statements that refer to the specified WLM environment or the DB2-established stored procedure address space. For example, the following RACF command authorizes DB2 user DB2USER1 to define stored procedures on DB2 subsystem DB2A that run in the WLM environment named PAYROLL.

```
PERMIT DB2A.WLMENV.PAYROLL CLASS(DSNR) ID(DB2USER1) ACCESS(READ)
```

Similarly, the following RACF command authorizes the same user to define stored procedures that run in the DB2 stored procedure address space named DB2ASPAS.

```
PERMIT DB2A.WLMENV.DB2ASPAS CLASS(DSNR) ID(DB2USER1) ACCESS(READ)
```

**Accessing result sets from nested stored procedures:** When another stored procedure or a user-defined function calls a stored procedure, only the calling routine can access the result sets that the stored procedure returns. The result sets are not returned to the application that contains the outermost stored procedure or user-defined function in the sequence of nested calls.

When a stored procedure is nested, the result sets that are returned by the stored procedure are accessible only by the calling routine. The result sets are not returned to the application that contains the outermost stored procedure or user-defined function in the sequence of nested calls.

**Restrictions for nested stored procedures:** A stored procedure, user-defined function, or trigger cannot call a stored procedure that is defined with the COMMIT ON RETURN clause.

## Examples

*Example 1:* Create the definition for a stored procedure that is written in COBOL. The procedure accepts an assembly part number and returns the number of parts that make up the assembly, the total part cost, and a result set. The result set lists the part numbers, quantity, and unit cost of each part. Assume that the input parameter cannot contain a null value and that the procedure is to run in a WLM environment called PARTSA.

```
CREATE PROCEDURE SYSPROC.MYPROC(IN INT, OUT INT, OUT DECIMAL(7,2))
LANGUAGE COBOL
EXTERNAL NAME MYMODULE
PARAMETER STYLE GENERAL
WLM ENVIRONMENT PARTSA
DYNAMIC RESULT SETS 1;
```

*Example 2:* Create the definition for the stored procedure described in Example 1, except use the linkage convention that passes more information than the parameter specified on the CALL statement. Specify Language Environment run-time options HEAP, BELOW, ALL31, and STACK.

```
CREATE PROCEDURE SYSPROC.MYPROC(IN INT, OUT INT, OUT DECIMAL(7,2))
LANGUAGE COBOL
EXTERNAL NAME MYMODULE
PARAMETER STYLE DB2SQL
WLM ENVIRONMENT PARTSA
DYNAMIC RESULT SETS 1
RUN OPTIONS 'HEAP(,,ANY),BELOW(4K,,),ALL31(ON),STACK(,,ANY,)';
```

*Example 3:* Create the procedure definition for a stored procedure, written in Java, that is passed a part number and returns the cost of the part and the quantity that is currently available.

```
CREATE PROCEDURE PARTS_ON_HAND(IN PARTNUM INT, OUT COST DECIMAL(7,2),
 OUT QUANTITY INT)
LANGUAGE JAVA
EXTERNAL NAME 'PARTS.ONHAND'
PARAMETER STYLE JAVA;
```

# CREATE PROCEDURE (SQL)

The CREATE PROCEDURE statement registers an SQL procedure with a database server and specifies the source statements for an SQL procedure. See Chapter 6, “SQL procedure statements,” on page 943.

## Invocation

This statement can only be dynamically prepared, but the DYNAMICRULES run behavior must be specified implicitly or explicitly. It is intended to be processed using one of the following methods:

- Using IBM DB2 Stored Procedure Builder
- Using JCL
- Using the DB2 for OS/390 SQL procedure processor (DSNTPSMP)

For more information on preparing SQL procedures for execution, see Part 6 of *DB2 Application Programming and SQL Guide*. Issuing the CREATE PROCEDURE statement from another context will result in an incomplete procedure definition even though the statement processing returns without error.

## Authorization

The privilege set that is defined below must include at least one of the following:

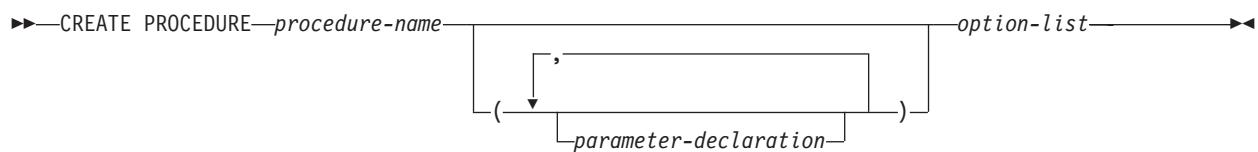
- The CREATEIN privilege for the schema or all schemas
- SYSADM or SYSCTRL authority

The authorization ID that matches the schema name implicitly has the CREATEIN privilege on the schema.

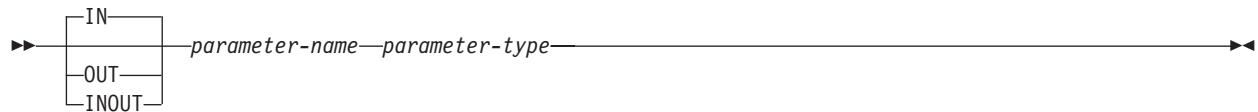
**Privilege set:** The privilege set is the privileges that are held by the authorization ID of the owner of the plan or package.

The authorization ID that is used to create the stored procedure must have authority to create programs that are to be run either in the DB2-established stored procedure address space or the specified workload manager (WLM) environment.

## Syntax



### parameter-declaration:

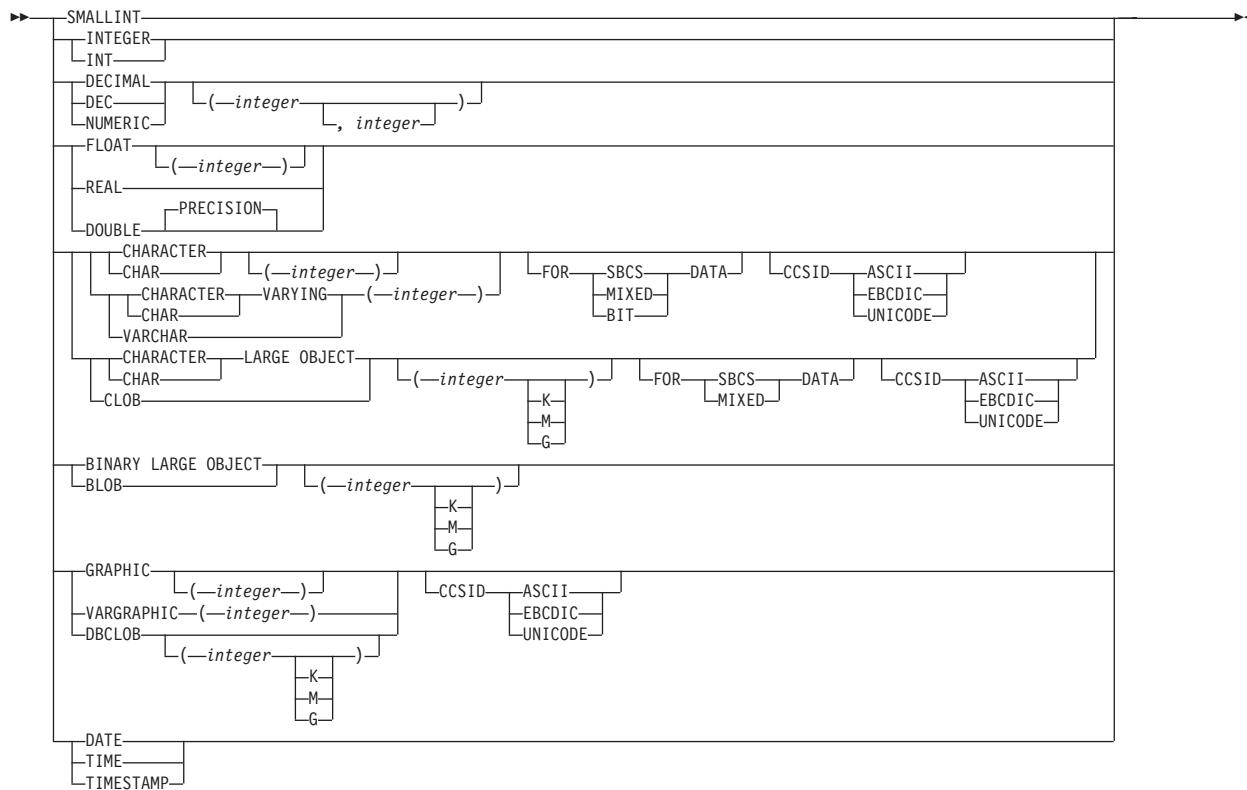


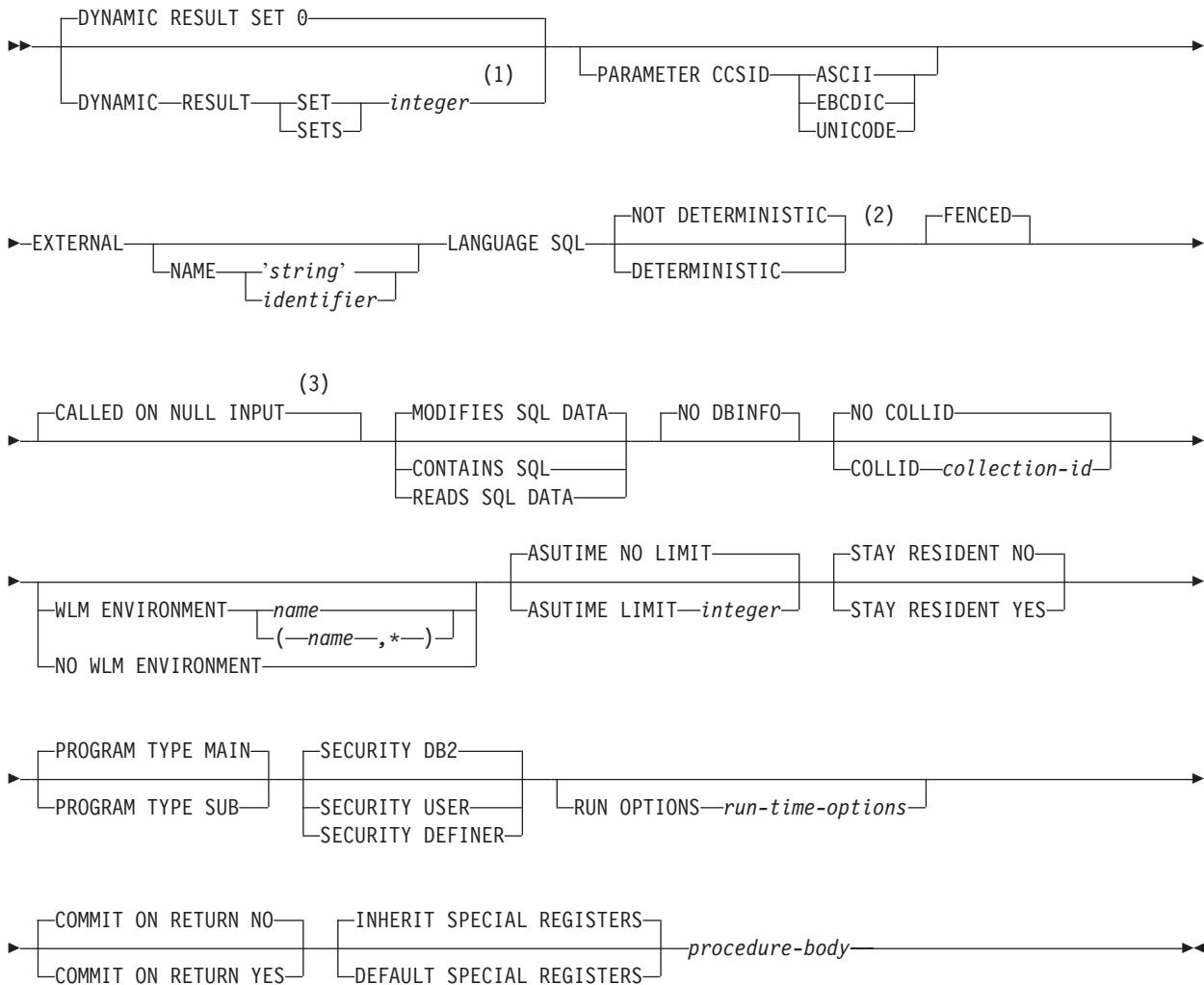
### parameter-type:



## CREATE PROCEDURE (SQL)

### built-in-data-type:



**option-list:****Notes:**

- 1 Synonyms include RESULT SET for DYNAMIC RESULT SET and RESULT SETS for DYNAMIC RESULT SETS.
- 2 Synonyms include VARIANT for NOT DETERMINISTIC, and NOT VARIANT for DETERMINISTIC.
- 3 NULL CALL is a synonym for CALLED ON NULL INPUT.

**Description***procedure-name*

Names the stored procedure. Although the name of an SQL procedure can be a delimited identifier, the name itself can contain only uppercase characters A through Z and digits 0 through 9 and must begin with an alphabetic character.

## CREATE PROCEDURE (SQL)

The name of an SQL procedure cannot contain the alphabetic extenders for national languages (#, @, \$) or underscore (\_).

The name is implicitly or explicitly qualified by a schema. The name, including the implicit or explicit qualifier, must not identify an existing stored procedure at the current server.

- The unqualified form of *procedure-name* is a long SQL identifier. The unqualified name is implicitly qualified with a schema name according to the following rules:

If the statement is embedded in a program, the schema name is the authorization ID in the QUALIFIER bind option when the plan or package was created or last rebound. If QUALIFIER was not specified, the schema name is the owner of the plan or package.

If the statement is dynamically prepared, the schema name is the SQL authorization ID in the CURRENT SQLID special register.

- The qualified form of *procedure-name* is a short SQL identifier (the schema name) followed by a period and a long SQL identifier.

The schema name must not begin with 'SYS' unless the schema name is 'SYSPROC' or 'SYSADM'. The schema name can be SYSIBM if the privilege set includes SYSADM or SYSCTRL authority.

The owner of the procedure is determined by how the CREATE PROCEDURE statement is invoked:

- If the statement is embedded in a program, the owner is the authorization ID of the owner of the plan or package.
- If the statement is dynamically prepared, the owner is the SQL authorization ID in the CURRENT SQLID special register.

The owner is implicitly given the EXECUTE privilege with the GRANT option for the procedure.

If the first 8 bytes of the procedure name contain an underscore and the procedure definition is not being processed by the Stored Procedure Builder or DSNTPSMP (the OS/390 SQL procedure processor), then specify the EXTERNAL NAME clause.

(*parameter-declaration*,...)

Specifies the number of parameters of the stored procedure, data type of each parameter, and the name of each parameter. All of the parameters are nullable. A parameter for a stored procedure can be used only for input, only for output, or for both input and output. If an error is returned by the procedure, OUT parameters are undefined, and INOUT parameters are unchanged.

**IN** Identifies the parameter as an input parameter to the stored procedure. The value of the parameter on entry to the procedure is the value that is returned to the calling SQL application.

IN is the default.

**OUT**

Identifies the parameter as an output parameter that is returned by the stored procedure.

**INOUT**

Identifies the parameter as both an input and output parameter for the stored procedure.

*parameter-name*

Names the parameter. *parameter-name* is a long identifier. A parameter name cannot be an SQL reserved word. For a list of SQL reserved words, see Appendix F, “SQL reserved words,” on page 1153.

*parameter-type*

Specifies the data type of the parameter.

*built-in-data-type*

The data type of the parameter is a built-in data type. You can use the same built-in data types as for the CREATE TABLE statement except LONG VARCHAR or LONG VARGRAPHIC. Use VARCHAR or VARGRAPHIC with an explicit length instead.

For more information on the data types, including the subtype of character data types (the FOR *subtype* DATA clause), see “built-in-data-type” on page 658.

If you do not specify a specific value for the data types that have length, precision, or scale attributes (CHAR, GRAPHIC, DECIMAL, NUMERIC, FLOAT), the defaults are as follows:

|                |                      |
|----------------|----------------------|
| <b>CHAR</b>    | CHAR(1)              |
| <b>GRAPHIC</b> | GRAPHIC(1)           |
| <b>DECIMAL</b> | DECIMAL(5,0)         |
| <b>FLOAT</b>   | DOUBLE (length of 8) |

For parameters with a string data type, the CCSID clause indicates whether the encoding scheme of the parameter value is ASCII, EBCDIC, or UNICODE. If you do not specify CCSID ASCII, CCSID EBCDIC, or CCSID UNICODE, the encoding scheme is the value of field DEF ENCODING SCHEME on installation panel DSNTIPF.

**TABLE LIKE *table-name* AS LOCATOR**

Specifies that the parameter is a transition table. However, when the procedure is called, the actual values in the transition table are not passed to the stored procedure. A single value is passed instead. This single value is a locator to the table, which the procedure uses to access the columns of the transition table. A procedure with a table parameter can only be invoked from the triggered action of a trigger.

For more information about the TABLE LIKE clause, see “TABLE LIKE” on page 624. For more information about using table locators, see *DB2 Application Programming and SQL Guide*.

Although an input parameter with a character data type has an implicitly or explicitly specified subtype (BIT, SBCS, or MIXED), the value that is actually passed in the input parameter can have any subtype. Therefore, conversion of the input data to the subtype of the parameter might occur when the procedure is called. With ASCII or EBCDIC, an error occurs if mixed data that actually contains DBCS characters is used as the value for an input parameter that is declared with an SBCS subtype.

A parameter with a datetime data type is passed to the SQL procedure as a different data type. A datetime type parameter is passed as a character data type, and the data is passed in ISO format.

The encoding scheme for a datetime type parameter is determined as follows:

## CREATE PROCEDURE (SQL)

- If there are one or more parameters with a character or graphic data type, the encoding scheme of the datetime type parameter is the same as the encoding scheme of the character or graphic parameters.
- Otherwise, the encoding scheme is the value of field DEF ENCODING SCHEME on installation panel DSNTIPF.

### DYNAMIC RESULT SET *integer* or DYNAMIC RESULT SETS *integer*

Specifies the maximum number of query result sets that the stored procedure can return. The default is DYNAMIC RESULT SETS 0, which indicates that there are no result sets. The value must be between 0 and 32767.

### PARAMETER CCSID

Indicates whether the encoding scheme for string parameters is ASCII, EBCDIC, or UNICODE. The default encoding scheme is the value specified in the CCSID clauses of the parameter list or in the field DEF ENCODING SCHEME on installation panel DSNTIPF.

This clause provides a convenient way to specify the encoding scheme for *all* string parameters. If individual CCSID clauses are specified for individual parameters in addition to this PARAMETER CCSID clause, the value specified in *all* of the CCSID clauses must be the same value that is specified in this clause.

This clause also specifies the encoding scheme to be used for system-generated parameters of the routine such as message tokens and DBINFO.

### EXTERNAL

Specifies the program that runs when the procedure name is specified in a CALL statement.

The program does not need to exist when the CREATE PROCEDURE statement is executed. However, it must exist and be accessible by the current server when a CALL statement to the stored procedure is issued.

You can specify the EXTERNAL clause in one of the following ways:

EXTERNAL

EXTERNAL NAME PKJVSP1

EXTERNAL NAME 'PKJVSP1'

If you specify an external program name, you must use the NAME keyword. For example, this syntax is not valid:

EXTERNAL PKJVSP1

### NAME '*string*' or *identifier*

Identifies the name of the MVS load module that contains the generated code that implements the logic of the procedure. The content of the value (whether specified as a string constant or an identifier) must contain only the uppercase alphabetic characters A through Z and the characters 0 through 9. The name must begin with an alphabetic character. The name specified must not be the same as the name used by another SQL procedure.

If the EXTERNAL NAME is not specified and the Stored Procedure Builder of DSNTPSMP (the DB2 for OS/390 SQL procedure processor) is used to process the procedure definition, then the load module name is generated. Otherwise, the procedure name (up to 8 bytes) is used as the load module

name. In this case, the first 8 bytes of the name should not contain an underscore because the underscore is not a valid character for the name of an MVS load module.

The name specified in the EXTERNAL NAME clause, generated by DB2, or defaulted to from the procedure name is used to identify the names of the files for the load module, DBRM, and C source code. DB2 does not check for the existence of PDS members when writing out the load module, DBRM, and C source code. Therefore, existing files may be overwritten.

#### **LANGUAGE**

Specifies the application programming language in which the stored procedure is written.

#### **SQL**

The stored procedure is written in DB2 SQL procedure language.

#### **NOT DETERMINISTIC or DETERMINISTIC**

Specifies whether the stored procedure returns the same result from successive calls with identical input arguments.

##### **NOT DETERMINISTIC**

The stored procedure might not return the same result from successive calls with identical input arguments. NOT DETERMINISTIC is the default.

##### **DETERMINISTIC**

The stored procedure returns the same result from successive calls with identical input arguments.

DB2 does not verify that the stored procedure code is consistent with the specification of DETERMINISTIC or NOT DETERMINISTIC.

#### **FENCED**

Specifies that the stored procedure runs in an external address space to prevent user programs from corrupting DB2 storage.

FENCED is the default.

#### **CALLED ON NULL INPUT**

Specifies that the stored procedure will be called even if any of the input arguments is null, making the procedure responsible for testing for null argument values. The result is the null value. CALLED ON NULL INPUT is the default.

#### **MODIFIES SQL DATA, READS SQL DATA, or CONTAINS SQL DATA**

Indicates whether the stored procedure can execute any SQL statements and, if so, what type. See Table 72 on page 972 for a detailed list of the SQL statements that can be executed under each data access indication.

##### **MODIFIES SQL DATA**

The stored procedure can execute any SQL statement except those statements that are not supported in any stored procedure.

MODIFIES SQL DATA is the default.

##### **READS SQL DATA**

The stored procedure cannot execute SQL statements that modify data. SQL statements that are not supported in any stored procedure return a different error.

## CREATE PROCEDURE (SQL)

### CONTAINS SQL

The stored procedure cannot execute any SQL statements that read or modify data. SQL statements that are not supported in any stored procedure return a different error.

### NO DBINFO

Specifies whether specific information known by DB2 is passed to the stored procedure when it is invoked.

### NO DBINFO

Additional information is not passed. Only NO DBINFO is allowed for SQL procedures.

### NO COLLID or COLLID *collection-id*

Identifies the package collection that is to be used when the stored procedure is executed. This is the package collection into which the DBRM that is associated with the stored procedure is bound.

### NO COLLID

The package collection for the stored procedure is the same as the package collection of the calling program. If the calling program does not use a package, the package collection is set to the value of special register CURRENT PACKAGESET.

NO COLLID is the default.

### COLLID *collection-id*

The package collection for the stored procedure is the one specified.

### WLM ENVIRONMENT

Identifies the MVS workload manager (WLM) environment in which the stored procedure is to run when the DB2 stored procedure address space is WLM-established. The *name* of the WLM environment is a long identifier.

If you do not specify WLM ENVIRONMENT, the stored procedure runs in the default WLM-established stored procedure address space specified at installation time.

#### *name*

The WLM environment in which the stored procedure must run. If another stored procedure or a user-defined function calls the stored procedure and that calling routine is running in an address space that is not associated with the specified WLM environment, DB2 routes the stored procedure request to a different MVS address space.

#### (*name*, \*)

When an SQL application program directly calls a stored procedure, the WLM environment in which the stored procedure runs.

If another stored procedure or a user-defined function calls the stored procedure, the stored procedure runs in the same WLM environment that the calling routine uses.

To define a stored procedure that is to run in a specified WLM environment, you must have appropriate authority for the WLM environment. For an example of a RACF command that provides this authorization, see “Running stored procedures” on page 634.

### NO WLM ENVIRONMENT

Indicates that the stored procedure is to run in the DB2-established stored procedure address space.

| Do not specify NO WLM ENVIRONMENT if you implicitly or explicitly define the stored procedure with SECURITY USER, SECURITY DEFINER, or PROGRAM TYPE SUB or if there are any LOB parameters.

To define a stored procedure that is to run in the DB2-established stored procedure address space, you must have appropriate authority for the address space. For an example of a RACF command that provides this authorization, see “Running stored procedures” on page 634.

#### **ASUTIME**

Specifies the total amount of processor time, in CPU service units, that a single invocation of a stored procedure can run. The value is unrelated to the ASUTIME column of the resource limit specification table.

When you are debugging a stored procedure, setting a limit can be helpful in case the stored procedure gets caught in a loop. For information on service units, see *OS/390 MVS Initialization and Tuning Guide*.

#### **NO LIMIT**

There is no limit on the service units. NO LIMIT is the default.

#### **LIMIT *integer***

The limit on the service units is a positive *integer* in the range of 1 to 2G. If the stored procedure uses more service units than the specified value, DB2 cancels the stored procedure.

#### **STAY RESIDENT**

Specifies whether the stored procedure load module remains resident in memory when the stored procedure ends.

#### **NO**

The load module is deleted from memory after the stored procedure ends. NO is the default.

#### **YES**

The load module remains resident in memory after the stored procedure ends.

#### **PROGRAM TYPE**

Specifies whether the stored procedure runs as a main routine or a subroutine.

#### **MAIN**

The stored procedure runs as a main routine. MAIN is the default for SQL procedures.

#### **SUB**

The stored procedure runs as a subroutine.

#### **SECURITY**

Specifies how the stored procedure interacts with an external security product, such as RACF, to control access to non-SQL resources.

#### **DB2**

The stored procedure does not require a special external security environment. If the stored procedure accesses resources that an external security product protects, the access is performed using the authorization ID associated with the stored procedure address space. DB2 is the default.

#### **USER**

An external security environment should be established for the stored procedure. If the stored procedure accesses resources that the external security product protects, the access is performed using the authorization ID of the user who invoked the stored procedure.

## CREATE PROCEDURE (SQL)

### DEFINER

An external security environment should be established for the stored procedure. If the stored procedure accesses resources that the external security product protects, the access is performed using the authorization ID of the owner of the stored procedure.

### RUN OPTIONS *run-time-options*

Specifies the Language Environment run-time options to be used for the stored procedure. You must specify *run-time-options* as a character string that is no longer than 254 bytes. If you do not specify RUN OPTIONS or pass an empty string, DB2 does not pass any run-time options to Language Environment, and Language Environment uses its installation defaults.

For a description of the Language Environment run-time options, see *OS/390 Language Environment for OS/390 & VM Programming Reference*.

### COMMIT ON RETURN

Indicates whether DB2 commits the transaction immediately on return from the stored procedure.

#### NO

DB2 does not issue a commit when the stored procedure returns. NO is the default.

#### YES

DB2 issues a commit when the stored procedure returns if the following statements are true:

- The SQLCODE that is returned by the CALL statement is not negative.
- The stored procedure is not in a must abort state.

The commit operation includes the work that is performed by the calling application process and the stored procedure.

If the stored procedure returns result sets, the cursors that are associated with the result sets must have been defined as WITH HOLD to be usable after the commit.

### INHERIT SPECIAL REGISTERS

Indicates that the values of special registers are inherited, according to the rules listed in the table for characteristics of special registers in a stored procedure function in Table 19 on page 93.

### DEFAULT SPECIAL REGISTERS

Indicates that special registers are initialized to the default values, as indicated by the rules in the table for characteristics of special registers in a stored procedure function in Table 19 on page 93.

### *procedure-body*

Specifies the source code for an SQL procedure. See Chapter 6, “SQL procedure statements,” on page 943 for information on how to write a procedure body.

## Notes

The following rules apply to the use of parameters in SQL procedures:

- If IN is specified for a parameter in an SQL procedure, the parameter can be modified within the SQL procedure body. When control returns to the caller, the original value of the IN parameter on entry to the procedure is returned to the caller.
- If OUT is specified for a parameter in an SQL procedure, the parameter can be used on the left or right side of an assignment statement in the SQL procedure

body. The parameter can be checked or used to set other variables. The last value that is assigned to an OUT parameter is returned to the caller. If the parameter is not set, DB2 returns the null value to the caller.

- If INOUT is specified for a parameter in an SQL procedure, the parameter can be used on the left or right side of an assignment statement in the SQL procedure body. The first value of the parameter is determined by the caller, and the last value that is assigned to the parameter is returned to the caller.

See “Notes” on page 633 for information about:

- Choosing data types for parameters
- Specifying the encoding scheme for parameters
- Environments for running stored procedures
- Accessing result sets from nested stored procedures

## Examples

*Example 1:* Create the definition for an SQL procedure. The procedure accepts an employee number and a multiplier for a pay raise as input. The following tasks are performed in the procedure body:

- Calculate the employee's new salary.
- Update the employee table with the new salary value.

```
CREATE PROCEDURE UPDATE_SALARY_1
 (IN EMPLOYEE_NUMBER CHAR(10),
 IN RATE DECIMAL(6,2))
LANGUAGE SQL
MODIFIES SQL DATA
UPDATE EMP
SET SALARY = SALARY * RATE
WHERE EMPNO = EMPLOYEE_NUMBER
```

*Example 2:* Create the definition for the SQL procedure described in example 1, but specify that the procedure has these characteristics:

- The procedure runs in a WLM environment called PARTSA.
- The same input always produces the same output.
- SQL work is committed on return to the caller.
- The Language Environment run-time options to be used when the SQL procedure executes are 'MSGFILE(OUTFILE),RPTSTG(ON),RPTOPTS(ON)'.

```
CREATE PROCEDURE UPDATE_SALARY_1
 (IN EMPLOYEE_NUMBER CHAR(10),
 IN RATE DECIMAL(6,2))
LANGUAGE SQL
MODIFIES SQL DATA
WLM ENVIRONMENT PARTSA
DETERMINISTIC
RUN OPTIONS 'MSGFILE(OUTFILE),RPTSTG(ON),RPTOPTS(ON)'
COMMIT ON RETURN YES
UPDATE EMP
SET SALARY = SALARY * RATE
WHERE EMPNO = EMPLOYEE_NUMBER
```

For more examples of SQL procedures, see Chapter 6, “SQL procedure statements,” on page 943.

## CREATE STOGROUP

### CREATE STOGROUP

The CREATE STOGROUP statement creates a storage group at the current server. Storage from the identified volumes can later be allocated for table spaces and index spaces.

### Invocation

This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is implicitly or explicitly specified.

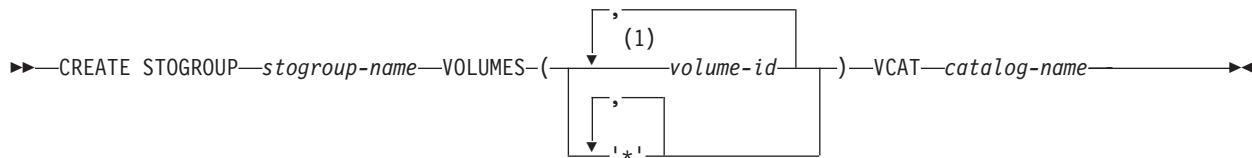
### Authorization

The privilege set that is defined below must include at least one of the following:

- The CREATESG privilege
- SYSADM or SYSCTRL authority

**Privilege set:** If the statement is embedded in an application program, the privilege set is the privileges that are held by the authorization ID of the owner of the plan or package. If the statement is dynamically prepared, the privilege set is the privileges that are held by the SQL authorization ID of the process.

### Syntax



#### Notes:

- 1 The same *volume-id* must not be specified more than once.

### Description

#### *stogroup-name*

Names the storage group. The name must not identify a storage group that exists at the current server.

#### **VOLUMES(*volume-id*,...)** or **VOLUMES('\*,...)**

Defines the volumes of the storage group. Each *volume-id* is a volume serial number of a storage volume. It can have a maximum of six characters and is specified as an identifier or a string constant.

Asterisks are recognized only by Storage Management Subsystem (SMS). To allow SMS control over volume selection, define DB2 STOGROUPs with VOLUMES('\*,...). SMS usage is recommended, rather than using DB2 to allocate data to specific volumes. Having DB2 select the volume requires non-SMS usage or assigning an SMS Storage Class with guaranteed space. However, because guaranteed space reduces the benefits of SMS allocation, it is not recommended.

If you do choose to use specific volume assignments, additional manual space management must be performed. Free space must be managed for each

| individual volume to prevent failures during the initial allocation and extension.  
| This process generally requires more time for space management and results in  
| more space shortages. Guaranteed space should be used only where the  
| space needs are relatively small and do not change.

**VCAT** *catalog-name*

Identifies the integrated catalog facility catalog for the storage group. You must specify the catalog name in the form of a short identifier. Thus, you must specify an alias if the name of the integrated catalog facility catalog is longer than 8 characters.

The designated catalog is the one in which entries are placed for the data sets created by DB2 with the aid of the storage group. These are linear VSAM data sets for associated table or index spaces or for their partitions. For each such space or partition, association is made through a USING clause in a CREATE TABLESPACE, CREATE INDEX, ALTER TABLESPACE, or ALTER INDEX statement. For more on the association, see the descriptions of those statements in this chapter.

Conventions for data set names are given in Part 2 (Volume 1) of *DB2 Administration Guide*. *catalog-name* is the first qualifier for each data set name.

One or more DB2 subsystems could share integrated catalog facility catalogs with the current server. To avoid the chance of having one of those subsystems attempt to assign the same name to different data sets, select a value for *catalog-name* that is not used by the other DB2 subsystems.

## Notes

**Device types:** When the storage group is used at run time, an error can occur if the volumes in the storage group are of different device types, or if a volume is not available to MVS for dynamic allocation of data sets.

When a storage group is used to extend a data set, all volumes in the storage group must be of the same device type as the volumes used when the data set was defined. Otherwise, an extend failure occurs if an attempt is made to extend the data set.

**Number of volumes:** There is no specific limit on the number of volumes that can be defined for a storage group. However, the maximum number of volumes that can be managed for a storage group is 133. Thus, there is no point in creating a storage group with more than 133 volumes.

MVS imposes a limit on the number of volumes that can be allocated per data set: 59 at this writing. For the latest information on that restriction, see *DFSMS/MVS: Access Method Services for the Integrated Catalog*.

**Storage group owner:** If the statement is embedded in an application program, the owner of the plan or package is the owner of the storage group. If the statement is dynamically prepared, the SQL authorization ID of the process is the owner of the storage group. The owner has the privilege of altering and dropping the storage group.

**Specifying volume IDs:** A new storage group must have either specific volume IDs or non-specific volume IDs. You cannot create a storage group that contains a mixture of specific and non-specific volume IDs.

## CREATE STOGROUP

**Verifying volume IDs:** When processing the VOLUMES clause, DB2 does not check the existence of the volumes or determine the types of devices that they identify. Later, whenever the storage group is used to allocate data sets, the list of volumes is passed in the specified order to Data Facilities (DFSMSSdfp), which does the actual work. See Part 2 (Volume 1) of *DB2 Administration Guide* for more information about creating DB2 storage groups.

### Example

Create storage group, DSN8G710, of volumes ABC005 and DEF008. DSNCAT is the integrated catalog facility catalog name.

```
CREATE STOGROUP DSN8G710
 VOLUMES (ABC005,DEF008)
 VCAT DSNCAT;
```

# CREATE SYNONYM

The CREATE SYNONYM statement defines a synonym for a table or view at the current server.

## Invocation

This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is implicitly or explicitly specified.

## Authorization

None required.

# Syntax

## Description

### *synonym*

Names the synonym. The name must not identify a synonym, table, view, or alias owned by authorization ID *x*. If the statement is embedded in an application program, *x* is the owner of the plan or package. If the statement is dynamically prepared, *x* is the value of CURRENT\_SQLID. In either case, *x* becomes the owner of the synonym.

**FOR** *authorization-name.table-name* or *authorization-name.view-name*

Identifies the object to which the synonym applies. The name must consist of two parts and must identify a table, view, or alias that exists at the current server. If a table is identified, it must not be an auxiliary table or a declared temporary table. If an alias is identified, it must be an alias for a table or view at the current server and the synonym is defined for that table or view.

## Notes

In cases where the statement is dynamically prepared, users with SYSADM authority can create synonyms for other users. This is done by changing the value of the CURRENT SQLID special register before issuing the CREATE SYNONYM statement. See “SET CURRENT SQLID” on page 916 for details on changing the value of the CURRENT SQLID special register.

The authorization ID recorded as the owner of a synonym is the only authorization ID for which the synonym is defined and the only authorization ID that can be used to drop it.

If an alias is used to denote the table or view, the name of that table or view, not the alias, is recorded in the catalog as the definition of the synonym. That severs the connection between the synonym and alias, and even if the alias is dropped and redefined, the synonym is still in effect and names the original table or view.

## Example

Define DEPT as a synonym for the table DSN8710.DEPT.

## **CREATE SYNONYM**

```
CREATE SYNONYM DEPT
FOR DSN8710.DEPT;
```

This example does not work if the current SQL authorization ID is DSN8710.

---

## CREATE TABLE

The CREATE TABLE statement defines a table at the current server. The definition must include its name and the names and attributes of its columns. The definition can include other attributes of the table, such as its primary key and its table space.

### Invocation

This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is implicitly or explicitly specified.

### Authorization

The privilege set that is defined below must include at least one of the following:

- The CREATETAB privilege for the database implicitly or explicitly specified by the IN clause
- DBADM, DBCTRL, or DBMAINT authority for the database
- SYSADM or SYSCTRL authority

Additional privileges might be required when:

- The clause IN, LIKE or FOREIGN KEY is specified.
- The data type of a column is a distinct type.
- The table space is implicitly created.

See the description of the appropriate clauses for details about these privileges.

**Privilege set:** If the statement is embedded in an application program, the privilege set is the privileges that are held by the authorization ID of the owner of the plan or package. If the specified table name includes a qualifier that is not the same as this authorization ID, the privilege set must include SYSADM or SYSCTRL authority, DBADM authority for the database, or DBCTRL authority for the database.

If the statement is dynamically prepared, the privilege set is the privileges that are held by the SQL authorization ID of the process. However, if the specified table name includes a qualifier that is not the same as this authorization ID, the following rules apply:

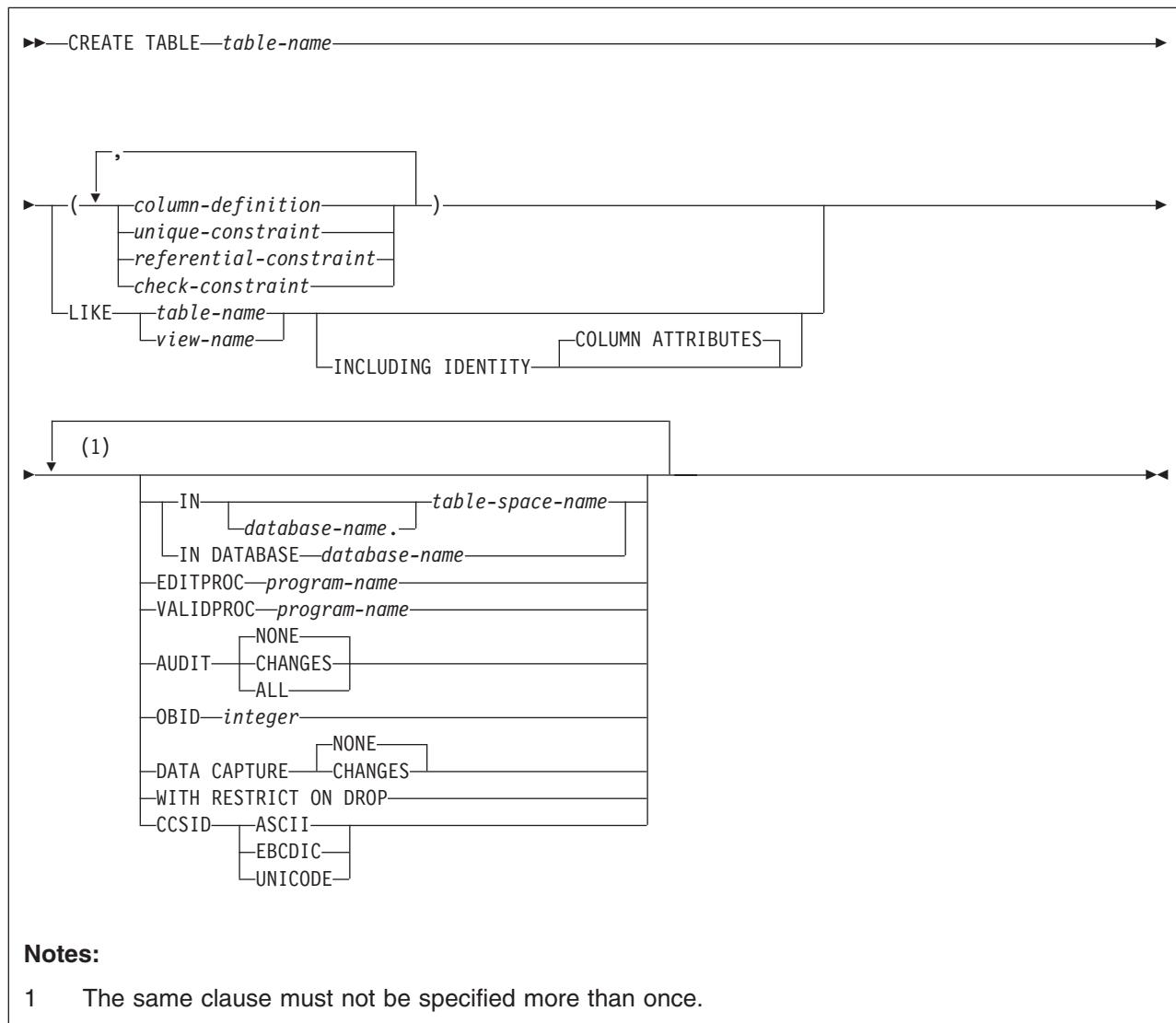
- If the privilege set includes SYSADM or SYSCTRL authority, DBADM authority for the database, or DBCTRL authority for the database, any qualifier is valid.
- If the privilege set does not include any of the authorities listed in item 1 above, the qualifier is valid only if it is the same as one of the authorization IDs of the process and the privilege set that are held by that authorization ID includes all<sup>33</sup> privileges needed to create the table.

---

33. Exception: The CREATETAB privilege is checked on the SQL authorization ID of the process.

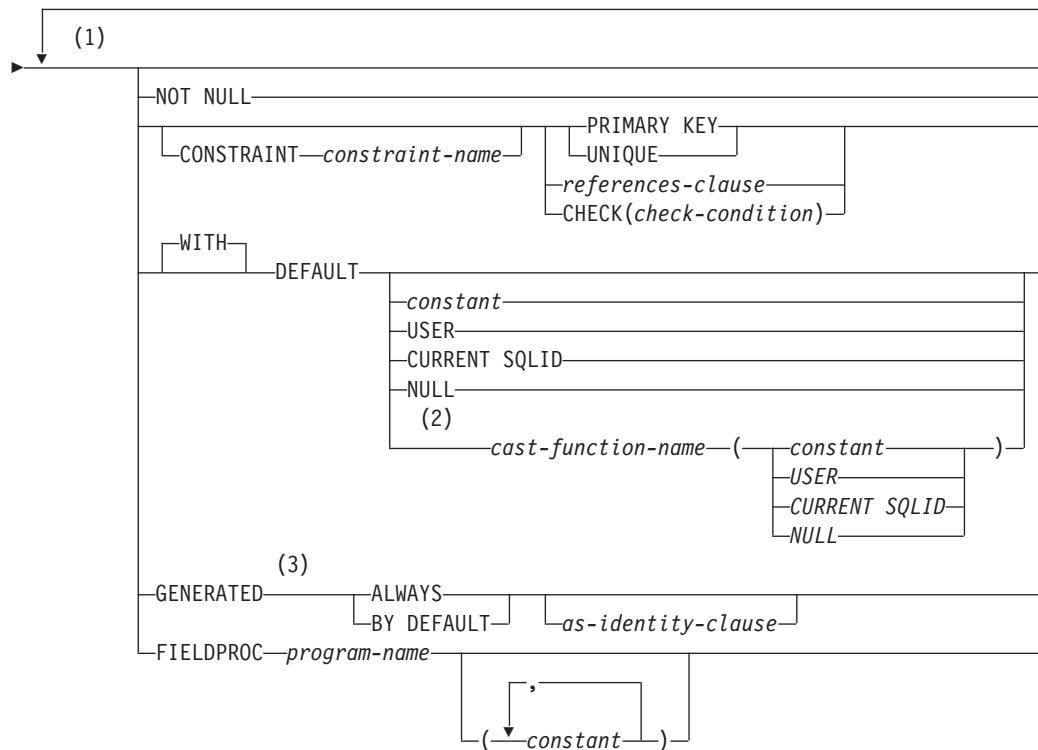
## CREATE TABLE

## Syntax



**column-definition:**

►►—*column-name*—*data-type*—►►

**Notes:**

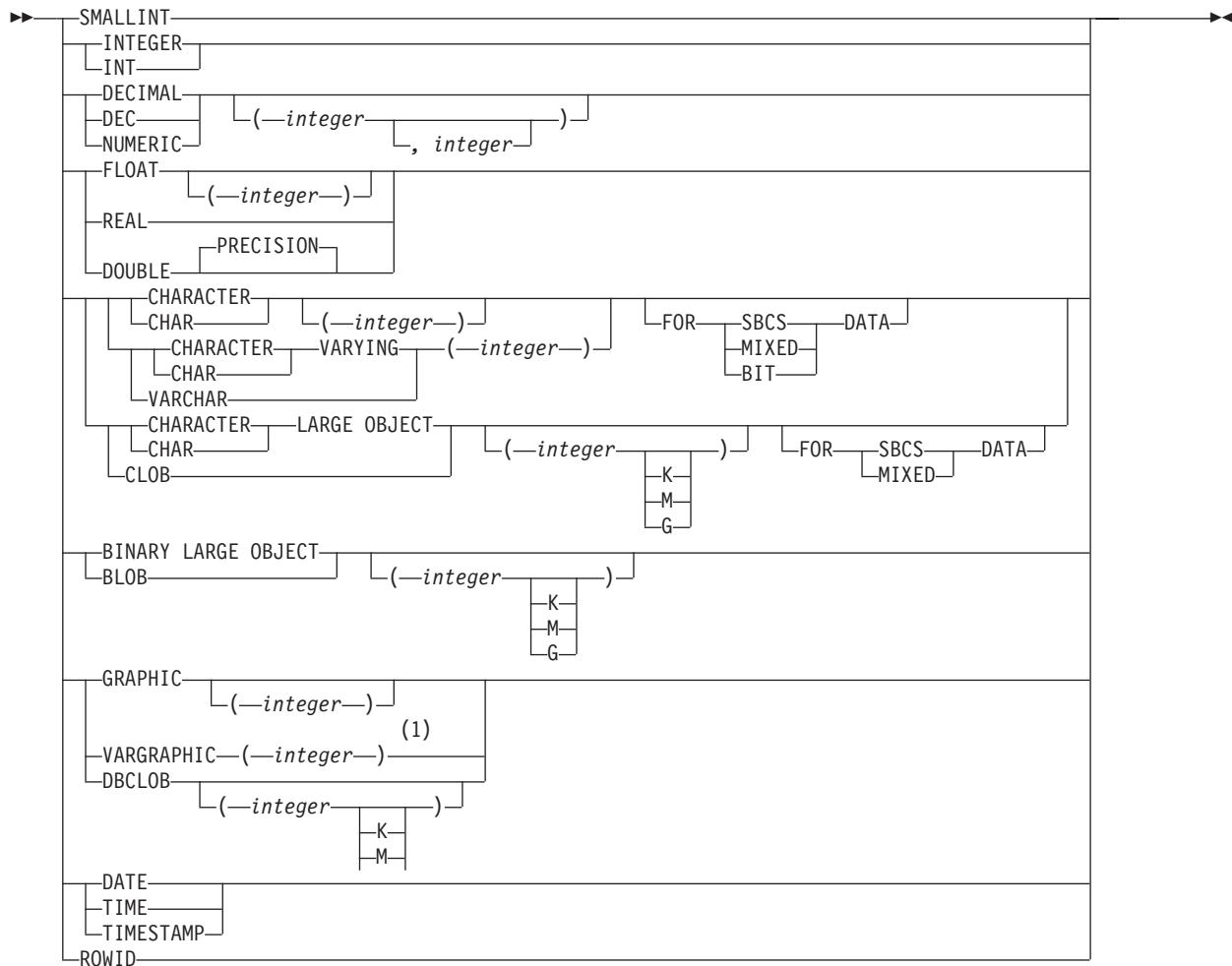
- 1 The same clause must not be specified more than once.
- 2 This form of the DEFAULT value can only be used with columns that are defined as a distinct type.
- 3 GENERATED can be specified only if the column has a ROWID data type (or a distinct type that is based on a ROWID data type), or the column is to be an identity column.

**data-type:**

►►—*built-in-data-type*  
—*distinct-type-name*—►►

## CREATE TABLE

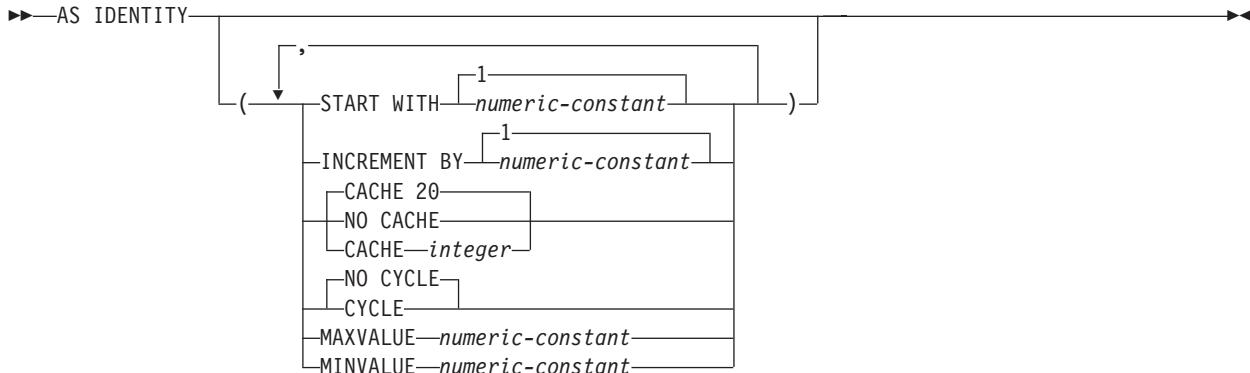
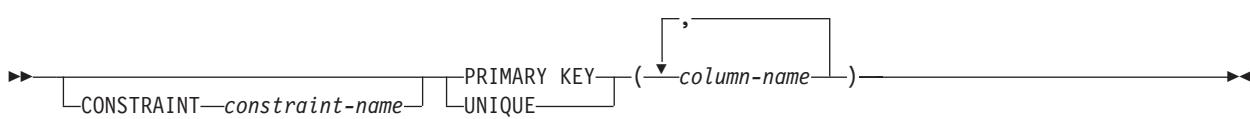
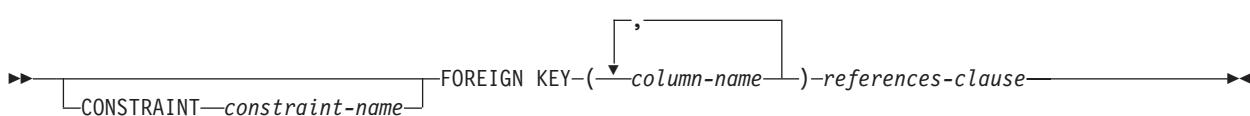
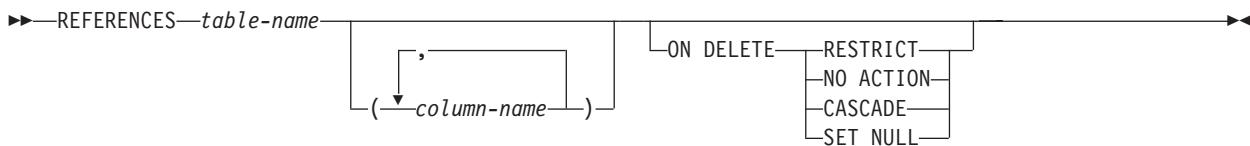
### built-in-data-type:



### Notes:

- 1 Although the syntax of LONG VARCHAR and LONG VARGRAPHIC is supported, the alternative syntax of VARCHAR(*integer*) and VARGRAPHIC(*integer*), is preferred. VARCHAR(*integer*) and VARGRAPHIC(*integer*) are recommended because after the CREATE TABLE statement is processed, DB2 considers a LONG VARCHAR column to be VARCHAR and a LONG VARGRAPHIC column to be VARGRAPHIC.

To determine the maximum length of a column defined as LONG VARCHAR or LONG VARGRAPHIC, see “Length of a LONG column” on page 677.

**as-identity-clause:****unique-constraint:****referential-constraint:****references-clause:**

## CREATE TABLE

### check-constraint:

```
► [CONSTRAINT constraint-name] CHECK (check-condition) ►
```

## Description

### *table-name*

Names the table. The name must not identify a table, view, alias, or synonym that exists at the current server.

If qualified, the name can be a two-part or three-part name. If a three-part name is used, the first part must match the value of field DB2 LOCATION NAME on installation panel DSNTIPR at the current server. (If the current server is not the local DB2, this name is not necessarily the name in the CURRENT SERVER special register.) Whether the name is two-part or three-part, the authorization ID that qualifies the name is the table's owner.

If the table name is unqualified and the statement is embedded in a program, the owner of the table is the authorization ID that serves as the implicit qualifier for unqualified object names. This is the authorization ID in the QUALIFIER operand when the plan or package was created or last rebound. If QUALIFIER was not used, the owner of the table is the owner of the package or plan.

If the table name is unqualified and the statement is dynamically prepared, the SQL authorization ID is the owner of the table.

The owner has all table privileges on the table (SELECT, UPDATE, and so on), and the authority to drop the table. All the owner's table privileges are grantable.

### column-definition

Defines the attributes of a column.

#### *column-name*

Names a column of the table. Do not qualify *column-name* and do not use the same name for more than one column of the table. For a dependent table, up to 749 columns can be named. For a table that is not a dependent, this number is 750.

#### *built-in-data-type*

Specifies the data type of the column as one of the following built-in data types, and for character string data types, specifies the subtype. If you define the table with a LOB column (CLOB, BLOB, or DBCLOB), you must also define a ROWID column. For more information, see “Creating a table with LOB columns” on page 675.

#### **INTEGER or INT**

For a large integer.

34. Columns with distinct types based on LOB or row ID types count as LOB or ROWID columns.

**SMALLINT**

For a small integer.

**FLOAT(*integer*)**

For a floating-point number. If *integer* is between 1 and 21 inclusive, the format is single precision floating-point. If the integer is between 22 and 53 inclusive, the format is double precision floating-point.

You can also specify:

|                         |                                     |
|-------------------------|-------------------------------------|
| <b>REAL</b>             | For single precision floating-point |
| <b>DOUBLE</b>           | For double precision floating-point |
| <b>DOUBLE PRECISION</b> | For double precision floating-point |
| <b>FLOAT</b>            | For double precision floating-point |

**DECIMAL(*integer,integer*) or DEC(*integer,integer*)**

For a decimal number. The first integer is the precision of the number. That is, the total number of digits, which can range from 1 to 31. The second integer is the scale of the number. That is, the number of digits to the right of the decimal point, which can range from 0 to the precision of the number. You can also specify:

|                                |                                 |
|--------------------------------|---------------------------------|
| <b>DECIMAL(<i>integer</i>)</b> | For DECIMAL( <i>integer,0</i> ) |
| <b>DECIMAL</b>                 | For DECIMAL(5,0)                |

The word NUMERIC can be used in place of DECIMAL. For example, NUMERIC(8) is equivalent to DECIMAL(8). Unlike DECIMAL, NUMERIC has no allowable abbreviation.

**CHARACTER(*integer*) or CHAR(*integer*)**

For a fixed-length character string of length *integer*, which can range from 1 to 255. If the length specification is omitted, a length of 1 character is assumed.

**VARCHAR(*integer*), CHAR VARYING(*integer*), or CHARACTER VARYING(*integer*)**

For a varying-length character string of maximum length *integer*, which can range from 1 to the maximum record size minus 8 bytes. See Table 47 on page 676 to determine the maximum record size. An integer greater than 255 defines a long string column.

**FOR *subtype* DATA**

Specifies a subtype for a character string column, which is a column with a data type of CHAR, VARCHAR, LONG VARCHAR, or CLOB. Do not use the FOR DATA clause with columns of any other data type (including any distinct type). *subtype* can be one of the following:

**SBCS**

Column holds single-byte data.

**MIXED**

Column holds mixed data. Do not specify MIXED if the value of field MIXED DATA on installation panel DSNTIPF is NO unless the CCSID UNICODE clause is also specified, or the table is being created in a Unicode table space or database.

**BIT**

Column holds BIT data. Do not specify BIT for a CLOB column.

If you do not specify the FOR clause, the column is defined with a default subtype. For ASCII or EBCDIC data:

- The default is SBCS when the value of field MIXED DATA on installation panel DSNTIPF is NO.

## CREATE TABLE

- The default is MIXED when the value is YES.

For Unicode data, the default subtype is mixed.

### **CLOB(*integer* [KIMIG]), CHAR LARGE OBJECT(*integer* [KIMIG]), or CHARACTER LARGE OBJECT(*integer* [KIMIG])**

For a character large object (CLOB) string of maximum length *integer*, which can range from 1 to 2 147 483 647. A CLOB column has a varying-length and is a long string column regardless of its length.

If the length specification is omitted, a length of 1M bytes is assumed.

The maximum value that can be specified for *integer* depends on whether a units indicator is also specified as shown in the following list.

|                  |                                                                                                             |
|------------------|-------------------------------------------------------------------------------------------------------------|
| <i>integer</i>   | The maximum value for <i>integer</i> is 2 147 483 647. The maximum length of the string is <i>integer</i> . |
| <i>integer</i> K | The maximum value for <i>integer</i> is 2 097 152. The maximum length is 1024 times <i>integer</i> .        |
| <i>integer</i> M | The maximum value for <i>integer</i> is 2048. The maximum length is 1 048 576 times <i>integer</i> .        |
| <i>integer</i> G | The maximum value for <i>integer</i> is 2. The maximum length is 1 073 741 824 times <i>integer</i> .       |

If you specify a value that evaluates to 2 gigabytes (2 147 483 648), DB2 uses a value that is one byte less, or 2 147 483 647.

### **BLOB (*integer* [KIMIG]), BINARY LARGE OBJECT(*integer* [KIMIG])**

For a binary large object (BLOB) string of maximum length *integer*, which can range from 1 to 2 147 483 647. A BLOB column has a varying-length and is a long string column regardless of its length.

If the length specification is omitted, a length of 1M bytes is assumed.

The meaning of *integer* KIMIG is the same as for CLOB.

### **GRAPHIC(*integer*)**

For a fixed-length graphic string of length *integer*, which can range from 1 to 127. If the length specification is omitted, a length of 1 character is assumed.

### **VARGRAPHIC(*integer*)**

For a varying-length graphic string of maximum length *integer*, which must range from 1 to  $n/2$ , where  $n$  is the maximum row size minus 2 bytes. An integer longer than 127 defines a long string column.

### **DBCLOB(*integer* [KIMIG])**

For a double-byte character large object (DBCLOB) string of maximum length *integer*. A DBCLOB column has a varying-length and is a long string column regardless of length.

The meaning of *integer* KIMIG is similar to CLOB. The difference is that the number specified is the number of double-byte characters and the maximum length is 1 073 741 823.

If the length specification is omitted, a length of 1M characters is assumed.

### **DATE**

For a date.

**TIME**

For a time.

**TIMESTAMP**

For a timestamp.

**ROWID**

For a row ID type.

A table can have only one ROWID column. The values in a ROWID column are unique for every row in the table and cannot be updated. You must specify NOT NULL with ROWID.

*distinct-type-name*

Specifies the data type of the column is a distinct type (a user-defined data type). The length, precision, and scale of the column are respectively the length, precision, and scale of the source type of the distinct type. The privilege set must implicitly or explicitly include the USAGE privilege on the distinct type.

The encoding scheme of the distinct type must be the same as the encoding scheme of the table. The subtype for the distinct type, if it has the attribute, is the subtype with which the distinct type was created.

If the column is to be used in the definition of the foreign key of a referential constraint, the data type of the corresponding column of the parent key must have the same distinct type.

**NOT NULL**

Prevents the column from containing null values.

**CONSTRAINT—constraint-name**

Names the constraint. If a constraint name is not specified, a unique constraint name is generated. If the name is specified, it must be different from the names of any referential, check, primary key, or unique key constraints previously specified on the table.

**PRIMARY KEY**

Provides a shorthand method of defining a primary key composed of a single column. Thus, if PRIMARY KEY is specified in the definition of column C, the effect is the same as if the PRIMARY KEY(C) clause is specified as a separate clause.

The NOT NULL clause must be specified with this clause. PRIMARY KEY cannot be specified more than once in a column definition, and must not be specified if the UNIQUE clause is specified in the definition or if the definition is for a LOB or ROWID column.

The table is marked as unavailable until its primary index is explicitly created unless the CREATE TABLE statement is processed by the schema processor. In that case, DB2 implicitly creates an index to enforce the uniqueness of the primary key and the table definition is considered complete. (For more information about implicitly created indexes, see “Implicitly created indexes” on page 677.)

**UNIQUE**

Provides a shorthand method of defining a unique key composed of a single column. Thus, if UNIQUE is specified in the definition of column C, the effect is the same as if the UNIQUE(C) clause is specified as a separate clause.

## CREATE TABLE

The NOT NULL clause must be specified with this clause. UNIQUE cannot be specified more than once in a column definition and must not be specified if the PRIMARY KEY clause is specified in the column definition or if the definition is for a LOB or ROWID column.

The table is marked as unavailable until all the required indexes are explicitly created unless the CREATE TABLE statement is processed by the schema processor. In that case, DB2 implicitly creates the indexes that are required for the unique keys and the table definition is considered complete. (For more information about implicitly created indexes, see “Implicitly created indexes” on page 677.)

### *references-clause*

The *references-clause* of a *column-definition* provides a shorthand method of defining a foreign key composed of a single column. Thus, if *references-clause* is specified in the definition of column C, the effect is the same as if that *references-clause* were specified as part of a FOREIGN KEY clause in which C is the only identified column.

Do not specify *references-clause* in the definition of a LOB or ROWID column; a LOB or ROWID column cannot be a foreign key.

### **CHECK** (*check-condition*)

CHECK (*check-condition*) provides a shorthand method of defining a check constraint that applies to a single column. For conformance with the SQL standard, if CHECK is specified in the column definition of column C, no columns other than C should be referenced in the check condition of the check constraint. The effect is the same as if the check condition were specified as a separate clause.

## DEFAULT

The default value assigned to the column in the absence of a value specified on INSERT or LOAD. Do not specify DEFAULT for a ROWID column or an identity column (a column that is defined AS IDENTITY); DB2 generates default values. If a value is not specified after DEFAULT, the default value depends on the data type of the column, as follows:

| Data Type             | Default Value                       |
|-----------------------|-------------------------------------|
| Numeric               | 0                                   |
| Fixed-length string   | Blanks                              |
| Varying-length string | A string of length 0                |
| Date                  | CURRENT DATE                        |
| Time                  | CURRENT TIME                        |
| Timestamp             | CURRENT TIMESTAMP                   |
| Distinct type         | The default of the source data type |

A default value other than the one that is listed above can be specified in one of the following forms, except for a LOB column. The only form that can be specified for a LOB column is DEFAULT NULL. Unlike other varying-length strings, a LOB column can only have the default value of a zero-length string as listed above or null.

### *constant*

Specifies a constant as the default value for the column. The value of the constant must conform to the rules for assigning that value to the column. If the table has an encoding scheme of Unicode and the column is a graphic data type (or a distinct type that is sourced on a graphic data type), a character string constant can also be specified.

**USER**

Specifies the value of the USER special register at the time of INSERT or LOAD as the default value for the column. If USER is specified, the data type of the column must be a character string with a length attribute greater than or equal to the length attribute of the USER special register, which is 8 bytes.

**CURRENT SQLID**

Specifies the value of the SQL authorization ID of the process at the time of INSERT or LOAD as the default value for the column. If CURRENT SQLID is specified, the data type of the column must be a character string with a length attribute greater than or equal to the length attribute of the CURRENT SQLID special register, which is 8 bytes.

**NULL**

Specifies null as the default value for the column.

*cast-function-name*

The name of the cast function that matches the name of the distinct type for the column. A cast function can only be specified if the data type of the column is a distinct type.

The schema name of the cast function, whether it is explicitly specified or implicitly resolved through function resolution, must be the same as the explicitly or implicitly specified schema name of the distinct type.

In a given column definition:

- DEFAULT and FIELDPROC cannot both be specified.
- NOT NULL and DEFAULT NULL cannot both be specified.
- DEFAULT cannot be specified for a ROWID column or an identity column.
- Omission of NOT NULL and DEFAULT for a column other than an identity column is an implicit specification of DEFAULT NULL. For an identity column, it is an implicit specification of NOT NULL, and DB2 generates default values.

Table 46 on page 664 summarizes the effect of specifying the various combinations of the NOT NULL and DEFAULT clauses on the CREATE TABLE statement *column-description* clause.

## CREATE TABLE

Table 46. Effect of specifying combinations of the NOT NULL and DEFAULT clauses

| If NOT NULL is:        | And DEFAULT is:              | The effect is:                                                                                      |
|------------------------|------------------------------|-----------------------------------------------------------------------------------------------------|
| Specified <sup>1</sup> | Omitted                      | An error occurs if a value is not provided for the column on INSERT or LOAD.                        |
|                        | Specified without an operand | The system defined nonnull default value is used.                                                   |
|                        | constant                     | The specified constant is used as the default value.                                                |
|                        | USER                         | The value of the USER special register at the time of INSERT or LOAD is used as the default value.  |
|                        | CURRENT SQLID                | The SQL authorization ID of the process at the time of INSERT or LOAD is used as the default value. |
|                        | NULL                         | An error occurs during the execution of CREATE TABLE.                                               |
| Omitted                | Omitted                      | Equivalent to an implicit specification of DEFAULT NULL.                                            |
|                        | Specified without an operand | The system defined nonnull default value is used.                                                   |
|                        | constant                     | The specified constant is used as the default value.                                                |
|                        | USER                         | The value of the USER special register at execution time is used as the default value.              |
|                        | CURRENT SQLID                | The SQL authorization ID of the process is used as the default value.                               |
|                        | NULL                         | Null is used as the default value.                                                                  |

**Note:** The table does not apply to a column with a ROWID data type or to an identity column.

### GENERATED

Indicates that DB2 generates values for the column. You must specify GENERATED if the column is to be considered an identity column (a column defined with the AS IDENTITY clause) or the data type of the column is a ROWID (or a distinct type that is based on a ROWID).

### ALWAYS

Indicates that DB2 will always generate a value for the column when a row is inserted into the table. ALWAYS is the recommended value unless you are using data propagation.

### BY DEFAULT

Indicates that DB2 will generate a value for the column when a row is inserted into the table unless a value is specified.

For a ROWID column, DB2 uses a specified value only if it is a valid row ID value that was previously generated by DB2 and the column has a unique, single-column index. Until this index is created on the ROWID column, the SQL INSERT statement and the LOAD utility cannot be used to add rows to the table. If the value of special register CURRENT RULES is 'STD' when the CREATE TABLE statement is processed, DB2 implicitly creates the index on the ROWID column. The name of

this index is 'I' followed by the first ten characters of the column name followed by seven randomly generated characters. If the column name is less than ten characters, DB2 adds underscore characters to the end of the name until it has ten characters. The implicitly created index has the COPY NO attribute.

For an identity column, DB2 inserts a specified value but does not verify that it is a unique value for the column unless the identity column has a unique, single-column index.

BY DEFAULT is the recommended value only when you are using data propagation.

#### **AS IDENTITY**

Specifies that the column is an identity column for the table. A table can have only one identity column. AS IDENTITY can be specified only if the data type for the column is an exact numeric type with a scale of zero (SMALLINT, INTEGER, DECIMAL with a scale of zero, or a distinct type based on one of these types).

An identity column is implicitly NOT NULL.

#### **START WITH *numeric-constant***

Specifies the first value for the identity column. The value can be any positive or negative value that could be assigned to the column without non-zero digits existing to the right of the decimal point.

If a value is not explicitly specified when the identity column is defined, the default is the MINVALUE for an ascending sequence and the MAXVALUE for a descending sequence. This value is not necessarily the value that a sequence would cycle to after reaching the maximum or minimum value of the sequence. The START WITH clause can be used to start a sequence outside the range that is used for cycles. The range used for cycles is defined by MINVALUE and MAXVALUE.

#### **INCREMENT BY *numeric-constant***

Specifies the interval between consecutive values of the identity column. The value can be any positive or negative value that is not 0, does not exceed the value of a large integer constant, and could be assigned to the column without any non-zero digits existing to the right of the decimal point. The default is 1.

If the value is positive, the sequence of values for the identity column ascends. If the value is negative, the sequence of values descends.

#### **CACHE or NO CACHE**

Specifies whether to keep some preallocated values in memory. Preallocating and storing values in the cache improves the performance of inserting rows into a table.

#### **CACHE *integer***

Specifies the number of values of the identity column sequence that DB2 preallocates and keeps in memory. The minimum value that can be specified is 2, and the maximum is the largest value that can be represented as an integer. The default is 20.

During a system failure, all cached identity column values that are yet to be assigned are lost, and thus, will never be used.

## CREATE TABLE

Therefore, the value specified for CACHE also represents the maximum number of values for the identity column that could be lost during a system failure.

In a data sharing environment, each member gets its own range of consecutive values to assign. For example, if CACHE 20 is specified, DB2A might get values 1-20 for a particular sequence, and DB2B might get values 21-40. Therefore, if transactions from different members generate values for the same identity column, the values that are assigned might not be in the order in which they are requested.

The minimum value is 2. The maximum is the largest value that can be represented as an integer. The default is CACHE 20.

### NO CACHE

Specifies that values for the identity column are not preallocated.

In a data sharing environment, use NO CACHE if you need to guarantee that the identity values are generated in the order in which they are requested.

### CYCLE or NO CYCLE

Specifies whether this identity column should continue to generate values after reaching either the maximum or minimum value of the sequence.

#### CYCLE

Specifies that values continue to be generated for this column after the maximum or minimum value has been reached. If this option is used, after an ascending sequence reaches the maximum value of the sequence, it generates its minimum value. After a descending sequence reaches its minimum value of the sequence, it generates its maximum value. The maximum and minimum values for the column determine the range that is used for cycling.

When CYCLE is in effect, duplicate values can be generated by DB2 for an identity column. However, if a unique index exists on the identity column, and a non-unique value is generated for it, an error occurs.

#### NO CYCLE

Specifies that values will not be generated for the identity column once the maximum or minimum value for the sequence has been reached. This is the default.

### MAXVALUE *numeric-constant*

Specifies the numeric constant that is the maximum value that is generated for this identity column. This value can be any positive or negative value that could be assigned to this column, but the value must be greater than the minimum value.

If a value is not explicitly specified when the identity column is defined, this is the maximum value of the data type (and precision, if DECIMAL) for an ascending sequence; or the **START WITH** value, or -1 if **START WITH** was not specified, for a descending sequence.

**MINVALUE** *numeric-constant*

Specifies the numeric constant that is the minimum value that is generated for this identity column. This value can be any positive or negative value that could be assigned to this column, but the value must be less than the maximum value.

If a value is not explicitly specified when the identity column is defined, this is the **START WITH** value, or 1 if **START WITH** was not specified, for an ascending sequence; or the minimum value of the data type (and precision, if DECIMAL) for a descending sequence.

*references-clause*

The *references-clause* of a *column-definition* provides a shorthand method of defining a foreign key composed of a single column. Thus, if a *references-clause* is specified in the definition of column C, the effect is the same as if that *references-clause* were specified as part of a FOREIGN KEY clause in which C is the only identified column.

Do not specify the *references-clause* in the definition of a LOB or ROWID column because a LOB or ROWID column cannot be a foreign key.

*check-constraint*

The *check-constraint* of a *column-definition* has the same effect as specifying a table check constraint in a separate ADD *check-constraint* clause. For conformance with the SQL standard, a table check constraint specified in the definition of column C should not reference any columns other than C.

Do not specify a table check constraint in the definition of a LOB or ROWID column.

**FIELDPROC** *program-name*

Designates *program-name* as the field procedure exit routine for the column. Writing a field procedure exit routine is described in Appendix B (Volume 2) of *DB2 Administration Guide*. Field procedures can only be specified for short string columns that do not have a nonnull default value. For more information about string comparisons with field procedures, see “String comparisons” on page 73.

The field procedure encodes and decodes column values: before a value is inserted in the column, it is passed to the field procedure for encoding. Before a value from the column is used by a program, it is passed to the field procedure for decoding. A field procedure could be used, for example, to alter the sorting sequence of values entered in the column.

The field procedure is also invoked during the processing of the CREATE TABLE statement. When so invoked, the procedure provides DB2 with the column’s *field description*. The field description defines the data characteristics of the encoded values. By contrast, the information you supply for the column in the CREATE TABLE statement defines the data characteristics of the decoded values.

*constant*

Is a parameter that is passed to the field procedure when it is invoked. A parameter list is optional. The *n*th parameter specified in the FIELDPROC clause on CREATE TABLE corresponds to the *n*th parameter of the specified field procedure. The maximum length of the parameter list is 254 bytes, including commas but excluding insignificant blanks and the delimiting parentheses.

## CREATE TABLE

If you omit FIELDPROC, the column has no field procedure.

End of column-definition

unique-constraint

### **CONSTRAINT** *constraint-name*

Names the primary key or unique key constraint. If a constraint name is not specified, a unique constraint name is generated. If a name is specified, it must be different from the names of any referential, check, primary key, or unique key constraints previously specified on the table.

### **PRIMARY KEY**(*column-name*,...)

Defines a primary key composed of the identified columns. The clause must not be specified more than once and the identified columns must be defined as NOT NULL. Each *column-name* must be an unqualified name that identifies a column of the table except a LOB or ROWID column , and the same column must not be identified more than once. The number of identified columns must not exceed 64, and the sum of their length attributes must not exceed 255.

The table is marked as unavailable until its primary index is explicitly created unless the CREATE TABLE statement is processed by the schema processor. In that case, DB2 implicitly creates an index to enforce the uniqueness of the primary key and the table definition is considered complete. (For more information about implicitly created indexes, see “Implicitly created indexes” on page 677.)

### **UNIQUE**(*column-name*,...)

Defines a unique key composed of the identified columns. Each *column-name* must be an unqualified name that identifies a column of the table except a LOB column, and the same column must not be identified more than once. Each identified column must be defined as NOT NULL. The number of identified columns must not exceed 64 and the sum of their length attributes must not exceed 255.

A unique key is a duplicate if it is the same as the primary key or a previously defined unique key. The specification of a duplicate unique key is ignored with a warning.

The table is marked as unavailable until all the required indexes are explicitly created unless the CREATE TABLE statement is processed by the schema processor. In that case, DB2 implicitly creates the indexes that are required for the unique keys and the table definition is considered complete. (For more information about implicitly created indexes, see “Implicitly created indexes” on page 677.)

End of unique-constraint

referential-constraint

### **CONSTRAINT** *constraint-name*

Names the referential constraint. If a constraint name is not specified, a unique constraint name is generated. If a name is specified, it must be different from the names of any referential, check, primary key, or unique key constraints previously specified on the table.

**FOREIGN KEY (*column-name*,...)** **references-clause**

Each specification of the FOREIGN KEY clause defines a referential constraint with the specified name.

The foreign key of the referential constraint is composed of the identified columns. Each *column-name* must be an unqualified name that identifies a column of the table except a LOB or ROWID column, and the same column must not be identified more than once. The number of identified columns must not exceed 64, and the sum of their length attributes must not exceed 255 minus the number of columns that allow null values. The referential constraint is a duplicate if the FOREIGN KEY and parent table are the same as the FOREIGN KEY and parent table of a previously defined referential constraint. The specification of a duplicate referential constraint is ignored with a warning.

**End of referential-constraint**

**references-clause****REFERENCES** *table-name* (*column-name*,...)

The table name specified after REFERENCES must identify a table that exists at the current server<sup>35</sup>, but it must not identify a catalog table. In the following discussion, let T2 denote an identified table and let T1 denote the table that you are creating (T1 and T2 cannot be the same table<sup>35</sup>).

T2 must have a unique index and the privilege set must include the ALTER or REFERENCES privilege on the parent table, or the REFERENCES privilege on the columns of the nominated parent key.

The parent key of the referential constraint is composed of the identified columns. Each *column-name* must be an unqualified name that identifies a column of T2. The identified column cannot be a LOB or a ROWID column. The same column must not be identified more than once.

The list of column names in the parent key must be identical to the list of column names in a primary key or unique key in the parent table T2. The column names must be specified in the *same order* as in the primary key or unique key.

If a list of column names is not specified, then T2 must have a primary key. Omission of a list of column names is an implicit specification of the columns of the primary key for T2.

The specified foreign key must have the same number of columns as the parent key of T2 and, except for their names, default values, null attributes and check constraints, the description of the *n*th column of the foreign key must be identical to the description of the *n*th column of the nominated parent key. If the foreign key includes a column defined as a distinct type, the corresponding column of the nominated parent key must be the same distinct type. If a column of the foreign key has a field procedure, the corresponding column of the nominated parent key must have the same field procedure and an identical field description. A field description is a description of the encoded value as it is stored in the database for a column that has been defined to have an associated field procedure.

---

35. This restriction is relaxed when the statement is processed by the schema processor and the other table is created within the same CREATE SCHEMA.

## CREATE TABLE

The referential constraint specified by a FOREIGN KEY clause defines a relationship in which T2 is the parent and T1 is the dependent. A description of the referential constraint is recorded in the catalog.

### ON DELETE

The delete rule of the relationship is determined by the ON DELETE clause. For more on the concepts used here, see “Referential constraints” on page 7.

SET NULL must not be specified unless some column of the foreign key allows null values. The default value for the rule depends on the value of the CURRENT RULES special register when the CREATE TABLE statement is processed. If the value of the register is 'DB2', the delete rule defaults to RESTRICT; if the value is 'STD', the delete rule defaults to NO ACTION.

The delete rule applies when a row of T2 is the object of a DELETE or propagated delete operation and that row has dependents in T1. Let  $p$  denote such a row of T2. Then:

- If RESTRICT or NO ACTION is specified, an error occurs and no rows are deleted.
- If CASCADE is specified, the delete operation is propagated to the dependents of  $p$  in T1.
- If SET NULL is specified, each nullable column of the foreign key of each dependent of  $p$  in T1 is set to null.

Let T3 denote a table identified in another FOREIGN KEY clause (if any) of the CREATE TABLE statement. The delete rules of the relationships involving T2 and T3 must be the same and must not be SET NULL if:

- T2 and T3 are the same table.
- T2 is a descendent of T3 and the deletion of rows from T3 cascades to T2.
- T2 and T3 are both descendants of the same table and the deletion of rows from that table cascades to both T2 and T3.

### End of references-clause

### check-constraint

#### CONSTRAINT *constraint-name*

Names the table check constraint. The constraint name must be different from the names of any referential, check, primary key, or unique key constraints previously specified on the table.

If constraint-name is not specified, a unique constraint name is derived from the name of the first column in the check-condition specified in the definition of the table check constraint.

#### CHECK (*check-condition*)

Defines a table check constraint. A check-condition can evaluate to unknown if a column that is an operand of the predicate is null. A check-condition that evaluates to unknown does not violate the check constraint. A *check-condition* is a search condition, with the following restrictions:

- It can refer only to columns of table *table-name*; however, the columns cannot be LOB or ROWID columns.

- It can be up to 3800 bytes long, not including redundant blanks.
- It must not contain any of the following:
  - Subselects
  - Built-in or user-defined functions
  - Cast functions other than those created when the distinct type was created
  - Host variables
  - Parameter markers
  - Special registers
  - Columns that include a field procedure
  - CASE Expressions
  - Quantified predicates
  - EXISTS predicates
- If a check-condition refers to a long string column, the reference must occur within a LIKE predicate.
- The AND and OR logical operators can be used between predicates. The NOT logical operator cannot be used.
- The first operand of every predicate must be the column name of a column in the table.
- The second operand in the check-condition must be either a constant or a column name of a column in the table.
  - If the second operand of a predicate is a constant, and if the constant is:
    - A floating-point number, then the column data type must be floating point.
    - A decimal number, then the column data type must be either floating point or decimal.
    - An integer number, then the column data type must not be a small integer.
    - A small integer number, then the column data type must be small integer.
    - A decimal constant, then its precision must not be larger than the precision of the column.
  - If the second operand of a predicate is a column, then both columns of the predicate must have:
    - The same data type.
    - Identical descriptions with the exception that the specification of the NOT NULL and DEFAULT clauses for the columns can be different, and that string columns with the same data type can have different length attributes

End of check-constraint

#### **LIKE** *table-name* or *view-name*

Specifies that the columns of the table have exactly the same name and description as the columns of the identified table or view. The name specified after LIKE must identify a table or view that exists at the current server or a declared temporary table. The privilege set must implicitly or explicitly include the SELECT privilege on the identified table or view. If the identified table or view contains a column with a distinct type, the USAGE privilege on the distinct type is also needed. An identified table must not be an auxiliary table. An identified view must not include a column that is considered to be a ROWID column or an identity column. (For more information, see “Notes” on page 675.)

## CREATE TABLE

The use of LIKE is an implicit definition of *n* columns, where *n* is the number of columns in the identified table or view. The implicit definition includes all attributes of the *n* columns as they are described in SYSCOLUMNS with these exceptions:

- When a table is identified in the LIKE clause and a column in the table has a field procedure, the corresponding column of the new table has the same field procedure and the field description. However, the field procedure is not invoked during the execution of the CREATE TABLE statement.
- When a table is identified in the LIKE clause and a column in the table an identity column, the corresponding column of the new table inherits only the data type of the identity column; none of the identity attributes of the column are inherited unless the INCLUDING IDENTITY clause is specified.
- When a view is identified in the LIKE clause, the default value that is associated with the corresponding column of the new table depends on the column of the underlying base table for the view. If the column of the base table does not have a default, the new column does not have a default. If the column of the base table has a default, the default of the new column is:
  - Null if the column of the underlying base table allows nulls.
  - The default for the data type of the underlying base table if the underlying base table does not allow nulls.

The above defaults are chosen regardless of the current default of the base table column. Also, no column in the new table has a field procedure because the catalog descriptions of view columns do not include field procedures.

The implicit definition does not include any other attributes of the identified table or view. For example, the new table does not have a primary key or foreign key. The table is created in the table space implicitly or explicitly specified by the IN clause, and the table has any other optional clause only if the optional clause is specified.

### INCLUDING IDENTITY COLUMN ATTRIBUTES

Specifies that a column of the new table inherits all of the identity attributes of the identity column. If the table identified by LIKE does not have an identity column, the INCLUDING IDENTITY clause is ignored. If the LIKE clause identifies a view, INCLUDING IDENTITY COLUMN ATTRIBUTES cannot be specified.

### WITH RESTRICT ON DROP

Indicates that the table can be dropped only by using REPAIR DBD DROP. In addition, the database and table space that contain the table can be dropped only by using REPAIR DBD DROP.

#### IN *database-name.table-space-name* or IN DATABASE *database-name*

Names the database and table space in which the table is created. Both forms are optional; the default is IN DATABASE DSNDB04.

You can name a database (with *database-name*), a table space (with *table-space-name*), or both. If you name a database, it must be described in the current server's catalog, and must not be DSNDB06 or a work file database.

If you use IN DATABASE, either explicitly or by default, a table space is implicitly created in *database-name*. The name of the table space is derived from the table name. Its other attributes are those it would have if it were created by a CREATE TABLESPACE statement with all optional clauses omitted.

If you name a table space, it must not be one that was created implicitly, be a partitioned table space that already contains a table, or be a LOB table space. If you name a partitioned table space, you cannot load or use the table until its partitioned index is created.

If you name both a database and a table space, the table space must belong to the database you name. If you name only a table space, it must belong to database DSNDB04.

To create a table space implicitly, the privilege set must have: SYSADM or SYSCTRL authority; DBADM, DBCTRL, or DBMAINT authority for the database; or the CREATETS privilege for the database. You must also have the USE privilege for the database's default buffer pool and default storage group.

If you name a table space, you must have SYSADM or SYSCTRL authority, DBADM authority for the database, or the USE privilege for the table space.

#### **EDITPROC** *program-name*

Designates *program-name* as the edit routine for the table. The edit routine, which must be provided by the current server's site, is invoked during the execution of LOAD, INSERT, UPDATE, and all row retrieval operations on the table.

An edit routine receives an entire table row, and can transform that row in any way. Also, it receives a transformed row and must change the row back to its original form. For information on writing an EDITPROC exit routine, see Appendix B (Volume 2) of *DB2 Administration Guide*.

You must not specify an edit routine for a table with a LOB, ROWID, or identity column.

If you omit EDITPROC, the table has no edit procedure.

#### **VALIDPROC** *program-name*

Designates *program-name* as the validation exit routine for the table. Writing a validation exit routine is described in Appendix B (Volume 2) of *DB2 Administration Guide*.

The validation routine can inhibit a load, insert, update, or delete operation on any row of the table: before the operation takes place, the procedure is passed the row. The values represented by any LOB columns in the table are not passed. After examining the row, the procedure returns a value that indicates whether the operation should proceed. A typical use is to impose restrictions on the values that can appear in various columns.

A table can have only one validation procedure at a time. In an ALTER TABLE statement, you can designate a replacement procedure or discontinue the use of a validation procedure.

If you omit VALIDPROC, the table has no validation routine.

#### **AUDIT**

Identifies the types of access to this table that causes auditing to be performed. For information about audit trace classes, see Part 3 (Volume 1) of *DB2 Administration Guide*.

#### **NONE**

Specifies that no auditing is to be done when this table is accessed. This is the default.

#### **CHANGES**

Specifies that auditing is to be done when the table is accessed during the

## CREATE TABLE

first insert, update, or delete operation performed by each unit of work. However, the auditing is done only if the appropriate audit trace class is active.

### ALL

Specifies that auditing is to be done when the table is accessed during the first operation of any kind performed by each unit of work of a utility or application process. However, the auditing is done only if the appropriate audit trace class is active and the access is not performed with COPY, RECOVER, REPAIR, or any stand-alone utility.

If the table is altered with an ALTER TABLE statement, the ALTER TABLE statement is audited only if AUDIT CHANGES or AUDIT ALL is specified and the appropriate audit trace class is active.

### OBID *integer*

Identifies the OBID to be used for this table. An OBID is the identifier for an object's internal descriptor. The integer must not identify an existing or previously used OBID of the database. If you omit OBID, DB2 generates a value.

The following statement retrieves the value of OBID:

```
SELECT OBID
 FROM SYSIBM.SYSTABLES
 WHERE CREATOR = 'ccc' AND NAME = 'nnn';
```

Here, *nnn* is the table name and *ccc* is the table's creator.

### DATA CAPTURE

Specifies whether the logging of SQL INSERT, UPDATE, and DELETE operations on the table is augmented by additional information. For guidance on intended uses of the expanded log records, see:

- The description of data propagation to IMS in *DataPropagator NonRelational MVS/ESA Administration Guide*
- The instructions for using Remote Recovery Data Facility (RRDF) in *Remote Recovery Data Facility Program Description and Operations*
- The instructions for reading log records in Appendix C (Volume 2) of *DB2 Administration Guide*

### NONE

Do not record additional information to the log. This is the default.

### CHANGES

Write additional data about SQL updates to the log. Information about the values that are represented by any LOB columns is not available.

### CCSID *encoding-scheme*

Specifies the encoding scheme for string data stored in the table. If the IN clause is specified, the value must agree with the encoding scheme that is already in use for the table space or database specified in the IN clause. The specific CCSIDs for SBCS, mixed, and graphic data are determined by the table space or database specified in the IN clause. If the IN clause is not specified, the value specified is used for the table being created as well as for the table space that DB2 implicitly creates. The specific CCSIDs for SBCS, mixed, and graphic data are determined by the default CCSIDs for the server for the specified encoding scheme. The valid values are ASCII, EBCDIC, and UNICODE.

If the CCSID clause is not specified, the encoding scheme for the table depends on the IN clause:

- If the IN clause is specified, the encoding scheme already in use for the table space or database specified in the IN clause is used.
- If the IN clause is not specified, the encoding scheme of the new table is the same as the scheme for the table that is specified in the LIKE clause.

## Notes

**Table design:** Designing tables is part of the process of database design. For information on design, see *An Introduction to DB2 for OS/390*.

**Creating a table while a utility runs:** You cannot use CREATE TABLE while a DB2 utility has control of the table space implicitly or explicitly specified by the IN clause.

**Creating a table in a segmented table space:** A table cannot be created in a segmented table space if:

- The available space in the data set is less than the segment size specified for the table space, and
- The data set cannot be extended.

**Creating a table with DBCS and mixed columns:** You cannot create an ASCII or EBCDIC table with a GRAPHIC, VARGRAPHIC, or DBCLOB column or a CHAR, VARCHAR, or CLOB column defined as FOR MIXED DATA when the setting for installation option MIXED DATA is NO.

**Distinct type columns based on LOB and ROWID columns:** Because a distinct type is subject to the same restrictions as its source type, all the syntactic rules that apply to LOB columns (CLOB, DBCLOB, and BLOB) and ROWID columns apply to distinct type columns that are sourced on LOBs and row IDs. For example, a table cannot have both a ROWID column and a column with a distinct type that is sourced on a row ID.

**Creating a table with LOB columns:** If you create a base table with a LOB column (CLOB, DBCLOB, or BLOB), you must also define a ROWID column for the table. The definition of the table is marked incomplete until an auxiliary table is created in a LOB table space for each LOB column in the base table and index is created on each auxiliary table. The auxiliary table stores the actual values of a LOB column. If you create a table with a LOB column in a partitioned table space, there must be one auxiliary table defined for each partition of the base table space.

Unless DB2 implicitly creates the LOB table space, auxiliary table, and index on the auxiliary table for each LOB column in the base table, you need to create these objects using the CREATE TABLESPACE, CREATE AUXILIARY TABLE, and CREATE INDEX statements.

If the value of special register CURRENT RULES is 'STD' when the CREATE STATEMENT is processed, DB2 implicitly creates the LOB table space, auxiliary table, and index on the auxiliary table for each LOB column in the base table. DB2 chooses the names of implicitly created objects using these conventions:

LOB table space

Name is 8 characters long, consisting of an 'L' followed by 7 random characters.

## CREATE TABLE

auxiliary table Name is 18 characters long. The first five characters of the name are the first five characters of the name of the base table. The second five characters are the first five characters of the name of the LOB column. The last eight characters are randomly generated. If a base table name or a LOB column name is less than five characters, DB2 adds underscore characters to the name to pad it to a length of five characters.

index on the auxiliary table

Name is 18 characters long. The first character of the name is an 'I'. The next ten characters are the first ten characters of the name of the auxiliary table. The last seven characters are randomly generated. The index has the COPY NO attribute.

The other attributes of these implicitly created objects are those that would have been created by their respective CREATE statements with all optional clauses omitted, with the following exceptions:

- The database name is the database name of the base table.
- If the size of the LOB column is greater than 1 GB, the LOG option for the LOB table space is LOG NO.

Utility REPORT TABLESPACESET identifies the LOB table spaces that DB2 implicitly created.

**Maximum record size:** The maximum record size of a table depends on the page size of the table space and whether the EDITPROC clause is specified, as shown in Table 47. The page size of the table space is the size of its buffer, which is determined by the BUFFERPOOL clause that was explicitly or implicitly specified when the table space was created.

Table 47. Maximum Record Size, in Bytes

| EDITPROC | Page Size<br>= 4KB | Page Size<br>= 8KB | Page Size<br>= 16KB | Page Size<br>= 32KB |
|----------|--------------------|--------------------|---------------------|---------------------|
| NO       | 4056               | 8138               | 16330               | 32714               |
| YES      | 4046               | 8128               | 16320               | 32704               |

The maximum record size corresponds to the maximum length of a VARCHAR column if that column is the only column in the table.

**Byte counts:** The sum of the byte counts of the columns must not exceed the maximum row size of the table. The maximum row size is eight less than the maximum record size.

For columns that do not allow null values, Table 48 gives the byte counts of columns by data type. For columns that allow null values, the byte count is one more than shown in the table.

Table 48. Byte Counts of Columns by Data Type

| Data Type         | Byte Count                                                                                                   |
|-------------------|--------------------------------------------------------------------------------------------------------------|
| INTEGER           | 4                                                                                                            |
| SMALLINT          | 2                                                                                                            |
| FLOAT( <i>n</i> ) | If <i>n</i> is between 1 and 21, the byte count is 4. If <i>n</i> is between 22 and 53, the byte count is 8. |

Table 48. Byte Counts of Columns by Data Type (continued)

| Data Type         | Byte Count                                                                                  |
|-------------------|---------------------------------------------------------------------------------------------|
| DECIMAL           | INTEGER( $p/2$ ) + 1, where $p$ is the precision                                            |
| CHAR( $n$ )       | $n$                                                                                         |
| VARCHAR( $n$ )    | $n+2$ (For LONG VARCHAR, see "Byte count of a LONG VARCHAR or LONG VARGRAPHIC column.")     |
| CLOB              | 6                                                                                           |
| BLOB              | 6                                                                                           |
| GRAPHIC( $n$ )    | $2n$                                                                                        |
| VARGRAPHIC( $n$ ) | $2n+2$ (For LONG VARGRAPHIC, see "Byte count of a LONG VARCHAR or LONG VARGRAPHIC column.") |
| DBCLOB            | 6                                                                                           |
| DATE              | 4                                                                                           |
| TIME              | 3                                                                                           |
| TIMESTAMP         | 10                                                                                          |
| ROWID             | 19                                                                                          |
| distinct type     | The length of the source data type upon which the distinct type was based                   |

**Byte count of a LONG VARCHAR or LONG VARGRAPHIC column:** To calculate the byte count, let:

- $m$  be the maximum row size (8 less than the maximum record size)
- $i$  be the sum of the byte counts of all columns in the table that are not LONG VARCHAR or LONG VARGRAPHIC
- $j$  be the number of LONG VARCHAR and LONG VARGRAPHIC columns in the table
- $k$  be the number of LONG VARCHAR and LONG VARGRAPHIC columns that allow nulls.

The count is  $2 * (\text{INTEGER}((\text{INTEGER}((m-i-k)/j))/2))$ .

**Length of a LONG column:** To find the character count:

1. Find the byte count from "Byte count of a LONG VARCHAR or LONG VARGRAPHIC column."
2. Subtract 2.
3. If the data type is LONG VARGRAPHIC, divide the result by 2. If the result is not an integer, drop the fractional part.

**Implicitly created indexes:** When the PRIMARY KEY or UNIQUE clause is used in the CREATE TABLE statement and the CREATE TABLE statement is processed by the schema processor, DB2 implicitly creates the unique indexes used to enforce the uniqueness of the primary or unique keys. Each index is created as if the following CREATE INDEX statement were issued:

```
CREATE UNIQUE INDEX xxx ON table-name (column1,...)
```

Where:

- $xxx$  is the name of the index that DB2 generates.
- $table-name$  is the name of the table specified in the CREATE TABLE statement.

## CREATE TABLE

- *(column1,...)* is the list of column names that were specified in the UNIQUE or PRIMARY KEY clause of the CREATE TABLE statement.

For more information about the schema processor, see Part 2 (Volume 1) of *DB2 Administration Guide*.

**Creating a table like a view:** If the LIKE clause is specified and the definition of the table is being based on a view, the view must not include a ROWID column or an identity column. A view column is considered to be an identity column if the corresponding column of the table or view indirectly or directly maps to the name of an identity column in a base table with these exceptions:

- The select-list of the view definition identifies the same identity column more than once.
- The select-list of the view definition references multiple identity columns and thus involves a join.
- A column in the view definition includes an expression that refers to an identity column.
- The view definition includes a set operation (a union).

**Using an identity column:** When a table has an identity column, DB2 can automatically generate sequential numeric values for the column as rows are inserted into the table. Thus, identity columns are ideal for primary keys. Identity columns and ROWID columns are similar in that both types of columns contain values that DB2 generates. ROWID columns are used in large object (LOB) table spaces and can be useful in direct-row access. ROWID columns contain values of the ROWID data type, which returns a 40-byte VARCHAR value that is not regularly ascending or descending. ROWID data values are therefore not well suited to many application uses, such as generating employee numbers or product numbers. For data that is not LOB data and that does not require direct-row access, identity columns are usually a better approach, because identity columns contain existing numeric data types and can be used in a wide variety of uses for which ROWID values would not be suitable.

When a table is recovered to a point-in-time, it is possible that a large gap in the sequence of generated values for the identity column might result. For example, assume a table has an identity column that has an incremental value of 1 and that the last generated value at time T1 was 100 and DB2 subsequently generates values up to 1000. Now, assume that the table space is recovered back to time T1. The generated value of the identity column for the next row that is inserted after the recovery completes will be 1001, leaving a gap from 100 to 1001 in the values of the identity column.

Sometimes you may need to change the attributes of an identity column. For example, if you had defined an identity column with a data type of SMALLINT and then run out of assignable values, you need to redefine the column as INTEGER. To change the attributes of an identity column, you unload the data from the table, drop the table, recreate the table, and reload the data.

But when you recreate the table, you must specify GENERATED BY DEFAULT and a new START WITH value for the identity column. Using GENERATED BY DEFAULT allows LOAD to reload the previously existing identity column values. You cannot use GENERATED ALWAYS in this case, but not using it is not a problem since DB2 always generates a value if a column value is not provided during insertion of an identity column defined as GENERATED BY DEFAULT. The new

START WITH value should be the next value in the sequence from where the original sequence of values left off; this value is the next value that DB2 would generate first.

When wrapping is in effect, duplicate values for a column are allowed even when the column is GENERATED ALWAYS, unless a unique index is defined on the column. If there is a unique index on the column and all possible values have been used, wrapping is possible only if rows are uploaded or deleted before wrapping occurs.

**Using tables with different encoding schemes:** The CCSID clause determines whether the data for a table is encoded in ASCII, EBCDIC, or Unicode. All created tables that are referenced in an SQL statement must have the same encoding scheme—the tables must be either all ASCII, all EBCDIC, or all Unicode. Once the data is created in a table with the CREATE TABLE statement, you cannot mix encoding schemes.

**Dropping a table in a partitioned table space:** You can only drop a table in a partitioned table space by using the DROP TABLESPACE statement.

## Examples

*Example 1:* Create a table named DSN8710.DEPT in the table space DSN8S71D of the database DSN8D71A. Name the table's five columns DEPTNO, DEPTNAME, MGRNO, ADMRDEPT, and LOCATION, allowing only MGRNO to contain nulls, and designating DEPTNO as the only column in the table's primary key. All five columns hold character string data. Assuming a value of NO for the field MIXED DATA on installation panel DSNTIPF, all five columns have the subtype SBCS.

```
CREATE TABLE DSN8710.DEPT
 (DEPTNO CHAR(3) NOT NULL,
 DEPTNAME VARCHAR(36) NOT NULL,
 MGRNO CHAR(6) ,
 ADMRDEPT CHAR(3) NOT NULL,
 LOCATION CHAR(16) ,
 PRIMARY KEY(DEPTNO))
IN DSN8D71A.DSN8S71D;
```

*Example 2:* Create a table named DSN8710.PROJ in an implicitly created table space of the database DSN8D71A. Assign the table a validation procedure named DSN8EAPR.

```
CREATE TABLE DSN8710.PROJ
 (PROJNO CHAR(6) NOT NULL,
 PROJNAME VARCHAR(24) NOT NULL,
 DEPTNO CHAR(3) NOT NULL,
 RESPEMP CHAR(6) NOT NULL,
 PRSTAFF DECIMAL(5,2) ,
 PRSTDATE DATE ,
 PRENDATE DATE ,
 MAJPROJ CHAR(6) NOT NULL)
IN DATABASE DSN8D71A
VALIDPROC DSN8EAPR;
```

*Example 3:* Assume that table PROJECT has a non-primary unique key that consists of columns DEPTNO and RESPEMP (the department number and employee responsible for a project). Create a project activity table named ACTIVITY with a foreign key on that unique key.

```
CREATE TABLE ACTIVITY
 (PROJNO CHAR(6) NOT NULL,
 ACTNO SMALLINT NOT NULL,
```

## CREATE TABLE

```
ACTDEPT CHAR(3) NOT NULL,
ACTOWNER CHAR(6) NOT NULL,
ACSTAFF DECIMAL(5,2) ,
ACSTDATE DATE NOT NULL,
ACENDATE DATE ,
FOREIGN KEY (ACTDEPT,ACTOWNER)
 REFERENCES PROJECT (DEPTNO,RESPEMP) ON DELETE RESTRICT)
IN DSN8D71A.DSN8S71D;
```

*Example 4:* Create an employee photo and resume table EMP\_PHOTO\_RESUME that complements the sample employee table. The table contains a photo and resume for each employee. Put the table in table space DSN8D71A.DSN8S71E. Let DB2 always generate the values for the ROWID column.

```
CREATE TABLE DSN8T10.EMP_PHOTO_RESUME
(EMPNO CHAR(6) NOT NULL,
EMP_ROWID ROWID NOT NULL GENERATED ALWAYS,
EMP_PHOTO BLOB(110K),
RESUME CLOB(5K),
PRIMARY KEY(EMPNO))
IN DSN8D71A.DSN8S71E
CCSID EBCDIC;
```

*Example 5:* Create an EMPLOYEE table with an identity column named EMP\_NO. Define the identity column so that DB2 will always generate the values for the column. Use the default value, which is 1, for the first value that should be assigned and for the incremental difference between the subsequently generated consecutive numbers.

```
CREATE TABLE EMPLOYEE
(EMPNO INTEGER GENERATED ALWAYS AS IDENTITY,
ID SMALLINT,
NAME CHAR(30),
SALARY DECIMAL(5,2),
DEPTNO SMALLINT)
IN DSN8D71A.DSN8S71D;
```

---

## CREATE TABLESPACE

The CREATE TABLESPACE statement defines a simple, segmented, or partitioned table space at the current server.

### Invocation

This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is implicitly or explicitly specified.

### Authorization

The privilege set that is defined below must include at least one of the following:

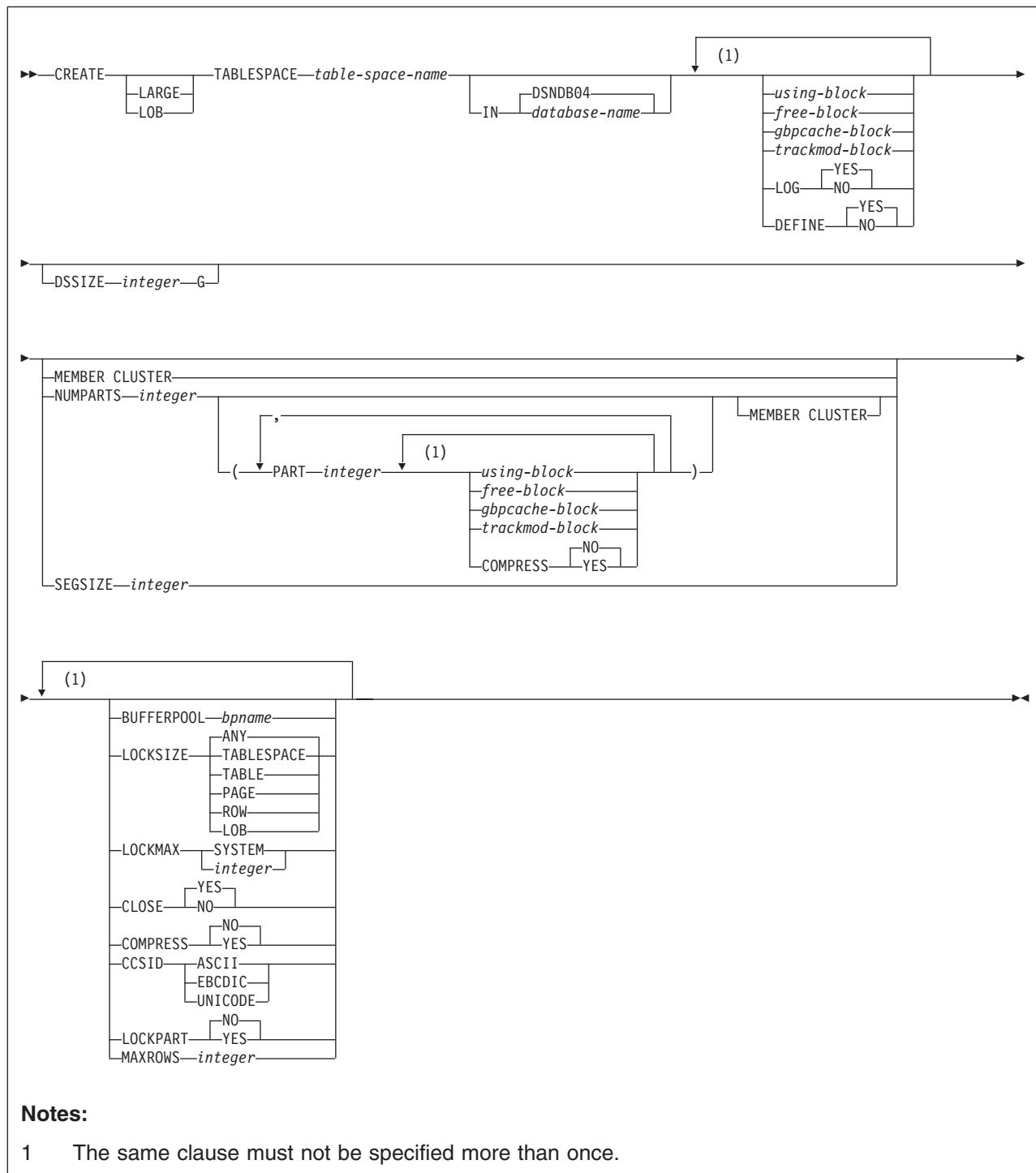
- The CREATETS privilege for the database
- DBADM, DBCTRL, or DBMAINT authority for the database
- SYSADM or SYSCTRL authority

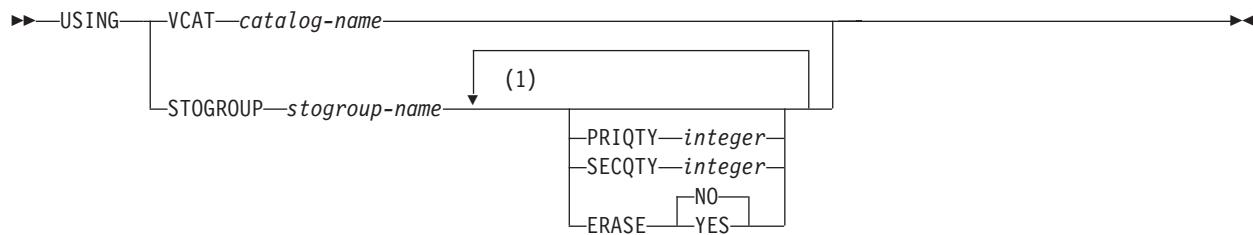
Additional privileges might be required, as explained in the description of the BUFFERPOOL and USING STOGROUP clauses.

**Privilege set:** If the statement is embedded in an application program, the privilege set is the privileges that are held by the authorization ID of the owner of the plan or package. If the statement is dynamically prepared, the privilege set is the privileges that are held by the SQL authorization ID of the process.

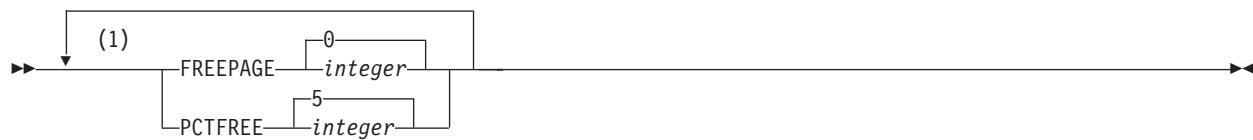
## CREATE TABLESPACE

### Syntax



**using-block:****Notes:**

- 1 The same clause must not be specified more than once.

**free-block:****Notes:**

- 1 The same clause must not be specified more than once.

**gbpcache-block:****trackmod-block:****Description****LARGE**

Identifies that each partition of a partitioned table space has a maximum partition size of 4 GB, which enables the table space to contain more than 64

## CREATE TABLESPACE

GB of data. The preferred method to specify a maximum partition size of 4 GB and larger is the DSSIZE clause. The LARGE clause is for compatibility of releases of DB2 for OS/390 and z/OS prior to Version 6. Do not specify LARGE if LOB or DSSIZE is specified.

### LOB

Identifies the table space as LOB table space. A LOB table space is used to hold LOB values.

The LOB table space must be in the same database as its associated base table space.

### *table-space-name*

Names the table space. The name, qualified with the *database-name* implicitly or explicitly specified by the IN clause, must not identify a table space, index space, or LOB table space that exists at the current server.

A table space that is for declared temporary tables must be in a TEMP database (a database that is defined AS TEMP). PUBLIC implicitly receives the USE privilege (without GRANT authority) on any table space created in the TEMP database. This implicit privilege is not recorded in the DB2 catalog, and it cannot be revoked.

### IN *database-name*

Identifies the database in which the table space is created. The name must identify a database that exists at the current server. DSNDB06 must not be specified for any type of table space, and a work file database must not be specified for a LOB table space. If the table space is for declared temporary tables, a TEMP database (a database that is defined with AS TEMP) must be specified. The default is DSNDB04.

### using-block

The components of the USING clause are discussed below, first for nonpartitioned table spaces and then for partitioned table spaces. If you omit USING, the default storage group of the database must exist.

#### USING Clause for Nonpartitioned Table Spaces:

For nonpartitioned table spaces, the USING clause indicates whether the data set for the table space is defined by you or by DB2. If DB2 is to define the data set, the clause also gives space allocation parameters and an erase rule.

If you omit USING, DB2 defines the data sets using the default storage group of the database and the defaults for PRIQTY, SECQTY, and ERASE.

### VCAT *catalog-name*

Specifies that the first data set for the table space is managed by the user, and following data sets, if needed, are also managed by the user.

The data sets defined for the table space are linear VSAM data sets cataloged in an integrated catalog facility catalog identified by *catalog-name*. Because *catalog-name* is a short identifier, an alias must be used if the catalog name is longer than eight characters.

Conventions for table space data set names are given in Part 2 (Volume 1) of *DB2 Administration Guide*. *catalog-name* is the first qualifier for each data set name.

One or more DB2 subsystems could share integrated catalog facility catalogs with the current server. To avoid the chance of having one of those subsystems attempt to assign the same name to different data sets, select a value for *catalog-name* that is not used by the other DB2 subsystems.

**STOGROUP *stogroup-name***

Specifies that DB2 will define and manage the data sets for the table space. Each data set will be defined on a volume of the identified storage group. The values specified (or the defaults) for PRIQTY and SECQTY determine the primary and secondary allocations for the data set. The storage group supplies the name of a volume for the data set and the first-level qualifier for the data set name. The first-level qualifier is also the name of, or an alias for, the integrated catalog facility catalog on which the data set is to be cataloged. The naming conventions for the data set are the same as if the data set is managed by the user. As was mentioned above for VCAT, the first-level qualifier could cause naming conflicts if the local DB2 can share integrated catalog facility catalogs with other DB2 subsystems.

*stogroup-name* must identify a storage group that exists at the current server. SYSADM or SYSCTRL authority, or the USE privilege on the storage group, is required.

The description of the storage group must include at least one volume serial number, or it must indicate that the choice of volumes is left to Storage Management Subsystem (SMS). If volume serial numbers appear in the description, each must identify a volume that is accessible to MVS for dynamic allocation of the data set, and all identified volumes must be of the same device type.

The integrated catalog facility catalog used for the storage group must **not** contain an entry for the first data set of the table space. If the integrated catalog facility catalog is password protected, the description of the storage group must include a valid password.

**PRIQTY *integer***

Specifies the minimum primary space allocation for a DB2-managed data set. The primary space allocation is at least *n* kilobytes, where *n* is the value of *integer* with the following exceptions:

- If PRIQTY *integer* is specified:
  - For 4KB page sizes, if *integer* is less than 12, *n* is 12.
  - For 8KB page sizes, if *integer* is less than 24, *n* is 24.
  - For 16KB page sizes, if *integer* is less than 48, *n* is 48.
  - For 32KB page sizes, if *integer* is less than 96, *n* is 96.
  - For any page size, if *integer* is greater than 4194304 kilobytes, *n* is 4194304.
- If PRIQTY is omitted, *n* is 12, 24, 48, or 96 for 4KB, 8KB, 16KB, and 32KB page sizes, respectively.

If neither the PRIQTY nor SECQTY value is specified and the TSQTY subsystem parameter is nonzero, the TSQTY value is used for the primary and the secondary quantities.

For LOB table spaces, the exceptions are:

- If PRIQTY *integer* is specified:
  - For 4KB page sizes, if *integer* is less than 200, *n* is 200.
  - For 8KB page sizes, if *integer* is less than 400, *n* is 400.
  - For 16KB page sizes, if *integer* is less than 800, *n* is 800.
  - For 32KB page sizes, if *integer* is less than 1600, *n* is 1600.
  - For any page size, if *integer* is greater than 4194304 kilobytes, *n* is 4194304.

```


#
```

## CREATE TABLESPACE

- If PRIQTY is omitted,  $n$  is 200, 400, 800, or 1600 for 4KB, 8KB, 16KB, and 32KB page sizes, respectively.

```


#
```

If the running DB2 is under DFP 1.5, the maximum value allowed for PRIQTY is 65GB (67108864 kilobytes); otherwise, the existing maximum value of 4GB (4194304 kilobytes) applies.

DB2 specifies the primary space allocation to access method services using the smallest multiple of  $pKB$  not less than  $n$ , where  $p$  is the page size of the table space. The allocated space can be greater than the amount of space requested by DB2. For example, it could be the smallest number of tracks that will accommodate the request. The amount of storage space requested must be available on some volume in the storage group based on VSAM space allocation restrictions. Otherwise, the primary space allocation will fail. To more closely estimate the actual amount of storage, see the description of the DEFINE CLUSTER command in *DFSMS/MVS: Access Method Services for the Integrated Catalog*.

Executing this statement causes only one data set to be created. However, you might have more data than this one data set can hold. DB2 automatically defines more data sets when they are needed. Regardless of the value in PRIQTY, when a data set reaches its maximum size, DB2 creates a new one. To avoid wasting space, use the following formula to make sure that PRIQTY and its associated secondary extent values do not exceed the maximum size of the data set:

PRIQTY + (number of extents \* SECQTY) <= DSSIZE (implicit or explicit)

### SECQTY *integer*

Specifies the minimum secondary space allocation for a DB2-managed data set. The secondary space allocation is at least  $n$  kilobytes, where  $n$  is the value of *integer* with the following exceptions:

- If SECQTY *integer* is specified and *integer* is greater than 4194304,  $n$  is 4194304 kilobytes. A value of 0 for *integer* indicates that no data set can be extended.
- If SECQTY and PRIQTY are omitted:
  - For 4KB page sizes,  $n$  is 12.
  - For 8KB page sizes,  $n$  is 24.
  - For 16KB page sizes,  $n$  is 48.
  - For 32KB page sizes,  $n$  is 96.
- If SECQTY is omitted and PRIQTY is specified,  $n$  is either 10% of PRIQTY or 3 times the page size of the table space, whichever is larger. If neither the PRIQTY nor SECQTY value is specified and the TSQTY subsystem parameter is nonzero, the TSQTY value is used for the primary and the secondary quantities.

For LOB table spaces the exceptions are:

- If SECQTY *integer* is specified:
  - For 4KB page sizes, if *integer* is greater than 0 and less than 200,  $n$  is 200.
  - For 8KB page sizes, if *integer* is greater than 0 and less than 400,  $n$  is 400.
  - For 16KB page sizes, if *integer* is greater than 0 and less than 800,  $n$  is 800.

```


#
```

- For 32KB page sizes, if *integer* is greater than 0 and less than 1600, *n* is 1600.
- For any page size, if *integer* is greater than 4194304 kilobytes, *n* is 4194304.
- If SECQTY is omitted, *n* is either 10% of PRIQTY or 50 times the page size of the table space, whichever is larger.

#

The maximum value allowed for SECQTY is 4GB (4194304 kilobytes).

DB2 specifies the secondary space allocation to access method services using the smallest multiple of *pKB* not less than *n*, where *p* is the page size of the table space. The allocated space can be greater than the amount of space requested by DB2. For example, it could be the smallest number of tracks that will accommodate the request. To more closely estimate the actual amount of storage, see the description of the DEFINE CLUSTER command in *DFSMS/MVS: Access Method Services for the Integrated Catalog*.

#### **ERASE**

Indicates whether the DB2-managed data sets for the table space or partition are to be erased when they are deleted during the execution of a utility or an SQL statement that drops the table space.

#### **NO**

Does not erase the data sets. Operations involving data set deletion will perform better than ERASE YES. However, the data is still accessible, though not through DB2. This is the default.

#### **YES**

Erases the data sets. As a security measure, DB2 overwrites all data in the data sets with zeros before they are deleted.

#### **USING Clause for Partitioned Table Spaces:**

If the table space is partitioned, there is a USING clause for each partition; either one you give explicitly or one provided by default. Except as explained below, the meaning of the clause and the rules that apply to it are the same as for a nonpartitioned table space.

The USING clause for a particular partition is the first of these choices that can be found:

- A USING clause in the PART clause for the partition
- A USING clause that is not in any PART clause
- An implicit USING STOGROUP clause that identifies the default storage group of the database and accepts the defaults for PRIQTY, SECQTY, and ERASE

#### **VCAT *catalog-name***

Indicates that the data set for the partition is managed by the user using the naming conventions set forth in Part 2 (Volume 1) of *DB2 Administration Guide*. As was true for the nonpartitioned case, *catalog-name* identifies the catalog for the data set and supplies the first-level qualifier for the data set name.

One or more DB2 subsystems could share integrated catalog facility catalogs with the current server. To avoid the chance of having one of those subsystems attempt to assign the same name to different data sets, select a value for *catalog-name* that is not used by the other DB2 subsystems.

DB2 assumes one and only one data set for each partition.

## CREATE TABLESPACE

### **STOGROUP** *stogroup-name*

Indicates that DB2 will create a data set for the partition with the aid of a storage group named *stogroup-name*. The data set is defined during the execution of this statement. DB2 assumes one and only one data set for each partition.

The *stogroup-name* must identify a storage group that exists at the current server and the privilege set must include SYSADM authority, SYSCTRL authority, or the USE privilege for the storage group. The integrated catalog facility catalog used for the storage group must **not** contain an entry for that data set.

When USING STOGROUP is specified for a partition, the defaults for PRIQTY, SECQTY, and ERASE are the values specified in the USING STOGROUP clause that is not in any PART clause. If that USING STOGROUP clause is not specified, the defaults are those specified in the description of PRIQTY, SECQTY, and ERASE.

**End of using-block**

**free-block**

### **FREEPAGE** *integer*

Specifies how often to leave a page of free space when the table space or partition is loaded or reorganized. You must specify an integer in the range 0 to 255. If you specify 0, no pages are left as free space. Otherwise, one free page is left after every *n* pages, where *n* is the specified integer. However, if the table space is segmented and the integer you specify is not less than the segment size, *n* is one less than the segment size.

If the table space is segmented, the number of pages left free must be less than the SEGSIZE value. If the number of pages to be left free is greater than or equal to the SEGSIZE value, then the number of pages is adjusted downward to one less than the SEGSIZE value.

The default is FREEPAGE 0, leaving no free pages. Do not specify FREEPAGE for a LOB table space, or a table space in a work file database or a TEMP database.

### **PCTFREE** *integer*

Indicates what percentage of each page to leave as free space when the table is loaded or reorganized. *integer* can range from 0 to 99. The first record on each page is loaded without restriction. When additional records are loaded, at least *integer* percent of free space is left on each page.

The default is PCTFREE 5. Do not specify PCTFREE for a LOB table space, or a table space in a work file database or a TEMP database.

If the table space is partitioned, the values of FREEPAGE and PCTFREE for a particular partition are given by the first of these choices that apply:

- The values of FREEPAGE and PCTFREE given in the PART clause for that partition
- The values given in a *free-block* that is not in any PART clause
- The default values are FREEPAGE 0 and PCTFREE 5.

End of free-block

gbpcache-block

#### **GBPCACHE**

In a data sharing environment, specifies what pages of the table space or partition are written to the group buffer pool in a data sharing environment. In a non-data-sharing environment, you can specify GBPCACHE for a table space other than one in a work file or TEMP database, but it is ignored. Do not specify GBPCACHE for a table space in a work file or TEMP database in either environment (data sharing or non-data-sharing).

#### **CHANGED**

When there is inter-DB2 R/W interest on the table space or partition, updated pages are written to the group buffer pool. When there is no inter-DB2 R/W interest, the group buffer pool is not used. Inter-DB2 R/W interest exists when more than one member in the data sharing group has the table space or partition open, and at least one member has it open for update. GBPCACHE CHANGED is the default.

If the table space is in a group buffer pool that is defined to be used only for cross-validation (GBPCACHE NO), CHANGED is ignored and no pages are cached to the group buffer pool.

#### **ALL**

Indicates that pages are to be cached in the group buffer pool as they are read in from DASD.

**Exception:** In the case of a single updating DB2 when no other DB2s have any interest in the page set, no pages are cached in the group buffer pool.

Hiperpools are not used for indexes or partitions that are defined with GBPCACHE ALL.

If the table space is in a group buffer pool that is defined to be used only for cross-validation (GBPCACHE NO), ALL is ignored and no pages are cached to the group buffer pool.

#### **SYSTEM**

Indicates that only changed system pages within the LOB table space are to be cached to the group buffer pool. A system page is a space map page or any other page that does not contain actual data values.

This is the default for LOB table spaces. You can use SYSTEM only for a LOB table space.

#### **NONE**

Indicates that no pages are to be cached to the group buffer pool. DB2 uses the group buffer pool only for cross-validation.

If you specify NONE, the table space or partition must not be in recover pending status and must be in the stopped state when the CREATE TABLESPACE statement is executed.

**If the table space is partitioned**, the value of GBPCACHE for a particular partition is given by the first of these choices that applies:

1. The value of GBPCACHE given in the PART clause for that partition. Do not use more than one *gbpcache-block* in any PART clause.
2. The value given in a *gbpcache-block* that is not in any PART clause.

## CREATE TABLESPACE

3. The default value CHANGED.

End of **gbpcache-block**

trackmod-block

### TRACKMOD

Specifies whether DB2 tracks modified pages in the space map pages of the table space or partition. Do not specify TRACKMOD for a LOB table space. For a table space in a TEMP database, DB2 uses TRACKMOD NO regardless of the value specified.

#### YES

DB2 tracks changed pages in the space map pages to improve the performance of incremental image copy. YES is the default unless the table space is in a TEMP database.

#### NO

DB2 does not track changed pages in the space map pages. It uses the LRSN value in each page to determine whether a page has been changed.

If the table space is partitioned, the value of TRACKMOD for a particular partition is given by the first of these choices that applies:

1. The value of TRACKMOD given in the PART clause for that partition.
2. The value given in a *trackmod-block* that is not in any PART clause.
3. The default value YES.

End of **trackmod-block**

### LOG

Specifies whether changes to a LOB column in the table space are to be written to the log. You can use the LOG clause only for a LOB table space.

#### YES

Indicates that changes to a LOB column are to be written to the log. You cannot use YES if the auxiliary table in the table space stores a LOB column that is greater than 1 gigabyte in length.

YES is the default.

#### NO

Indicates that changes to a LOB column are not to be written to the log.

LOG NO has no effect on a commit or rollback operation; the consistency of the database is maintained regardless of whether the LOB value is logged. All committed changes and changes that are rolled back reflect the expected results.

Even when LOG NO is specified, changes to system pages and to the auxiliary index are logged. During the log apply operation of the RECOVER utility, LPL recovery, or GPB recovery, all LOB values that were not logged are marked invalid and cannot be accessed by a SELECT or FETCH statement. Invalid LOB values can be updated or deleted.

### DEFINE

Specifies when the underlying data sets for the table space are physically created.

**YES**

The data sets are created when the table space is created (the CREATE TABLESPACE statement is executed). YES is the default.

**NO**

The data sets are not created until data is inserted into the table space. DEFINE NO is applicable only for DB2-managed data sets (USING STOGROUP is specified). DEFINE NO is ignored for user-managed data sets (USING VCAT is specified). DB2 uses the SPACE column in catalog table SYSTABLEPART to record the status of the data sets (undefined or allocated). DEFINE NO is also ignored for a LOB table space.

Do not specify DEFINE NO for a table space in a work file database or a TEMP database; otherwise, an error occurs. DEFINE NO is not recommended if you intend to use any tools outside of DB2 to manipulate data, such as to load data, because data sets might then exist when DB2 does not expect them to exist. When DB2 encounters this inconsistent state, applications will receive an error.

**DSSIZE integer G**

A value in gigabytes that indicates the maximum size for each partition or, for LOB table spaces, each data set. If you specify DSSIZE, you must also specify NUMPARTS or LOB.

The following values are valid:

|             |              |
|-------------|--------------|
| <b>1 G</b>  | 1 gigabyte   |
| <b>2 G</b>  | 2 gigabytes  |
| <b>4 G</b>  | 4 gigabytes  |
| <b>8 G</b>  | 8 gigabytes  |
| <b>16 G</b> | 16 gigabytes |
| <b>32 G</b> | 32 gigabytes |
| <b>64 G</b> | 64 gigabytes |

To specify a value greater than 4 G, the following conditions must be true:

- DB2 is running with DFSMS Version 1 Release 5.
- The data sets for the table space are associated with a DFSMS data class that has been specified with extended format and extended addressability.

For all table spaces except LOB table spaces, if DSSIZE (or LARGE) is omitted, the default for the maximum size of each partition depends on the value of NUMPARTS:

| If NUMPARTS is ... | Maximum partition size is... |
|--------------------|------------------------------|
| 1 to 16            | 4 GB                         |
| 17 to 32           | 2 GB                         |
| 33 to 64           | 1 GB                         |
| 65 to 254          | 4 GB                         |

The partition size shown is not necessarily the actual number of bytes used or allocated for any one partition; it is the largest number that can be logically addressed. Each partition occupies one data set.

For LOB table spaces, if DSSIZE is not specified, the default for the maximum size of each data set is 4 GB. The maximum number of data sets is 254.

When you define a table space with DSSIZE, you automatically give the same size to all indexes that point to that table space.

**MEMBER CLUSTER**

Specifies that data inserted by the INSERT statement is not clustered by the

## CREATE TABLESPACE

implicit clustering index (the first index) or the explicit clustering index. Instead, DB2 chooses where to locate the data in the table space based on available space.

Do not specify MEMBER CLUSTER for a LOB table space, or a table space in a work file database or a TEMP database.

### **NUMPARTS** *integer*

Indicates that the table space is partitioned.

*integer* is the number of partitions, and can range from 1 to 254 inclusive. NUMPARTS must be specified if DSSIZE is specified and LOB is omitted, or LARGE is specified.

The maximum size of each partition depends on the value specified for DSSIZE or LARGE. If DSSIZE or LARGE is not specified, the number of partitions specified determines the maximum size of each partition. For a summary of the values for the maximum size, see the description of "DSSIZE" on page 691.

#  
#  
The number of partitions specified determines the maximum size of the partitioned index key.

If you omit NUMPARTS, the table space is not partitioned and initially occupies one data set. Do not specify NUMPARTS for a LOB table space, or a table space in a work file database or a TEMP database.

### **PART** *integer*

Specifies to which partition the following *using-block* or *free-block* applies. *integer* can range from 1 to the number of partitions given by NUMPARTS.

You can code the PART clause (and any *using-block* or *free-block* that follows it) as many times as needed. If you use the same partition number more than once, only the last specification for that partition is used.

### **BUFFERPOOL** *bpname*

Identifies the buffer pool to be used for the table space and determines the page size of the table space. For 4KB, 8KB, 16KB and 32KB page buffer pools, the page sizes are 4 KB, 8 KB, 16 KB, and 32 KB, respectively. The *bpname* must identify an activated buffer pool, and the privilege set must include SYSADM or SYSCTRL authority, or the USE privilege on the buffer pool. If the table space is to be created in a work file database, you cannot specify 8KB and 16KB buffer pools.

If you do not specify the BUFFERPOOL clause, the default buffer pool of the database is used.

See "Naming conventions" on page 34 for more details about *bpname*. See Chapter 2 of *DB2 Command Reference* for a description of active and inactive buffer pools.

### **LOCKSIZE**

Specifies the size of locks used within the table space and, in some cases, also the threshold at which lock escalation occurs. Do not use this clause for a table space in a work file database or a TEMP database.

### **ANY**

Specifies that DB2 can use any lock size. Currently, DB2 never chooses row locks, but reserves the right to do so.

In most cases, DB2 uses LOCKSIZE PAGE LOCKMAX SYSTEM for non-LOB table spaces and LOCKSIZE LOB LOCKMAX SYSTEM for LOB table spaces. However, when the number of locks acquired for the table space exceeds the maximum number of locks allowed for a table space (an

installation parameter), the page or LOB locks are released and locking is set at the next higher level. If the table space is segmented, the next higher level is the table. If the table space is nonsegmented, the next higher level is the table space.

#### **TABLESPACE**

Specifies table space locks.

#### **TABLE**

Specifies table locks. Use TABLE only for a segmented table space.

#### **PAGE**

Specifies page locks. Do not use PAGE for a LOB table space.

#### **ROW**

Specifies row locks. Do not use ROW for a LOB table space.

#### **LOB**

Specifies LOB locks. Use LOB only for a LOB table space.

#### **LOCKMAX**

Specifies the maximum number of page, row, or LOB locks an application process can hold simultaneously in the table space. If a program requests more than that number, locks are escalated. The page, row, or LOB locks are released and the intent lock on the table space or segmented table is promoted to S or X mode. If you specify LOCKMAX for table space in a TEMP database, DB2 ignores the value because these types of locks are not used.

#### *integer*

Specifies the number of locks allowed before escalating, in the range 0 to 2 147 483 647.

Zero (0) indicates that the number of locks on the table or table space are not counted and escalation does not occur.

#### **SYSTEM**

Indicates that the value of LOCKS PER TABLE(SPACE), on installation panel DSNTIPJ, specifies the maximum number of page, row, or LOB locks a program can hold simultaneously in the table or table space.

The following table summarizes the results of specifying a LOCKSIZE value while omitting LOCKMAX.

| LOCKSIZE                                | Resultant LOCKMAX |
|-----------------------------------------|-------------------|
| ANY                                     | SYSTEM            |
| TABLESPACE, TABLE, PAGE,<br>ROW, or LOB | 0                 |

If the lock size is TABLESPACE or TABLE, LOCKMAX must be omitted, or its operand must be 0.

For an application that uses Sysplex query parallelism, a lock count is maintained on each member.

#### **CLOSE**

When the limit on the number of open data sets is reached, specifies the priority in which data sets are closed.

## CREATE TABLESPACE

### YES

Eligible for closing before CLOSE NO data sets. This is the default unless the table space is in a TEMP database.

### NO

Eligible for closing after all eligible CLOSE YES data sets are closed.

For a table space in a TEMP database, DB2 uses CLOSE NO regardless of the value specified.

### COMPRESS

Specifies whether data compression applies to the rows of the table space or partition. Do not specify COMPRESS for a LOB table space or a table space in a TEMP database.

For partitioned table spaces, the COMPRESS attribute for each partition is the value from the first of the following conditions that apply:

- The value specified in the COMPRESS clause in the PART clause for the partition
- The value specified in the COMPRESS clause that is not in any PART clause
- An implicit COMPRESS NO by default.

See Part 5 (Volume 2) of *DB2 Administration Guide* for more information about data compression.

### YES

Specifies data compression. The rows are not compressed until the LOAD or REORG utility is run on the table in the table space or partition.

### NO

Specifies no data compression for the table space or partition.

### SEGSIZE *integer*

Indicates that the table space will be segmented. *integer* specifies how many pages are to be assigned to each segment. *integer* must be a multiple of 4 such that  $4 \leq \text{integer} \leq 64$ . If the SEGSIZE clause is not specified, the table space is not segmented.

SEGSIZE must be specified for a table space in a TEMP database because the table space must be segmented. Do not specify SEGSIZE for a LOB table space or a table space in work file database; neither can be segmented.

A segmented table space cannot be partitioned. Therefore, do not specify NUMPARTS if you specify SEGSIZE.

### CCSID *encoding-scheme*

Specifies the encoding scheme for tables stored in the table space.

If you do not specify a CCSID when it is allowed, the default is the encoding scheme of the database in which the table space resides, except for table spaces in database DSNDB04; for table spaces in DSNDB04, the default is the value of field DEF ENCODING SCHEME on installation panel DSNTIPF.

**ASCII** Specifies that the data is to be encoded using ASCII CCSIDs. If the database in which the table space is to reside is already defined as ASCII, the ASCII CCSIDs associated with that database are used. Otherwise, the default ASCII CCSIDs of the server are used.

### EBCDIC

Specifies that the data is to be encoded using EBCDIC CCSIDs. If the database in which the table space is to reside is already defined as

EBCDIC, the EBCDIC CCSIDs associated with that database are used. Otherwise, the default EBCDIC CCSIDs of the server are used.

#### **UNICODE**

Specifies that the data is to be encoded using UNICODE CCSIDs. If the database in which the table space is to reside is already defined as Unicode, the UNICODE CCSIDs associated with that database are used. Otherwise, the default UNICODE CCSIDs of the server are used.

Usually, each encoding scheme requires only a single CCSID. Additional CCSIDs are needed when mixed, graphic, or Unicode data is used.

All data stored within a table space must use the same encoding scheme unless the table space is in a TEMP database.

Do not specify CCSID for a LOB table space or a table space in a TEMP database. The encoding scheme for a LOB table space is inherited from the base table space. A table space in a TEMP database does not have an associated encoding scheme because the table space can contain declared temporary tables with a mixture of encoding schemes.

#### **LOCKPART**

Indicates whether selective partition locking (SPL) is to be used when locking a partitioned table space.

**YES** If all the conditions that are required for SPL are met, specifies that only the partitions accessed will be locked. If all the conditions that are required for SPL are not met, every partition of the table space is locked. LOCKPART YES is not allowed with LOCKSIZE TABLESPACE.

**NO** Specifies that selective partition locking is not used. The table space is locked with a single lock on the last partition. This has the effect of locking all partitions in the table space.

#### **MAXROWS *integer***

Specifies the maximum number of rows that DB2 will consider placing on each data page. The integer can range from 1 through 255. This value is considered for INSERT, LOAD, and REORG. For LOAD and REORG (which do not apply for a table space in the TEMP database), the PCTFREE specification is considered before MAXROWS; therefore, fewer rows might be stored than the value you specify for MAXROWS.

If you do not specify MAXROWS, the default number of rows is 255.

Do not use MAXROWS for a LOB table space or a table space in a work file database.

## **Notes**

**Simple table spaces:** If neither LOB, NUMPARTS, nor SEGSIZE are specified, the table space that is created is a simple table space. See *An Introduction to DB2 for OS/390* for a discussion of types of table spaces.

**Table spaces in a work file database:** The following restrictions apply to table spaces created in a work file database:

- They can be created for another member only if both the executing DB2 subsystem and the other member can access the work file data sets. That is required whether the data sets are user-managed or in a DB2 storage group.
- They cannot use 8-KB or 16-KB page sizes.(The buffer pool in which you define the table space determines the page size. For example, a table space that is defined in a 4-KB buffer pool has 4-KB page sizes.)

#  
#  
#

## CREATE TABLESPACE

- The following clauses are not allowed:

|           |          |          |
|-----------|----------|----------|
| DEFINE NO | LOB      | NUMPARTS |
| FREEPAGE  | LOG      | PCTFREE  |
| GBPCACHE  | LOCKSIZE | SEGSIZE  |

### ***Table spaces in a TEMP database (table spaces for declared temporary tables):***

Declared temporary tables must reside in segmented table spaces in a database that is defined AS TEMP (the TEMP database). At least one segmented table space must exist in the TEMP database before a declared temporary table can be defined and used. DB2 does not implicitly create a table space for declared temporary tables. A table space for declared temporary tables can be shared. You cannot choose which table space in a TEMP database is used for a specific declared temporary table. Therefore, multiple application processes can use the same table space for their declared temporary tables.

When you create table spaces for in the TEMP database, it is recommended that you give them all the same segment size, with the same minimum primary and secondary space allocation values for the data sets, to maximize the use of all the table spaces for all declared temporary tables in all application processes.

When you create a table space in a TEMP database, the following clauses are not allowed:

|           |          |                |
|-----------|----------|----------------|
| CCSID     | GBPCACHE | LOG            |
| COMPRESS  | LARGE    | MEMBER CLUSTER |
| DEFINE NO | LOB      | NUMPARTS       |
| DSSIZE    | LOCKSIZE | PCTFREE        |
| FREEPAGE  | LOCKPART | TRACKMOD       |

***Table spaces in a TEMP database (table spaces for scrollable cursors):***For information on creating table spaces in a TEMP database for scrollable cursors, see *DB2 Installation Guide*.

***Creating LOB table spaces:*** When you create a LOB table space, the following clauses are not allowed:

|                |               |          |
|----------------|---------------|----------|
| CCSID          | LOCKSIZE PAGE | PCTFREE  |
| COMPRESS       | LOCKSIZE ROW  | SEGSIZE  |
| FREEPAGE       | MAXROWS       | TRACKMOD |
| LOCKSIZE TABLE | NUMPARTS      |          |

***Converting a partitioned table space to be larger:*** To increase the size of a partitioned table space so that it can hold more data, take the following steps:

1. Unload the data rows from the table space, if necessary.
2. Drop the table space. The table and any indexes, views, or synonyms dependent on the table are dropped, and authorizations for the table and views are revoked.
3. Create the table space, specifying an appropriate value for the DSSIZE clause. Also redefine the partitioning index (with different key range values), the table, and the nonclustering indexes.
4. Recreate views and synonyms. Reestablish appropriate authorizations.
5. Load data into the new table.

6. Rebind the plans and packages that changed.

## Examples

*Example 1:* Create table space DSN8S71D in database DSN8D71A. Let DB2 define the data sets, using storage group DSN8G710. The primary space allocation is 52 kilobytes; the secondary, 20 kilobytes. The data sets need not be erased before they are deleted.

Locking on tables in the space is to take place at the page level. Associate the table space with buffer pool BP1. The data sets can be closed when no one is using the table space.

```
CREATE TABLESPACE DSN8S71D
 IN DSN8D71A
 USING STOGROUP DSN8G710
 PRIQTY 52
 SECQTY 20
 ERASE NO
 LOCKSIZE PAGE
 BUFFERPOOL BP1
 CLOSE YES;
```

For the above example, the underlying data sets for the table space will be created immediately, which is the default (DEFINE YES). If you want to defer the creation of the data sets until data is first inserted into the table space, you would specify DEFINE NO instead of accepting the default behavior.

*Example 2:* Assume that a large query database application uses a table space to record historical sales data for marketing statistics. Create large table space SALESHX in database DSN8D71A for the application. Create it with 82 partitions, specifying that the data in partitions 80 through 82 is to be compressed.

Let DB2 define the data sets for all the partitions in the table space, using storage group DSN8G710. For each data set, the primary space allocation is 4000 kilobytes, and the secondary space allocation is 130 kilobytes. Except for the data set for partition 82, the data sets do not need to be erased before they are deleted.

Locking on the table is to take place at the page level. There can only be one table in a partitioned table space. Associate the table space with buffer pool BP1. The data sets cannot be closed when no one is using the table space. If there are no CLOSE YES data sets to close, DB2 may close the CLOSE NO data sets when the DSMAX is reached.

```
CREATE TABLESPACE SALESHX
 IN DSN8D71A
 USING STOGROUP DSN8G710
 PRIQTY 4000
 SECQTY 130
 ERASE NO
 NUMPARTS 82
 (PART 80
 COMPRESS YES,
 PART 81
 COMPRESS YES,
 PART 82
 COMPRESS YES)
 LOCKSIZE PAGE
 BUFFERPOOL BP1
 CLOSE NO;
```

## CREATE TABLESPACE

*Example 3:* Assume that a column named EMP\_PHOTO with a data type of BLOB(110K) has been added to the sample employee table for each employee's photo. Create LOB table space PHOTOLTS in database DSN8D71A for the auxiliary table that will hold the BLOB data.

Let DB2 define the data sets for the table space, using storage group DSN8G710. For each data set, the primary space allocation is 3200 kilobytes, and the secondary space allocation is 1600 kilobytes. The data sets do not need to be erased before they are deleted.

```
CREATE LOB TABLESPACE PHOTOLTS
 IN DSN8D71A
 USING ST0GROUP DSN8G710
 PRIQTY 3200
 SECQTY 1600
 LOCKSIZE LOB
 BUFFERPOOL BP16K0
 GBPCACHE SYSTEM
 LOG NO
 CLOSE NO;
```

---

## CREATE TRIGGER

The CREATE TRIGGER statement defines a trigger in a schema and builds a trigger package at the current server.

### Invocation

This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is implicitly or explicitly specified.

### Authorization

The privilege set that is defined below must include all of the following:

- Either of these privileges:
  - The CREATEIN privilege for the schema or all schemas
  - SYSADM or SYSCTRL authority
- The TRIGGER privilege on the table. The privilege set must include at least one of the following:
  - The TRIGGER privilege on the table on which the trigger is defined
  - The ALTER privilege on the table on which the trigger is defined
  - DBADM authority on the database that contains the table
  - SYSADM or SYSCTRL authority
- The SELECT privilege on the table on which the trigger is defined if any transition variables or transition tables are specified
- The SELECT privilege on any table or view to which the search condition of triggered action refers
- The EXECUTE privilege on any user-defined function or stored procedure that is invoked in the triggered action
- The necessary privileges to invoke the triggered SQL statements in the triggered action

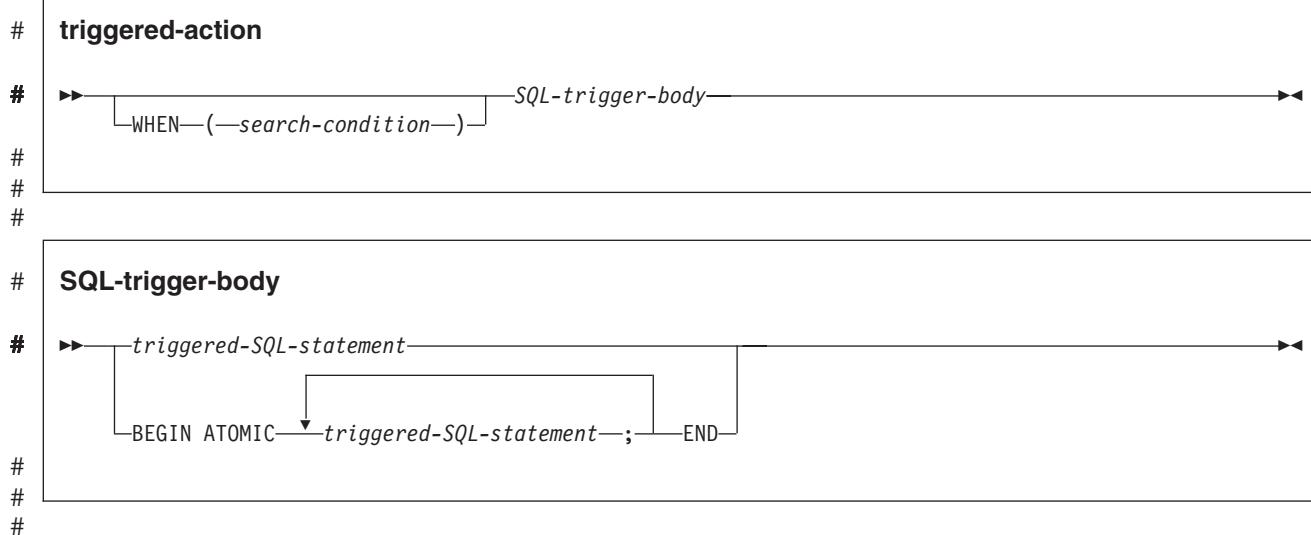
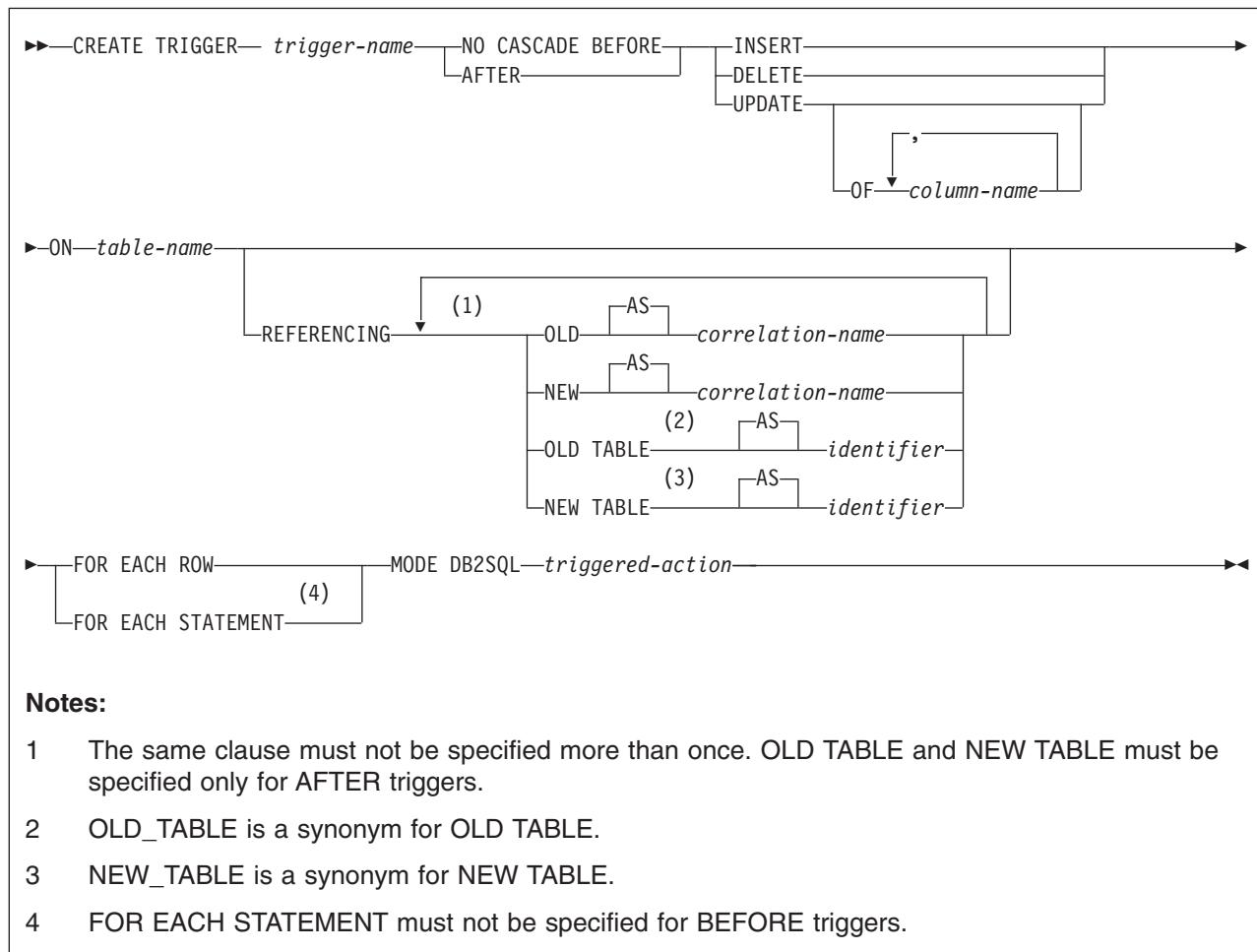
**Privilege set:** If the statement is embedded in an application program, the privilege set is the privileges that are held by the authorization ID of the owner of the plan or package.

If the statement is dynamically prepared, the privilege set is the privileges that are held by the SQL authorization ID of the process. The specified trigger name can include a schema name (a qualifier). However, if the specified name includes a schema name that is not the same as the SQL authorization ID, one of the following conditions must be met:

- The privilege set includes SYSADM or SYSCTRL authority.
- The SQL authorization ID of the process has the CREATEIN privilege on the schema.

## CREATE TRIGGER

### Syntax



## Description

### *trigger-name*

Names the trigger. The name is implicitly or explicitly qualified by a schema. The name, including the implicit or explicit schema name, must not identify a trigger that exists at the current server.

The name is also used to create the trigger package; therefore, the name must also not identify a package that is already described in the catalog. The schema name becomes the collection-id of the trigger package.

- The unqualified form of *trigger-name* is a short SQL identifier. The unqualified name is implicitly qualified with a schema name according to the following rules:

If the statement is embedded in a program, the schema name of the trigger is the authorization ID in the QUALIFIER bind option when the plan or package was created or last rebound. If QUALIFIER was not used, the schema name of the trigger is the owner of the package or plan.

If the statement is dynamically prepared, the schema name of the trigger is the SQL authorization ID of the process.

- The qualified form of *trigger-name* is a short SQL identifier (the schema name) followed by a period and a short SQL identifier. The schema name must not begin with 'SYS' unless the name is 'SYSADM'. The schema name that qualifies the trigger name is the trigger's owner.

The owner of the trigger is determined by how the CREATE TRIGGER statement is invoked:

- If the statement is embedded in a program, the owner is the authorization ID of the owner of the plan or package.
- If the statement is dynamically prepared, the owner is the SQL authorization ID in the CURRENT SQLID special register.

### NO CASCADE BEFORE

Specifies that the trigger is a before trigger. DB2 executes the triggered action before it applies any changes caused by an insert, delete, or update operation on the subject table. It also specifies that the triggered action does not activate other triggers because the triggered action of a before trigger cannot contain any updates.

### AFTER

Specifies that the trigger is an after trigger. DB2 executes the triggered action after it applies any changes caused by an insert, delete, or update operation on the subject table.

### INSERT

Specifies that the trigger is an insert trigger. DB2 executes the triggered action whenever there is an insert operation on the subject table. However, if the insert trigger is defined on PLAN\_TABLE, DSN\_STATEMNT\_TABLE, or DSN\_FUNCTION\_TABLE, and the insert operation was caused by DB2 adding a row to the table, the triggered action is not executed.

### DELETE

Specifies that the trigger is a delete trigger. DB2 executes the triggered action whenever there is a delete operation on the subject table.

### UPDATE

Specifies that the trigger is an update trigger. DB2 executes the triggered action whenever there is an update operation on the subject table.

## CREATE TRIGGER

If you do not specify a list of column names, an update operation on any column of the subject table, including columns that are subsequently added with the ALTER TABLE statement, activates the triggered action.

### **OF** *column-name*,...

Each *column-name* that you specify must be a column of the subject table and must appear in the list only once. An update operation on any of the listed columns activates the triggered action.

### **ON** *table-name*

Identifies the subject table with which the trigger is associated. The name must identify a base table at the current server. It must not identify a temporary table, an auxiliary table, an alias, a synonym, or a catalog table.

### **REFERENCING**

Specifies the correlation names for the transition variables and the table names for the transition tables. For the rows in the subject table that are modified by the triggering SQL operation (insert, delete, or update), a correlation name identifies the columns of a specific row. A table name identifies the complete set of modified rows.

Each row that is modified by the triggering operation is available to the triggered action by using column names that are qualified with correlation names that are specified as follows:

#### **OLD AS** *correlation-name*

Specifies the correlation name that identifies the state of the row prior to the triggering SQL operation.

#### **NEW AS** *correlation-name*

Specifies the correlation name that identifies the state of the row as modified by the triggering SQL operation and by any SET statement in a before trigger that has already been executed.

The complete set of rows that is modified by the triggering operation is available to the triggered action by using a temporary table name that is specified as follows:

#### **OLD TABLE AS** *identifier*

Specifies the name of a temporary table that identifies the state of the complete set of rows that are modified rows by the triggering SQL operation prior to any actual changes. *identifier* is a long SQL identifier.

#### **NEW TABLE AS** *identifier*

Specifies the name of a temporary table that identifies the state of the complete set of rows as modified by the triggering SQL operation and by any SET statement in a before trigger that has already been executed. *identifier* is a long SQL identifier.

At most, the trigger definition can include two correlation names (OLD and NEW) and two table names (OLD TABLE and NEW TABLE). All the names must be unique from one another.

Table 49 on page 703 summarizes the allowable combinations of transition variables and transition tables that you can specify for the various trigger types. OLD and OLD TABLE are valid only if the triggering SQL operation is a delete or an update. For a delete operation, OLD captures the values of the columns in the deleted row, and OLD TABLE captures the values in the set of deleted

rows. For an update operation, OLD captures the values of the columns of a row before the update, and OLD TABLE captures the values in the set of updated rows.

NEW and NEW TABLE are valid only if the triggering SQL operation is an insert or an update. For both operations, NEW captures the values of the columns in the inserted or updated row. For before triggers, the values of the updated rows include the changes from any SET statement in the triggered action if the trigger is a before trigger.

OLD and NEW are valid only if you also specify FOR EACH ROW, and OLD TABLE and NEW TABLE are valid only if you specify AFTER.

*Table 49. Allowable combinations of attributes in a trigger definition*

| Activation time | Triggering SQL operation | Transition variables | Transition tables       | Granularity        |
|-----------------|--------------------------|----------------------|-------------------------|--------------------|
| BEFORE          | DELETE                   | OLD                  |                         |                    |
|                 | INSERT                   | NEW                  |                         | FOR EACH ROW       |
|                 | UPDATE                   | OLD, NEW             |                         |                    |
|                 | DELETE                   | OLD                  | OLD TABLE               |                    |
|                 | INSERT                   | NEW                  | NEW TABLE               | FOR EACH ROW       |
|                 | UPDATE                   | OLD, NEW             | OLD TABLE,<br>NEW TABLE |                    |
| AFTER           | DELETE                   |                      | OLD TABLE               |                    |
|                 | INSERT                   |                      | NEW TABLE               | FOR EACH STATEMENT |
|                 | UPDATE                   |                      | OLD TABLE,<br>NEW TABLE |                    |

A transition variable that has a character data type inherits the subtype and CCSID of the column of the subject table. During the execution of the triggered action, the transition variables are treated like host variables. Therefore, character conversion might occur. However, unlike a host variable, a transition variable can have the bit data attribute, and character conversion never occurs for bit data. A transition variable is considered to be bit data if the column of the table to which it corresponds is bit data.

#  
#  
#  
#

You cannot modify a transition table; transition tables are read-only. Although a transition table does not inherit any edit or validation procedures from the subject table, it does inherit the subject table's encoding scheme and field procedures.

The scope of the transition variables and transition tables is the triggered action. Do not refer to their names outside of the triggered action.

#### FOR EACH ROW

Specifies that DB2 executes the triggered action for each row of the subject table that the triggering SQL operation modifies. If the triggering SQL operation does not modify any rows, the triggered action is not executed.

#### FOR EACH STATEMENT

Specifies that DB2 executes the triggered action only once for the triggering SQL operation. Even if the triggering SQL operation does not modify any rows, the triggered action is executed once. Do not specify FOR EACH STATEMENT for a before trigger.

## CREATE TRIGGER

### MODE DB2SQL

Specifies the mode of the trigger. Currently, DB2 supports only MODE DB2SQL.

### triggered-action

Specifies the action to be performed when the trigger is activated. The triggered action is composed of one or more SQL statements and by an optional condition that controls whether the statements are executed.

### WHEN (*search-condition*)

Specifies a condition that evaluates to true, false, or unknown. The condition for a before trigger must not include a subselect that references the subject table.

The triggered SQL statements are executed only if the search condition evaluates to true, or if WHEN is omitted.

# *triggered-SQL-statement or BEGIN ATOMIC triggered-SQL-statement,... END*

Specifies the SQL statement or SQL statements that are to be executed for the triggered action. If there is more than one statement, they are executed in the order in which you specify them. The keywords BEGIN ATOMIC and END are required only if you specify more than one SQL statement. In which case, you must enclose the SQL statements in these keywords and end each statement with a semicolon (;).

SQL processor programs, such as SPUFI and DSNTEP2, might not correctly parse SQL statements in the triggered action that are ended with semicolons. These processor programs accept multiple SQL statements, each separated with a terminator character, as input. Processor programs that use a semicolon as the SQL statement terminator can truncate a CREATE TRIGGER statement with embedded semicolons and pass only a portion of it to DB2. Therefore, you might need to change the SQL terminator character for these processor programs. For information on changing the terminator character for SPUFI and DSNTEP2, see *DB2 Application Programming and SQL Guide*.

Table 50 shows the list of allowable SQL statements, which differs depending on whether the trigger is being defined as BEFORE or AFTER. An 'X' in the table indicates that the statement is valid.

Table 50. Allowable SQL statements

| SQL statement           | Trigger activation time |       |
|-------------------------|-------------------------|-------|
|                         | BEFORE                  | AFTER |
| fullselect              | X                       | X     |
| CALL                    | X                       | X     |
| SIGNAL SQLSTATE         | X                       | X     |
| VALUES                  | X                       | X     |
| SET transition variable | X                       |       |
| INSERT                  |                         | X     |
| DELETE (searched)       |                         | X     |
| UPDATE (searched)       |                         | X     |

The statements in the triggered action have these restrictions:

- They must not refer to host variables, parameter markers, undefined transition variables, or declared temporary tables.
- They must only refer to a table or view that is at the current server.

- They must only invoke a stored procedure or user-defined function that is at the current server. An invoked routine can, however, access a server other than the current server.
- They must not contain a fullselect that refers to the subject table if the trigger is defined as BEFORE.

The triggered action may refer to the values in the set of affected rows. This action is supported through the use of transition variables and transition tables.

Transition variables use the names of the columns in the subject table qualified by a specified name that identifies whether the reference is to the old value (before the update) or the new value (after the update). A transition variable can be referenced in *search-condition* or *triggered-SQL-statement* of the triggered action wherever a host variable is allowed in the statement if it were issued outside the body of a trigger.

Transition tables can be referenced in the triggered action of an after trigger. Transition tables are read-only. Transition tables also use the name of the columns of the subject table but have a name specified that allows the complete set of affected rows to be treated as a table. The name of the transition table can be referenced in *triggered-SQL-statement* of the triggered action whenever a table name is allowed in the statement if it were issued outside the body of a trigger. The name of the transition table can be specified in *search-condition* or *triggered-SQL-statement* of the triggered action whenever a column name is allowed in the statement if it were issued outside the body of a trigger.

In addition, a transition table can be passed as a parameter to a user-defined function or procedure specifying the TABLE keyword before the name of the transition table. When the function or procedure is invoked, a table locator is passed for the transition table.

A transition variable or transition table is not affected after being returned from a procedure invoked from within a triggered action regardless of whether the corresponding parameter was defined in the CREATE PROCEDURE statement as IN, INOUT, or OUT.

## Notes

**The implicitly created trigger package:** When you create a trigger, DB2 automatically creates a trigger package with the same name as the trigger name. The collection name of the trigger package is the schema name of the trigger, and the version identifier is the empty string. Multiple versions of a trigger package are not allowed.

The user executing the triggering SQL operation does not need authority to execute a trigger package. The trigger package does not need to be in the package list for the plan that is associated with the program that contains the SQL statement.

A trigger package becomes invalid if an object or privilege on which it depends is dropped or revoked. The next time the trigger is activated, DB2 attempts to rebind the invalid trigger package. If the automatic rebind is unsuccessful, the trigger package remains invalid.

You cannot create another package from the trigger package, such as with the BIND COPY command. The only way to drop a trigger package is to drop the

## CREATE TRIGGER

trigger or the subject table. Dropping the trigger drops the trigger package; dropping the subject table drops the trigger and the trigger package.

DB2 creates the trigger package with the following attributes:

- ACTION(ADD)
- CURRENTDATA(YES)
- DBPROTOCOL(DRDA)
- DEGREE(1)
- DYNAMICRULES(BIND)
- ENABLE(\*)
- ENCODING(0)
- EXPLAIN(NO)
- FLAG(I)
- ISOLATION(CS)
- NOREOPT(VARS) and NODEFER(PREPARE)
- OWNER(authorization ID)
- QUERYOPT(1)
- PATH(path)
- RELEASE(COMMIT)
- SQLERROR(NOPACKAGE)
- QUALIFIER(authorization ID)
- VALIDATE(BIND)

The values of OWNER, QUALIFIER, and PATH are set depending on whether the CREATE TRIGGER statement is embedded in a program or issued interactively. If the statement is embedded in a program, OWNER and QUALIFIER are the owner and qualifier of the package or plan. PATH is the value from the PATH bind option. If the statement is issued interactively, both OWNER and QUALIFIER are the SQL authorization ID. PATH is the value in the CURRENT PATH special register.

**Activating a trigger:** Only the SQL statements INSERT, DELETE, or UPDATE, or an update or delete operation that occurs as the result of a referential constraint with ON DELETE SET NULL or ON DELETE CASCADE can activate a trigger. Loading a table with the LOAD utility does not activate any triggers that are defined for the table.

**Simultaneously activated triggers:** Multiple triggers that have the same triggering SQL operation and activation time (BEFORE or AFTER) can be defined on a table. The triggers are activated in the order in which they were created. For example, the trigger that was created first is executed first; the trigger that was created second is executed second; and so on.

**Adding columns to subject tables or tables that the triggered action references:** If a column is added to a table for which a trigger is defined (the subject table), the following rules apply:

- If the trigger is an update trigger that was defined without an explicit list of column names, an update to the new column activates the trigger.
- If the SQL statements in the triggered action refer to the subject table, the new column is not accessible to the SQL statements until the trigger package is rebound.
- The transition tables contain the new column. If the transition tables are passed to a user-defined function or a stored procedure, the user-defined function or stored procedure, the user-defined function or stored procedure must be

recreated with the new definition of the table (that is, the function or procedure must be dropped and recreated), and the package for the user-defined function or stored procedure must be rebound.

If a column is added to any table to which the SQL statements in the triggered action refers, the new column is not accessible to the SQL statements until the trigger package is rebound.

**Adding triggers to enforce constraints:** Creating a trigger on a table that already has rows does not cause the triggered action to be executed. Thus, if the trigger is designed to enforce constraints on the data in the table, the data in the existing rows might not satisfy those constraints.

**Defining triggers on plan, statement, and function tables:** You can create a trigger on PLAN\_TABLE, DSN\_STATEMENT\_TABLE, or DSN\_FUNCTION\_TABLE. However, insert triggers that are defined on these tables are not activated when DB2 adds rows to the tables.

**Renaming subject tables or tables that the triggered action references:** You cannot rename a table for which a trigger is defined (the subject table). Except for the subject table, you can rename any table to which the SQL statements in the triggered action refer. After renaming such a table, drop the trigger and then re-create the trigger so that it refers to the renamed table.

**Dependencies when dropping objects and revoking privileges:** The following dependencies apply to a trigger:

- Dropping the subject table (the table on which the trigger is defined) causes the trigger and its package to also be dropped.
- Dropping any table, view, alias, or index that is referenced or used within the SQL statements in the triggered action causes the trigger and its package to be invalidated. Dropping a referenced synonym has no effect.
- Dropping a user-defined function that is referenced by the SQL statements in the triggered action is not allowed. An error occurs.
- Revoking a privilege on which the trigger depends causes the trigger and its package to be invalidated.

**Result sets for stored procedures:** If a trigger invokes a stored procedure that returns result sets, the application that activated the trigger cannot access those result sets.

**Values of special registers:** The values of the special registers are saved before a trigger is activated and are restored on return from the trigger.

Table 51 on page 708 gives the rules for special registers within a trigger. Some of the special registers are applicable only to dynamic SQL. Although dynamic SQL statements are not allowed directly in the triggered SQL statements, they are allowed in a user-defined function or stored procedure that is invoked by the triggered SQL statements.

## CREATE TRIGGER

Table 51. Rules for the values of special registers in triggers

| Special register          | The value is ....                                                                                                                                                                                                               |
|---------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| CURRENT DATE              | Inherited from the triggering SQL operation (delete, insert, update). All triggered SQL statements, including the SQL statements in a user-defined function or a stored procedure invoked by the trigger, inherit these values. |
| CURRENT TIME              |                                                                                                                                                                                                                                 |
| CURRENT TIMESTAMP         |                                                                                                                                                                                                                                 |
| CURRENT PACKAGESET        | Set to the schema name of the trigger                                                                                                                                                                                           |
| CURRENT TIMEZONE          | Set to the MVS TIMEZONE parameter                                                                                                                                                                                               |
| CURRENT DEGREE            | Inherited from the triggering SQL operation (delete, insert, update)                                                                                                                                                            |
| CURRENT LC_CTYPE          |                                                                                                                                                                                                                                 |
| CURRENT OPTIMIZATION_HINT |                                                                                                                                                                                                                                 |
| CURRENT PATH              |                                                                                                                                                                                                                                 |
| CURRENT PRECISION         |                                                                                                                                                                                                                                 |
| CURRENT RULES             |                                                                                                                                                                                                                                 |
| CURRENT SERVER            |                                                                                                                                                                                                                                 |
| CURRENT SQLID             |                                                                                                                                                                                                                                 |
| USER                      |                                                                                                                                                                                                                                 |

**Errors when binding triggers:** When a CREATE TRIGGER statement is bound, the SQL statements within the triggered action may not be fully parsed. Syntax errors in those statements might not be caught until the CREATE TRIGGER statement is executed.

**Errors when executing triggers:** Severe errors that occur during the execution of triggered SQL statements are returned with SQLCODE -901, -906, -911, and -913 and the corresponding SQLSTATE. Non-severe errors raised by a triggered SQL statement that is a SIGNAL SQLSTATE statement or that contains a RAISE\_ERROR function are returned with SQLCODE -438 and the SQLSTATE that is specified in the SIGNAL SQLSTATE statement or the RAISE\_ERROR condition. Other non-severe errors are returned with SQLCODE -723 and SQLSTATE 09000.

Warnings are not returned.

**Limiting processor time:** DB2's resource limit facility allows you to specify the maximum amount of processor time for a dynamic, manipulative SQL statement (SELECT, INSERT, UPDATE, and DELETE). The execution of a trigger is counted as part of the triggering SQL statement.

## Examples

*Example 1:* Create two triggers that track the number of employees that a company manages. The subject table is the EMPLOYEE table, and the triggers increment and decrement a column with the total number of employees in the COMPANY\_STATS table. The tables have these columns:

EMPLOYEE table: ID, NAME, ADDRESS, and POSITION

COMPANY\_STATS table: NBEMP, NBPRODUCT, and REVENUE

This example shows the use of transition variables in a row trigger to maintain summary data in another table.

Create the first trigger, NEW\_HIRE, so that it increments the number of employees each time a new person is hired; that is, each time a new row is inserted into the EMPLOYEE table, increase the value of column NBEMP in table COMPANY\_STATS by 1.

```

CREATE TRIGGER NEW_HIRE
 AFTER INSERT ON EMPLOYEE
 FOR EACH ROW MODE DB2SQL
 BEGIN ATOMIC
 UPDATE COMPANY_STATS SET NBEMP = NBEMP + 1;
 END

```

Create the second trigger, FORM\_EMP, so that it decrements the number of employees each time an employee leaves the company; that is, each time a row is deleted from the table EMPLOYEE, decrease the value of column NBEMP in table COMPANY\_STATS by 1.

```

CREATE TRIGGER FORM_EMP
 AFTER DELETE ON EMPLOYEE
 FOR EACH ROW MODE DB2SQL
 BEGIN ATOMIC
 UPDATE COMPANY_STATS SET NBEMP = NBEMP - 1;
 END

```

*Example 2:* Create a trigger, REORDER, that invokes user-defined function ISSUE\_SHIP\_REQUEST to issue a shipping request whenever a parts record is updated and the on-hand quantity for the affected part is less than 10% of its maximum stocked quantity. User-defined function ISSUE\_SHIP\_REQUEST orders a quantity of the part that is equal to the part's maximum stocked quantity minus its on-hand quantity; the function also ensures that the request is sent to the appropriate supplier.

#  
#  
#  
The parts records are in the PARTS table. Although the table has more columns, the trigger is activated only when columns ON\_HAND and MAX\_STOCKED are updated.

```

CREATE TRIGGER REORDER
 AFTER UPDATE OF ON_HAND, MAX_STOCKED ON PARTS
 REFERENCING NEW AS NROW
 FOR EACH ROW MODE DB2SQL
 WHEN (NROW.ON_HAND < 0.10 * NROW.MAX_STOCKED)
 BEGIN ATOMIC
 VALUES(ISSUE_SHIP_REQUEST(NROW.MAX_STOCKED - NROW.ON_HAND, NROW.PARTNO));
 END

```

*Example 3:* Repeat the scenario in *Example 2* except use a fullselect instead of a VALUES statement to invoke the user-defined function. This example also shows how to define the trigger as a statement trigger instead of a row trigger. For each row in the transition table that evaluates to true for the WHERE clause, a shipping request is issued for the part.

```

CREATE TRIGGER REORDER
 AFTER UPDATE OF ON_HAND, MAX_STOCKED ON PARTS
 REFERENCING NEW TABLE AS NTABLE
 FOR EACH STATEMENT MODE DB2SQL
 BEGIN ATOMIC
 SELECT ISSUE_SHIP_REQUEST(MAX_STOCKED - ON_HAND, PARTNO)
 FROM NTABLE
 WHERE (ON_HAND < 0.10 * MAX_STOCKED);
 END

```

*Example 4:* Assume that table EMPLOYEE contains column SALARY. Create a trigger, SAL\_ADJ, that prevents an update to an employee's salary that exceeds 20% and signals such an error. Have the error that is returned with an SQLSTATE of '75001' and a description. This example shows that the SIGNAL SQLSTATE statement is useful for restricting changes that violate business rules.

## CREATE TRIGGER

```
CREATE TRIGGER SAL_ADJ
 AFTER UPDATE OF SALARY ON EMPLOYEE
 REFERENCING OLD AS OLD_EMP
 NEW AS NEW_EMP
 FOR EACH ROW MODE DB2SQL
 WHEN (NEW_EMP.SALARY > (OLD_EMP.SALARY * 1.20))
 BEGIN ATOMIC
 SIGNAL SQLSTATE '75001' ('Invalid Salary Increase - Exceeds 20%');
 END
```

## CREATE VIEW

The CREATE VIEW statement creates a view on tables or views at the current server.

### Invocation

This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is implicitly or explicitly specified.

### Authorization

For every table or view identified in the *fullselect*, the privilege set that is defined below must include at least one of the following:

- The SELECT privilege on the table or view
- Ownership of the table or view
- DBADM authority for the database (tables only)
- SYSADM authority
- SYSCTRL authority (catalog tables only)

Additional authorization is required if the definition of the view references any user-defined functions or cast functions that were generated for a distinct type. The privilege set defined below must include the EXECUTE privilege on the referenced functions.

Authority requirements depend in part on the choice of the view's owner. For information on how to choose the owner, see the description of *view-name* in "Description" on page 712.

**Privilege set:** If the statement is embedded in an application program, the privilege set is the privileges that are held by the authorization ID of the owner of the plan or package:

- If this privilege set includes SYSADM authority, the owner of the view can be any authorization ID. If that set includes SYSCTRL but not SYSADM authority, the following is true: the owner of the view can be any authorization ID, provided the view does not refer to user tables or views in the first FROM clause of its defining fullselect. (It could refer instead, for example, to catalog tables or views thereof.)

If the view satisfies the rules in the preceding paragraph, and if no errors are present in the CREATE statement, the view is created, even if the owner has no privileges at all on the tables and views identified in the view's fullselect.

- If the privilege set includes DBADM authority on at least one of the databases that contains a table from which the view is created, the owner of the view can be any authorization ID if all of the following conditions are true:
  - The value of field DBADM CREATE AUTH was set to YES on panel DSNTIPP during DB2 installation.
  - The view is not based only on views.

**Note:** The owner of the view must have the SELECT privilege on all tables and views in the CREATE VIEW statement, or, if the owner does not have the SELECT privilege on a table, the creator must have DBADM authority on the database that contains that table.

## CREATE VIEW

- If the privilege set lacks SYSADM, SYSCTRL, or DBADM authority, or if the authorization ID of the application plan or package fails to meet any of the previous conditions, the owner of the view must be the owner of the application plan or package.

If the statement is dynamically prepared, the following rules apply:

- If the SQL authorization ID of the process has SYSADM authority, the owner of the view can be any authorization ID. If that authorization ID has SYSCTRL but not SYSADM authority, the following is true: the owner of the view can be any authorization ID, provided the view does not refer to user tables or views in the first FROM clause of its defining fullselect. (It could refer instead, for example, to catalog tables or views thereof.)

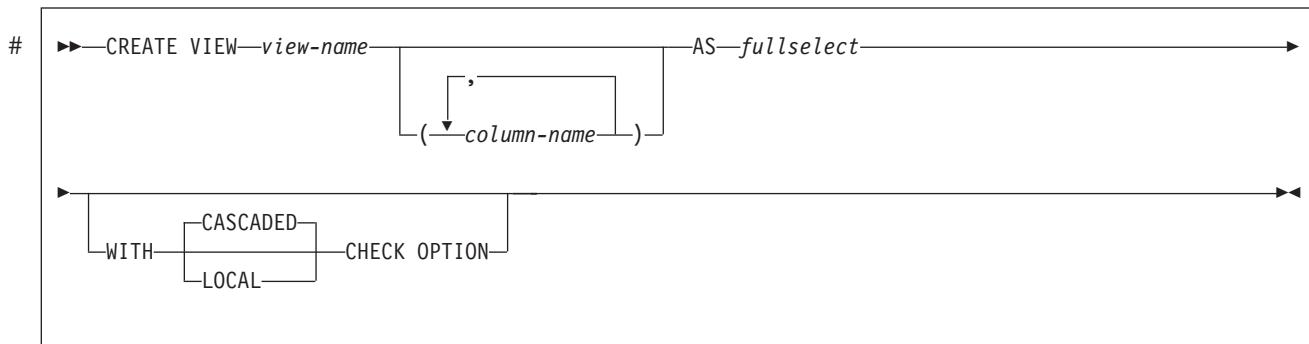
If the view satisfies the rules in the preceding paragraph, and if no errors are present in the CREATE statement, the view is created, even if the owner has no privileges at all on the tables and views identified in the view's fullselect.

- If the SQL authorization ID of the process includes DBADM authority on at least one of the databases that contains a table from which the view is created, the owner of the view can be different from the SQL authorization ID if all of the following conditions are true:
  - The value of field DBADM CREATE AUTH was set to YES on panel DSNTIPP during DB2 installation.
  - The view is not based only on views.

**Note:** The owner of the view must have the SELECT privilege on all tables and views in the CREATE VIEW statement, or, if the owner does not have the SELECT privilege on a table, the creator must have DBADM authority on the database that contains that table.

- If the SQL authorization ID of the process lacks SYSADM, SYSCTRL, or DBADM authority, or if the SQL authorization ID of the process fails to meet any of the previous conditions, only the authorization IDs of the process can own the view. In this case, the privilege set is the privileges that are held by the authorization ID selected for ownership.

## Syntax



## Description

### *view-name*

Names the view. The name must not identify a table, view, alias, or synonym that exists at the current server.

If qualified, the name can be a two-part or three-part name. If a three-part name is used, the first part must match the value of the field DB2 LOCATION NAME of installation panel DSNTIPR at the current server. (If the current server is not the local DB2, this name is not necessarily the name in the CURRENT SERVER special register.) In either case, the authorization ID that qualifies the name is the view's owner.

If the view name is unqualified and the statement is embedded in an application program, the owner of the view is the authorization ID that serves as the implicit qualifier for unqualified object names. This is the authorization ID of the QUALIFIER operand when the plan or package was created or last rebound. If QUALIFIER was not used, the owner of the view is the owner of the package or plan.

If the view name is unqualified and the statement is dynamically prepared, the owner of the view is the SQL authorization ID of the process.

The owner of a view always acquires the SELECT privilege on the view and the authority to drop the view. If all of the privileges that are required to create the view are held with the GRANT option before the view is created, the owner of the view receives the SELECT privilege with the GRANT option. Otherwise, the owner receives the SELECT privilege without the GRANT option. For example, assume that a view is created on a table for which the owner has the SELECT privilege with the GRANT option and the view definition also refers to a user-defined function. If the owner's EXECUTE privilege on the user-defined function is held without the GRANT option, the owner acquires the SELECT privilege on the view without the GRANT option.

The owner can also acquire INSERT, UPDATE, and DELETE privileges on the view. Acquiring these privileges is possible if the view is not "read only", which means a single table or view is identified in the first FROM clause of the fullselect. For each privilege that the owner has on the identified table or view (INSERT, UPDATE, and DELETE) before the new view is created, the owner acquires that privilege on the new view. The owner receives the privilege with the GRANT option if the privilege is held on the table or view with the GRANT option. Otherwise, the owner receives the privilege without the GRANT option.

With appropriate DB2 authority, a process can create views for those who have no authority to create the views themselves. The owner of such a view has the SELECT privilege on the view, without the GRANT option, and can drop the view.

*column-name*,...

Names the columns in the view. If you specify a list of column names, it must consist of as many names as there are columns in the result table of the fullselect. Each name must be unique and unqualified. If you do not specify a list of column names, the columns of the view inherit the names of the columns of the result table of the fullselect.

You must specify a list of column names if the result table of the fullselect has duplicate column names or an unnamed column (a column derived from a constant, function, or expression that was not given a name by the AS clause).

**AS** *fullselect*

Defines the view. At any time, the view consists of the rows that would result if the fullselect were executed.

*fullselect* must not refer to any declared temporary tables. They must also not refer to host variables or include parameter markers (question marks). For an explanation of *fullselect*, see "fullselect" on 365.

## CREATE VIEW

### WITH ... CHECK OPTION

Specifies the constraint that every row that is inserted or updated through the view must conform to the definition of the view. DB2 enforces this constraint whenever rows of the view are inserted or updated. If the search condition is not true for an inserted or updated row, an error occurs and no rows are inserted or updated.

The search condition of a view is the search condition that is specified in the first WHERE clause of the fullselect that defines the view. If the view is defined without a search condition (a WHERE clause was not specified) then the view behaves as if it were defined with a search condition that is always true.

A check option must not be specified if any of the following conditions are true:

- The view is read-only.
- The search condition of the view includes a subquery, or the search condition of an underlying view includes a subquery.
- The search condition of the view includes a user-defined function that is nondeterministic or has an external action.
- The *fullselect* refers to a created temporary table.

A check option is ignored if the view is updatable but does not have a search condition. If a check option is specified for an updatable view that does not allow inserts, the constraint applies only to updates.

If a check option is not specified, the search condition of the view is not used to check any insert or update operations that use the view. Rows that do not conform to the definition of the view can be inserted or updated, but then the rows are not accessible through the view (SELECT \* FROM V).

The difference between the two forms of the check option, CASCDED and LOCAL, is meaningful only when views are defined on each other. The view upon which another view is directly or indirectly defined is an *underlying view*.

#### CASCDED

Update and insert operations on view V must satisfy the search conditions of view V and all underlying views, regardless of whether the underlying views were defined with a check option. Furthermore, every updatable view that is directly or indirectly defined on view V inherits those search conditions (the search conditions of view V and all underlying views of V) as a constraint on insert or update operations.

#### LOCAL

Update and insert operations on view V must satisfy the search conditions of view V and underlying views that are defined with a check option (either WITH CASCDED CHECK OPTION or WITH LOCAL CHECK OPTION). Furthermore, every updatable view that is directly or indirectly defined on view V inherits those search conditions (the search conditions of view V and all underlying views of V that are defined with a check option) as a constraint on insert or update operations.

The LOCAL form of the CHECK option lets you update or insert rows that do not conform to the search condition of view V. You can perform these operations if the view is directly or indirectly defined on a view that was defined without a check option.

Table 52 on page 715 illustrates the effect of using the default check option, CASCDED. The information in Table 52 on page 715 is based on the following views:

- CREATE VIEW V1 AS SELECT COL1 FROM T1 WHERE COL1 > 10

- CREATE VIEW V2 AS SELECT COL1 FROM V1 WITH CASCDED CHECK OPTION
- CREATE VIEW V3 AS SELECT COL1 FROM V2 WHERE COL1 < 100

Table 52. Examples using default check option, CASCDED

| SQL statement              | Description of result                                                                                                                                                                                           |
|----------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| INSERT INTO V1 VALUES(5)   | Succeeds because V1 does not have a check option and it is not dependent on any other view that has a check option.                                                                                             |
| INSERT INTO V2 VALUES(5)   | Results in an error because the inserted row does not conform to the search condition of V1 which is implicitly part of the definition of V2.                                                                   |
| INSERT INTO V3 VALUES(5)   | Results in an error because the inserted row does not conform to the search condition of V1.                                                                                                                    |
| INSERT INTO V3 VALUES(200) | Succeeds even though it does not conform to the definition of V3 (V3 does not have the view check option specified); it does conform to the definition of V2 (which does have the view check option specified). |

The difference between CASCDED and LOCAL is shown best by example. Consider the following updatable views, where x and y represent either LOCAL or CASCDED:

- V1 is defined on Table T0.
- V2 is defined on V1 WITH x CHECK OPTION.
- V3 is defined on V2.
- V4 is defined on V3 WITH y CHECK OPTION.
- V5 is defined on V4.

Table 53 shows the views in which search conditions are checked during an INSERT or UPDATE operation:

Table 53. Views in which search conditions are checked during INSERT and UPDATE operations

| View used in INSERT or UPDATE operation | x = LOCAL<br>y = LOCAL | x = CASCDED<br>y = CASCDED | x = LOCAL<br>y = CASCDED | x = CASCDED<br>y = LOCAL |
|-----------------------------------------|------------------------|----------------------------|--------------------------|--------------------------|
| V1                                      | None                   | None                       | None                     | None                     |
| V2                                      | V2                     | V2, V1                     | V2                       | V2, V1                   |
| V3                                      | V2                     | V2, V1                     | V2                       | V2, V1                   |
| V4                                      | V4, V2                 | V4, V3, V2, V1             | V4, V3, V2, V1           | V4, V2, V1               |
| V5                                      | V4, V2                 | V4, V3, V2, V1             | V4, V3, V2, V1           | V4, V2, V1               |

## Notes

**Authorization for views created for other users:** When a process with appropriate authority creates a view for another user that does not have authorization for the underlying table or view, the SELECT privilege for the created view is implicitly granted to the user.

**Read-only views:** A view is *read-only* if one or more of the following statements is true of its definition:

## CREATE VIEW

- The first FROM clause identifies more than one table or view, or identifies a table function
- The first SELECT clause specifies the keyword DISTINCT
- The outer fullselect contains a GROUP BY clause
- The outer fullselect contains a HAVING clause
- The first SELECT clause contains a column function
- It contains a subquery such that the base object of the outer fullselect, and of the subquery, is the same table
- The first FROM clause identifies a read-only view

A read-only view cannot be the object of an INSERT, UPDATE, or DELETE statement. A view that includes GROUP BY or HAVING cannot be referred to in a subquery of a basic predicate.

**Testing a view definition:** You can test the semantics of your view definition by executing SELECT \* FROM *view-name*.

**The two forms of a view definition:** Both the source and the operational form of a view definition are stored in the DB2 catalog. Those two forms are not necessarily equivalent because the operational form reflects the state that exists when the view is created. For example, consider the following statement:

```
CREATE VIEW V AS SELECT * FROM S;
```

In this example, S is a synonym or alias for A.T, which is a table with columns C1, C2, and C3. The operational form of the view definition is equivalent to:

```
SELECT C1, C2, C3 FROM A.T;
```

Adding columns to A.T using ALTER TABLE and dropping S does not affect the operational form of the view definition. Thus, if columns are added to A.T or if S is redefined, the source form of the view definition can be misleading.

**View restrictions:** A view definition cannot contain unions or references to remote objects. A view definition cannot map to more than 15 base table instances. A view definition cannot reference a declared global temporary table.

## Examples

*Example 1:* Create the view DSN8710.VPROJRE1. PROJNO, PROJNAME, PROJDEP, RESPEMP, FIRSTNME, MIDINIT, and LASTNAME are column names. The view is a join of tables and is therefore read-only.

```
CREATE VIEW DSN8710.VPROJRE1
 (PROJNO,PROJNAME,PROJDEP,RESPEMP,
 FIRSTNME,MIDINIT,LASTNAME)
 AS SELECT ALL
 PROJNO,PROJNAME,DEPTNO,EMPNO,
 FIRSTNME,MIDINIT,LASTNAME
 FROM DSN8710.PROJ, DSN8710.EMP
 WHERE RESPEMP = EMPNO;
```

In the example, the WHERE clause refers to the column EMPNO, which is contained in one of the base tables but is not part of the view. In general, a column named in the WHERE, GROUP BY, or HAVING clause need not be part of the view.

*Example 2:* Create the view DSN8710.FIRSTQTR that is the UNION ALL of three fullselects, one for each month of the first quarter of 2000. The common names are SNO, CHARGES, and DATE.

## CREATE VIEW

```
| CREATE VIEW DSN8710.FIRSTQTR (SNO, CHARGES, DATE) AS
| SELECT SNO, CHARGES, DATE
| FROM MONTH1
| WHERE DATE BETWEEN '01/01/2000' and '01/31/2000'
| UNION A11
| SELECT SNO, CHARGES, DATE
| FROM MONTH2
| WHERE DATE BETWEEN '02/01/2000' and '02/29/2000'
| UNION A11
| SELECT SNO, CHARGES, DATE
| FROM MONTH3
| WHERE DATE BETWEEN '03/01/2000' and '03/31/2000';
|
```

---

## DECLARE CURSOR

The DECLARE CURSOR statement defines a cursor.

### Invocation

This statement can only be embedded in an application program. It is not an executable statement.

### Authorization

For each table or view identified in the SELECT statement of the cursor, the privilege set must include at least one of the following:

- The SELECT privilege
- Ownership of the object
- DBADM authority for the corresponding database (tables only)
- SYSADM authority
- SYSCTRL authority (catalog tables only)

The SELECT statement of the cursor is one of the following:

- The prepared select statement identified by *statement-name*
- The specified *select-statement*

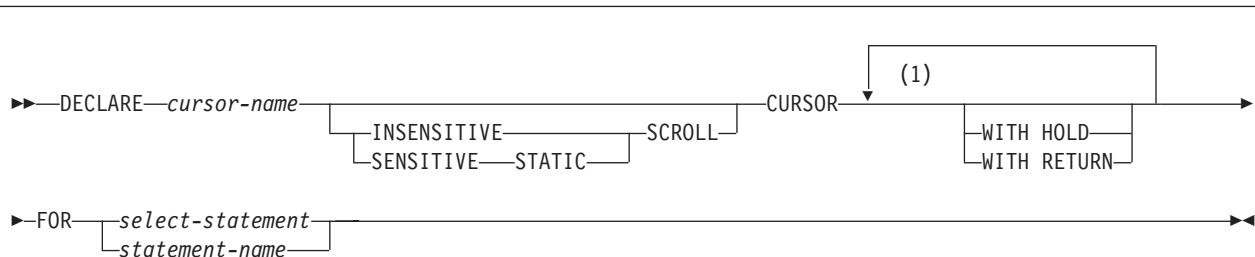
#### If *statement-name* is specified:

- The privilege set is determined by the DYNAMICRULES behavior in effect (run, bind, define, or invoke) and is summarized in Table 40 on page 382. (For more information on these behaviors, including a list of the DYNAMICRULES bind option values that determine them, see “Authorization IDs and dynamic SQL” on page 43.)
- The authorization check is performed when the SELECT statement is prepared.
- The cursor cannot be opened unless the SELECT statement is successfully prepared.

#### If *select-statement* is specified:

- The privilege set consists of the privileges that are held by the authorization ID of the owner of the plan or package.
- If the plan or package is bound with VALIDATE(BIND), the authorization check is performed at bind time, and the bind is unsuccessful if any required privilege does not exist.
- If the plan or package is bound with VALIDATE(RUN), an authorization check is performed at bind time, but all required privileges need not exist at that time. If all privileges exist at bind time, no authorization checking is performed when the cursor is opened. If any privilege does not exist at bind time, an authorization check is performed the first time the cursor is opened within a unit of work. The OPEN is unsuccessful if any required privilege does not exist.

## Syntax



### Notes:

- 1 The same clause must not be specified more than once.

## Description

### *cursor-name*

Names the cursor. The name must not identify a cursor that has already been declared in the source program.

### INSENSITIVE

Specifies that changes to the base table after the result table is *materialized* are not visible to the cursor. (The values of the rows in the result table have already been captured from the database.) The cursor does not have sensitivity to any updates or deletes made to the rows underlying its result table after the table is materialized. As a result, the size of the result table, the order of the rows, and the values for each row do not change after the cursor is opened. A cursor defined as INSENSITIVE is *read-only* (that is, cannot be used for positioned updates or deletes). Any update or delete attempt results in an error.

Additionally, if the SELECT statement contains a FOR UPDATE clause, an error is returned.

### SENSITIVE

Specifies that changes made to the database after the result table is materialized are visible to the cursor. The cursor has some level of sensitivity to any updates or deletes made to the rows underlying its result table after the table is materialized. The cursor is always sensitive to positioned updates or deletes using the same cursor. Additionally, the cursor can have sensitivity to committed changes made outside this cursor.

If DB2 cannot make changes visible to the cursor, then an error is issued at bind time for OPEN CURSOR. DB2 cannot make changes visible to the cursor when the cursor implicitly becomes read-only. Such is the case when the result table must be materialized, as when the FROM clause of the SELECT statement contains more than one table or view. The current list of conditions that result in an implicit read-only cursor can be found in "Notes" on page 721. Whether the cursor is sensitive to changes made outside this cursor also depends on whether a SENSITIVE or INSENSITIVE FETCH statement is used.

### STATIC

Specifies that the size of the result table and the order of the rows does not change after the cursor is opened. Rows inserted into the underlying table are not added to the result table regardless of how the rows are inserted.

## DECLARE CURSOR

Rows in the result table do not move if columns in the ORDER BY clause are updated in rows that have already been materialized. Positioned updates and deletes are allowed if the result table is updatable.

A STATIC cursor has visibility to changes made by *this* cursor using positioned updates or deletes. Committed changes made outside this cursor are visible with the SENSITIVE option of the FETCH statement. A FETCH SENSITIVE can result in a *hole* in the result table (that is, a difference between the result table and its underlying base table). If an updated row in the base table of a cursor no longer satisfies the predicate of its SELECT statement, an update hole occurs in the result table. If a row of a cursor was deleted in the base table, a delete hole occurs in the result table. When a FETCH SENSITIVE detects an update hole, no data is returned (a warning is issued), and the cursor is left positioned on the update hole. When a FETCH SENSITIVE detects a delete hole, no data is returned (a warning is issued), and the cursor is left positioned on the delete hole.

Updates through a cursor result in an automatic re-fetch of the row. This re-fetch means that updates can create a hole themselves. The re-fetched row also reflects changes as a result of triggers updating the same row. It is important to reflect these changes to maintain the consistency of data in the row.

Using a non-deterministic function (built-in or user-defined) in the WHERE clause of the *select-statement* or *statement-name* of a SENSITIVE STATIC cursor can cause misleading results. This situation occurs because DB2 constructs a temporary result table and retrieves rows from this table for FETCH INSENSITIVE statements. When DB2 processes a FETCH SENSITIVE statement, rows are fetched from the underlying table and predicates are re-evaluated. Using a non-deterministic function can yield a different result on each FETCH SENSITIVE of the same row, which could also result in the row no longer being considered a match.

A FETCH INSENSITIVE on a SENSITIVE STATIC SCROLL cursor is not sensitive to changes made outside the cursor, unless a previous FETCH SENSITIVE has already refreshed that row; however, positioned updates and delete changes with the cursor are visible.

STATIC cursors are insensitive to insertions.

### SCROLL

Specifies that the cursor is scrollable. If SCROLL is specified, INSENSITIVE or SENSITIVE STATIC must be specified. A scrollable cursor permits arbitrary navigation through the rows of its result table. If SCROLL is not specified, the cursor is forward moving only, and the rows of its result table can be fetched only once, in a predetermined order.

### WITH HOLD

Prevents the cursor from being closed as a consequence of a commit operation. A cursor declared with WITH HOLD is closed at commit time if one of the following is true:

- The connection associated with the cursor is in the release pending status.
- The bind option DISCONNECT(AUTOMATIC) is in effect.
- The environment is one in which the option WITH HOLD is ignored.

When WITH HOLD is specified, a commit operation commits all the changes in the current unit of work, but releases only locks that are not required to maintain the cursor. Afterwards, an initial FETCH statement is required before a

positioned update or delete statement can be executed. After the initial FETCH, the cursor is positioned on the row following the one it was positioned on before the commit operation.

All cursors are implicitly closed by a connect (Type 1) or rollback operation. A cursor is also implicitly closed by a commit operation if WITH HOLD is ignored or not specified.

Cursors that are declared with WITH HOLD in CICS or in IMS non-message-driven programs will not be closed by a rollback operation if the cursor was opened in a previous unit of work and no changes have been made to the database in the current unit of work. The cursor cannot be closed because CICS and IMS do not broadcast the rollback request to DB2 for a null unit of work.

If a cursor is closed before the commit operation, the effect is the same as if the cursor was declared without the option WITH HOLD.

WITH HOLD is ignored in IMS message driven programs (MPP, IFP, and message-driven BMP). WITH HOLD maintains the cursor position in a CICS pseudo-conversational program until the end-of-task (EOT).

For details on restrictions that apply to declaring cursors with WITH HOLD, see Part 2 of *DB2 Application Programming and SQL Guide*.

#### **WITH RETURN**

Specifies that the cursor, if it is declared in a stored procedure, can return a result set to the caller.

##### *select-statement*

Specifies the result table of the cursor. The *select-statement* must not include parameter markers, but can include references to host variables. The declarations of the host variables must precede the DECLARE CURSOR statement in the source program. See “*select-statement*” on page 369 for an explanation of *select-statement*.

##### *statement-name*

Identifies the prepared *select-statement* that specifies the result table of the cursor whenever the cursor is opened. The *statement-name* must not be identical to a statement name specified in another DECLARE CURSOR statement of the source program. For an explanation of prepared SELECT statements, see “PREPARE” on page 846.

## Notes

A cursor in the open state designates a result table and a position relative to the rows of that table. The table is the result table specified by the SELECT statement of the cursor.

**Read-only cursors:** If the result table is *read-only*, the cursor is *read-only*. The result table is *read-only* if one or more of the following statements is true about the SELECT statement of the cursor:

- The first FROM clause identifies or contains any of the following:
  - More than one table or view
  - A catalog table with no updatable columns
  - A read-only view
  - A nested table expression
  - A table function
- The first SELECT clause specifies the keyword DISTINCT, contains a column function, or uses both

## DECLARE CURSOR

- The outer subselect contains a GROUP BY clause, a HAVING clause, or both clauses
- It contains a subquery such that the base object of the outer subselect, and of the subquery, is the same table
- Any of the following operators or clauses are specified:
  - A UNION or UNION ALL operator
  - An ORDER BY clause (except when the cursor is declared as SENSITIVE STATIC scrollable)
  - A FOR FETCH ONLY or a FOR READ ONLY clause
- It is executed with isolation level UR and a FOR UPDATE clause is not specified.

If the result table is not *read-only*, the cursor can be used to update or delete the underlying rows of the result table.

**TEMP database requirement for scrollable cursors:** To use a scrollable cursor, you must first create a TEMP database and table spaces in this database because a scrollable cursor requires a temporary table for its result table while the cursor is open. DB2 chooses a table space to use for the temporary result table.

**Cursors in COBOL and Fortran programs:** In COBOL and Fortran source programs, the DECLARE CURSOR statement must precede all statements that explicitly refer to the cursor by name. This rule does not necessarily apply to the other host languages because the precompiler provides a two-pass option for these languages. This rule applies to other host languages if the two-pass option is not used.

**Scope of a cursor:** The scope of *cursor-name* is the source program in which it is defined; that is, the application program submitted to the precompiler. Thus, you can only refer to a cursor by statements that are precompiled with the cursor declaration. For example, a COBOL program called from another program cannot use a cursor that was opened by the calling program. Furthermore, a cursor defined in a Fortran subprogram can only be referred to in that subprogram. Cursors that specify WITH RETURN in a procedure and are left open are returned as result sets.

Although the scope of a cursor is the program in which it is declared, each package (or DBRM of a plan) created from the program includes a separate instance of the cursor, and more than one instance of the cursor can be used in the same execution of the program. For example, assume a program is precompiled with the CONNECT(2) option and its DBRM is used to create a package at location X and a package at location Y. The program contains the following SQL statements:

```
DECLARE C CURSOR FOR ...
CONNECT TO X
OPEN C
FETCH C INTO ...
CONNECT TO Y
OPEN C
FETCH C INTO ...
```

The second OPEN C statement does not cause an error because it refers to a different instance of cursor C. The same notion applies to a single location if the packages are in different collections and the SET CURRENT PACKAGESET statement is used to select the packages.

**Positioned deletes and isolation level UR:** Specify FOR UPDATE if you want to use the cursor for a positioned DELETE and the isolation level is UR because of a BIND option. In this case, the isolation level is CS.

**Returning a result set from a stored procedure:** A cursor that is declared in a stored procedure returns a result set when all of the following conditions are true:

- The cursor is declared with the WITH RETURN option. In a distributed environment, blocks of each result set of the cursor's data are returned with the CALL statement reply.
- The cursor is left open after exiting from the stored procedure. A cursor declared with the SCROLL option must be left positioned *before* the first row before exiting from the stored procedure.
- The cursor is declared with the WITH HOLD option if the stored procedure performs a COMMIT\_ON\_RETURN.

The result set is the set of all rows after the current position of the cursor after exiting the stored procedure. The result set is assumed to be read-only. If that same procedure is reinvoked, open result set cursors for a stored procedure at a given site are automatically closed by the database management system.

**Scrollable cursors specified with user-defined functions:** A row can be fetched more than once with a scrollable cursor. Therefore, if a scrollable cursor is defined with a non-deterministic function in the select list of the cursor, a row can be fetched multiple times with different results for each fetch. (However, the value of a non-deterministic function in the WHERE clause of a scrollable cursor is captured when the cursor is opened and remains unchanged until the cursor is closed.) Similarly, if a scrollable cursor is defined with a user-defined function with external action, the action is executed with every fetch.

## Examples

The statements in the following examples are assumed to be in PL/I programs.

*Example 1:* Declare C1 as the cursor of a query to retrieve data from the table DSN8710.DEPT. The query itself appears in the DECLARE CURSOR statement.

```
EXEC SQL DECLARE C1 CURSOR FOR
 SELECT DEPTNO, DEPTNAME, MGRNO
 FROM DSN8710.DEPT
 WHERE ADMRDEPT = 'A00';
```

*Example 2:* Declare C2 as the cursor for a statement named STMT2.

```
EXEC SQL DECLARE C2 CURSOR FOR STMT2;
```

*Example 3:* Declare C3 as the cursor for a query to be used in positioned updates of the table DSN8710.EMP. Allow the completed updates to be committed from time to time without closing the cursor.

```
EXEC SQL DECLARE C3 CURSOR WITH HOLD FOR
 SELECT * FROM DSN8710.EMP
 FOR UPDATE OF WORKDEPT, PHONENO, JOB, EDLEVEL, SALARY;
```

Instead of specifying which columns should be updated, you could use a FOR UPDATE clause without the names of the columns to indicate that all updatable columns are updated.

*Example 4:* In stored procedure SP1, declare C4 as the cursor for a query of the table DSN8710.PROJ. Enable the cursor to return a result set to the caller of SP1, which performs a commit on return.

```
EXEC SQL DECLARE C4 CURSOR WITH HOLD WITH RETURN FOR
 SELECT PROJNO, PROJNAME
 FROM DSN8710.PROJ
 WHERE DEPTNO = 'A01';
```

## DECLARE CURSOR

*Example 5:* In the following example, the DECLARE CURSOR statement associates the cursor name C5 with the results of the SELECT. C5 allows positioned updates and deletes because the result table can be updated.

```
DECLARE C5 SENSITIVE STATIC SCROLL CURSOR FOR
 SELECT DEPTNO, DEPTNAME, MGRNO
 FROM DSN8710.DEPT
 WHERE ADMRDEPT = 'A00';
```

*Example 6:* In the following example, the DECLARE CURSOR statement associates the cursor name C6 with the results of the SELECT.

```
DECLARE C6 INSENSITIVE SCROLL CURSOR FOR
 SELECT DEPTNO, DEPTNAME, MGRNO
 FROM DSN8710.DEPT
 WHERE DEPTNO;
```

## DECLARE GLOBAL TEMPORARY TABLE

The DECLARE GLOBAL TEMPORARY TABLE statement defines a declared temporary table for the current application process and instantiates an empty instance of the table for the process.

### Invocation

This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared.

### Authorization

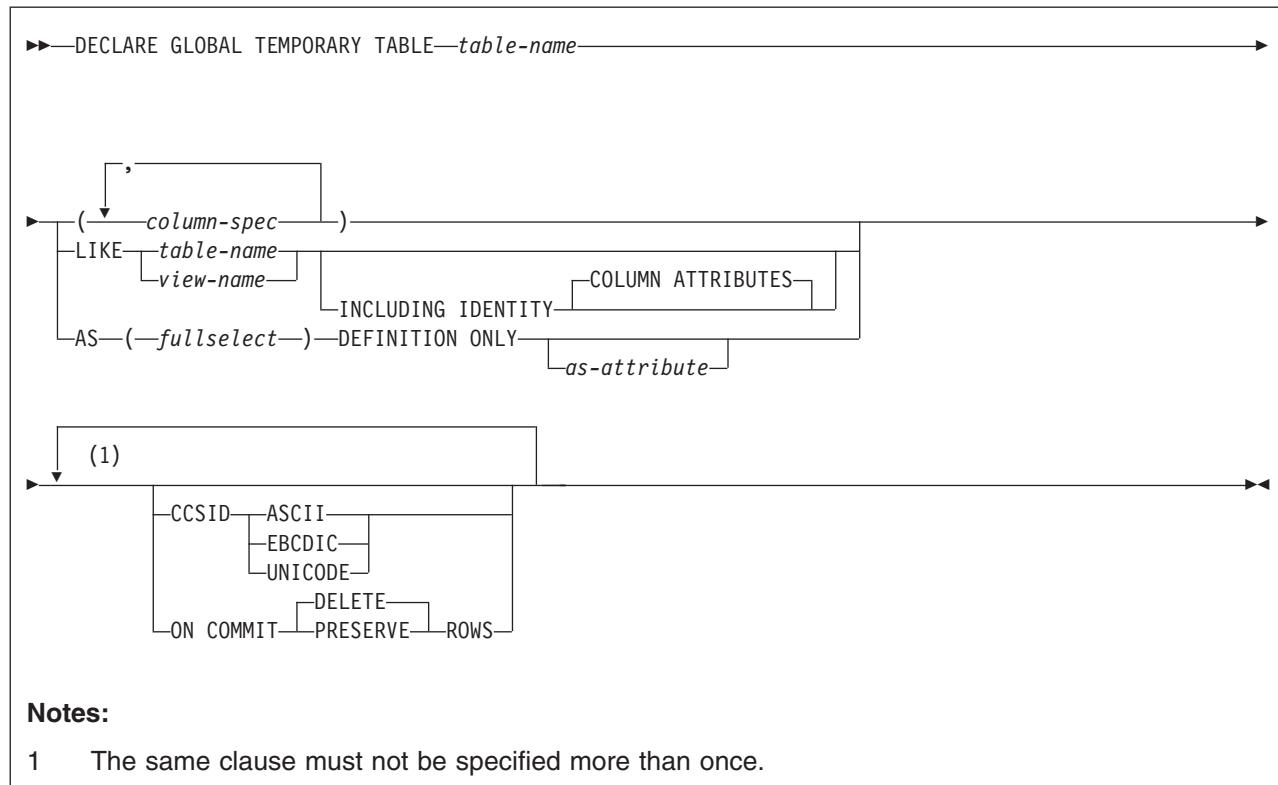
None are required, unless the LIKE clause is specified when additional privileges might be required.

PUBLIC implicitly has the following privileges without GRANT authority for declared temporary tables:

- The CREATETAB privilege to define a declared temporary table in the database that is defined AS TEMP, which is the database for declared temporary tables.
- The USE privilege to use the table spaces in the database that is defined as TEMP.
- All table privileges on the table and authority to drop the table. (Table privileges for a declared temporary table cannot be granted or revoked.)

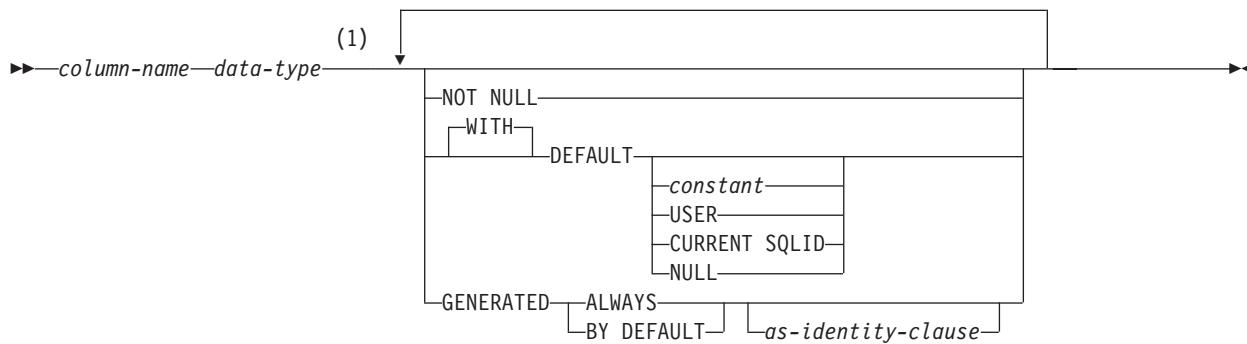
These implicit privileges are not recorded in the DB2 catalog and cannot be revoked.

### Syntax



## DECLARE GLOBAL TEMPORARY TABLE

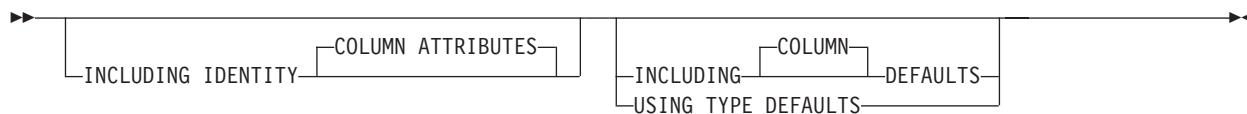
### column-spec:



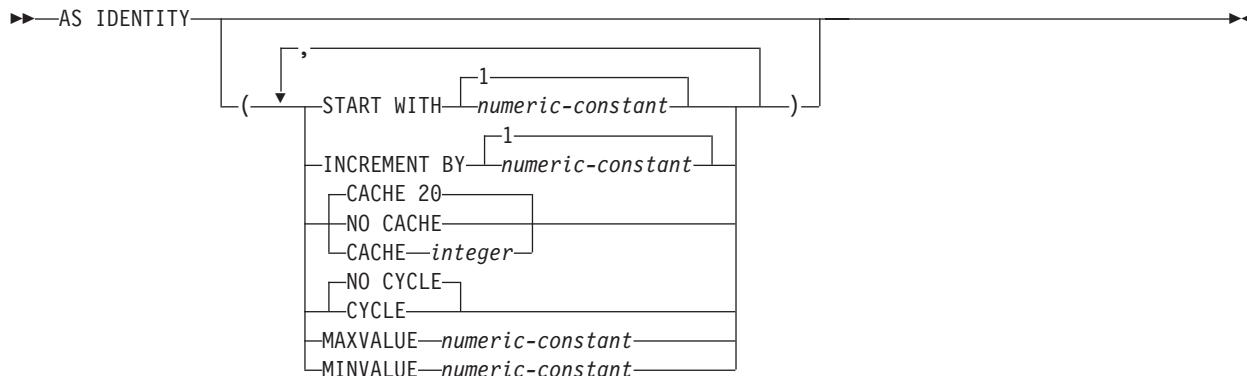
### Notes:

- 1 The same clause must not be specified more than once. The FOR sub-type DATA clause can be specified as part of *data-type*.

### as-attribute:



### as-identity-clause:



## Description

*table-name*

Names the temporary table. The qualifier, if specified explicitly, must be

## DECLARE GLOBAL TEMPORARY TABLE

SESSION. If the qualifier is not specified, it is implicitly defined to be SESSION. The name must not identify a declared temporary table that exists in the application process.

If a table, view, synonym, or alias already exists with the same name and an implicit or explicit qualifier of SESSION:

- The declared temporary table is still defined with SESSION.*table-name*. An error is not issued because the resolution of a declared temporary table name does not include the persistent and shared names in the DB2 catalog tables.
- Any references to SESSION.*table-name* will resolve to the declared temporary table rather than to any existing SESSION.*table-name* whose definition is persistent and is in the DB2 catalog tables.

PUBLIC implicitly acquires ALL PRIVILEGES on the table and authority to drop the table. These implicit privileges are not recorded in the DB2 catalog and cannot be revoked.

### *column-spec*

Defines the attributes of a column for each instance of the table. The number of columns defined must not exceed 750. The maximum record size must not exceed 32714 bytes. The maximum row size must not exceed 32706 bytes (8 bytes less than the maximum record size).

### *column-name*

Names the column. The name must not be qualified and must not be the same as the name of another column in the table.

### *data-type*

Specifies the data type of the column. The data type can be any built-in data type that can be specified for the CREATE TABLE statement except for a LOB (BLOB, CLOB, and DBCLOB) or ROWID type. The FOR *subtype* DATA clause can be specified as part of *data-type*. For more information on the data types and the rules that apply to them, see “built-in-data-type” on page 658.

### NOT NULL

Specifies that the column cannot contain nulls. Omission of NOT NULL indicates that the column can contain nulls.

### DEFAULT

The default value assigned to the column in the absence of a value specified on INSERT. Do not specify DEFAULT for a column that is defined AS IDENTITY (an identity column); DB2 generates default values..

If DEFAULT is not specified, the default value for the column is the null value.

If DEFAULT is specified without a value after it, the default value of the column depends on the data type of the column, as follows:

| Data type             | Default value        |
|-----------------------|----------------------|
| Numeric               | 0                    |
| Fixed-length string   | Blanks               |
| Varying-length string | A string of length 0 |
| Date                  | CURRENT DATE         |
| Time                  | CURRENT TIME         |
| Timestamp             | CURRENT TIMESTAMP    |

A default value other than the one that is listed above can be specified in one of the following forms:

## DECLARE GLOBAL TEMPORARY TABLE

### *constant*

Specifies a constant as the default value for the column. The value of the constant must conform to the rules for assigning that value to the column.

### **USER**

Specifies the value of the USER special register at the time of INSERT or LOAD as the default value for the column. If USER is specified, the data type of the column must be a character string with a length attribute greater than or equal to the length attribute of the USER special register, which is 8 bytes.

### **CURRENT SQLID**

Specifies the value of the SQL authorization ID of the process at the time of INSERT or LOAD as the default value for the column. If CURRENT SQLID is specified, the data type of the column must be a character string with a length attribute greater than or equal to the length attribute of the CURRENT SQLID special register, which is 8 bytes.

### **NULL**

Specifies null as the default value for the column.

In a given column definition:

- NOT NULL and DEFAULT NULL cannot both be specified.
- DEFAULT cannot be specified for an identity column.
- Omission of NOT NULL and DEFAULT for a column other than an identity column is an implicit specification of DEFAULT NULL. For an identity column, it is an implicit specification of NOT NULL, and DB2 generates default values.

For more information on the effect of specifying various combinations of the NOT NULL and DEFAULT clauses, see Table 46 on page 664, which provides a summary.

### **GENERATED**

Specifies that DB2 generates values for the column. You must specify GENERATED if the column is to be considered an identity column (a column defined with the AS IDENTITY clause).

### **ALWAYS**

Specifies that DB2 always generates a value for the column when a row is inserted into the table.

### **BY DEFAULT**

Specifies that DB2 generates a value for the column when a row is inserted into the table unless a value is specified. BY DEFAULT is the recommended value only when you are using data propagation.

### **AS IDENTITY**

Specifies that the column is an identity column for the table. A table can have only one identity column. AS IDENTITY can be specified only if the data type for the column is an exact numeric type with a scale of zero (SMALLINT, INTEGER, DECIMAL with a scale of zero). For more information, see the description of the AS IDENTITY clause for “CREATE TABLE” on page 653.

### **LIKE *table-name* or *view-name***

Specifies that the columns of the table have the same name, data type, and nullability attributes as the columns of the identified table or view. If a table is identified, the column default attributes are also defined by that table. The name

## DECLARE GLOBAL TEMPORARY TABLE

specified must identify a table, view, synonym, or alias that exists at the current server. The identified table must not be an auxiliary table or a declared temporary table.

The privilege set must include the SELECT privilege on the identified table or view.

This clause is similar to the LIKE clause on CREATE TABLE, but it has the following differences:

- If LIKE results in a column having a LOB data type, a ROWID data type, or distinct type, the DECLARE GLOBAL TEMPORARY TABLE statement fails.
- In addition to these data type restrictions, if any column has any other attribute value that is not allowed in a declared temporary table, that attribute value is ignored. The corresponding column in the new temporary table has the default value for that attribute unless otherwise indicated.

When the identified object is a table, the column name, data type, nullability, and default attributes are determined from the columns of the specified table; any identity column attributes are inherited only if the INCLUDING IDENTITY COLUMN ATTRIBUTES clause is specified.

### INCLUDING IDENTITY COLUMN ATTRIBUTES

Specifies that the new table inherits the identity attributes of the identity column. If the table identified by LIKE does not have an identity column, the INCLUDING IDENTITY clause is ignored. If the LIKE clause identifies a view, INCLUDING IDENTITY COLUMN ATTRIBUTES cannot be specified.

### AS (*fullselect*) DEFINITION ONLY

Specifies that the columns of the table are to have the same name and description as the columns that would appear in the derived result table of the *fullselect* if the *fullselect* were to be executed. The use of AS *fullselect* is an implicit definition of *n* columns for the table, where *n* is the number of columns that would result from the *fullselect*.

DEFINITION ONLY indicates that the *fullselect* is not executed. Therefore, there is no result table with a set of rows with which to automatically populate the declared temporary table. However, you can use the INSERT INTO statement with the same *fullselect* specified in the AS clause to populate the declared temporary table with the set of rows from the result table of the *fullselect*.

The implicit definition includes all the attributes of the *n* columns of *fullselect* that are applicable for a declared temporary table (see *column-spec*) with the exception of these column attributes:

- The default value assigned to a column when a value is not specified on INSERT
- The identity attribute, if any

The behavior of these column attributes is controlled with the INCLUDING or USING TYPE DEFAULTS clauses, which are defined below.

If *fullselect* results in a column having a LOB data type, a ROWID data type, or a distinct type, the DECLARE GLOBAL TEMPORARY statement fails.

If *fullselect* results in other column attributes that are not applicable for a declared temporary table, those attributes are ignored in the implicit definition for the declared temporary table.

## DECLARE GLOBAL TEMPORARY TABLE

The implicitly defined columns of the declared temporary table inherit the names of the columns from the result table of the *fullselect*. Therefore, a column name must be specified in the *fullselect* for all result columns. For result columns that are derived from expressions, constants, and functions, the *fullselect* must include the AS *column-name* clause immediately after the result column.

The *fullselect* must not refer to host variables or include parameter markers (question marks).

### INCLUDING IDENTITY COLUMN ATTRIBUTES

Specifies that the declared temporary table inherits the identity attribute, if any, of the columns resulting from *fullselect*. In general, the identity attribute is copied if the element of the corresponding column in the table, view, or *fullselect* is the name of a table column or the name of a view column that directly or indirectly maps to the name of a base table column with the identity property. The columns of the new table do not inherit the identity attribute in the following cases:

- The select list of the *fullselect* includes multiple instances of an identity column name (that is, selecting the same column more than once).
- The select list of the *fullselect* includes multiple identity columns (that is, it involves a join).
- The identity column is included in an expression in the select list.
- The *fullselect* includes a set operation (union).

If INCLUDING IDENTITY is not specified, the declared temporary table will not have an identity column.

### INCLUDING COLUMN DEFAULTS

Specifies that the declared temporary table inherits the default values of the columns resulting from *fullselect*. A default value is the value assigned to a column when a value is not specified on an INSERT.

Do not specify INCLUDING COLUMN DEFAULTS, if you specify USING TYPE DEFAULTS.

If neither INCLUDING COLUMN DEFAULTS nor USING TYPE DEFAULTS is specified, the default values of the columns of the declared temporary table are either null or there are no default values. If the column can be null, the default is the null value; if the column cannot be null, there is no default value, and an error occurs if a value is not provided for a column on an INSERT for the declared temporary table.

### USING TYPE DEFAULTS

Specifies that the default values for the declared temporary table depend on the data type of the columns that result from *fullselect*, as follows:

| Data type             | Default value        |
|-----------------------|----------------------|
| Numeric               | 0                    |
| Fixed-length string   | Blanks               |
| Varying-length string | A string of length 0 |
| Date                  | CURRENT DATE         |
| Time                  | CURRENT TIME         |
| Timestamp             | CURRENT TIMESTAMP    |

Do not specify USING TYPE DEFAULTS, if you specify INCLUDING COLUMN DEFAULTS.

**CCSID encoding-scheme**

Specifies the encoding scheme for string data that is stored in the table. For declared temporary tables, the encoding scheme for the data cannot be specified for the table space or database, and all data in one table space or the database need not use the same encoding scheme. Because there can be only one TEMP database for all declared temporary tables for each DB2 member, there can be a mixture of encoding schemes in both the database and each table space.

For the creation of temporary tables, the CCSID clause can be specified whether or not the LIKE clause is specified. If the CCSID clause is specified, the encoding scheme of the new table is the scheme that is specified in the CCSID clause. If the CCSID clause is not specified, the encoding scheme of the new table is the same as the scheme for the table specified in the LIKE clause or as the scheme for the table identified by the AS (fullselect) clause.

**ASCII** Specifies that the data is encoded by using the ASCII CCSIDs of the server.

**EBCDIC**

Specifies that the data is encoded by using the EBCDIC CCSIDs of the server.

**UNICODE**

Specifies that the data is encoded by using the UNICODE CCSIDs of the server.

An error occurs if the CCSIDs for the encoding scheme have not been defined. Usually, each encoding scheme requires only a single CCSID. Additional CCSIDs are needed when mixed, graphic, or UNICODE data is used.

**ON COMMIT commit-action ROWS**

Specifies whether the contents of the table are to be deleted or preserved across a commit operation.

**DELETE**

The rows of the table are deleted if no WITH HOLD cursors are open on the table. DELETE is the default.

**PRESERVE**

The rows of the table are preserved. Thread reuse capability is not available to any application process or thread that contains, at its most recent COMMIT, an active declared temporary table that was defined with the ON COMMIT PRESERVE ROWS clause.

## Notes

**Instantiation, scope, and termination:** Let P denote an application process and let T be a declared temporary table in an application program in P:

- When a program in P issues a DECLARE GLOBAL TEMPORARY TABLE statement, an empty instance of T is created.
- Any program in P can reference T, and any of those references is a reference to that same instance of T. (If a DECLARE GLOBAL TEMPORARY statement is specified within the SQL procedure language compound statement, BEGIN-END, the scope of the declared temporary table is the application process and not the compound statement.)

If T was declared at a remote server, the reference to T must use the same DB2 connection that was used to declare T and that connection must not have been

## DECLARE GLOBAL TEMPORARY TABLE

terminated after T was declared. When the connection to the database server at which T was declared terminates, T is dropped, and its instantiated rows are destroyed.

- If T is defined with the ON COMMIT DELETE ROWS clause, when a commit operation terminates a unit of work in P and no program in P has a WITH HOLD cursor open that is dependent on T, the commit includes the operation DELETE FROM T (all rows).
- When a rollback operation terminates a unit of work in P, the rollback undoes the rows of T up to the last commit or specified external savepoint but leaves all rows that existed up to that point.
- When the application process that declared T terminates, T is dropped, and its instantiated rows are destroyed.

**Thread reuse:** If a declared temporary table is defined in an application process that is running as a local thread, the application process or local thread that declared the table qualifies for explicit thread reuse if:

- The table was defined with the ON COMMIT DELETE ROWS attribute, which is the default.
- The table was defined with the ON PRESERVE COMMIT DELETE ROWS attribute and the table was explicitly dropped with the DROP TABLE statement before the thread's commit operation.

When the thread is reused, the declared temporary table is dropped and its rows are destroyed. However, if you do not explicitly drop all declared temporary tables before your thread performs a commit and the thread becomes idle waiting to be reused, as with all thread reuse situations, the idle thread holds resources and locks. This includes some declared temporary table resources and locks on the table spaces and the database descriptor (DBD) for the TEMP database. So, instead of using the implicit drop feature of thread reuse to drop your declared temporary tables, it is recommended that you explicitly use the DROP TABLE statement to drop your declared temporary tables before the thread performs a commit operation and becomes idle. Explicitly dropping the tables enables you to maximize the use of declared temporary table resources and release locks when multiple threads are using declared temporary table.

Remote threads qualify for thread reuse differently than local threads. If a declared temporary table is defined (with or without ON COMMIT DELETE ROWS) in an application process that is running as a remote or DDF thread (also known as Database Access Thread or DBAT), the remote thread qualifies for thread reuse only when the declared temporary table is explicitly dropped before the thread performs a commit operation. Dropping the declared temporary table enables the remote thread to qualify for the implicit thread reuse that is supported for DDF threads via connection pooling and to become an inactive type 1 or type 2 thread.

**Privileges:** When a declared temporary table is defined, PUBLIC implicitly is granted all table privileges on the table and authority to drop the table. These implicit privileges are not recorded in the DB2 catalog and cannot be revoked.

**Referring to a declared temporary table in other SQL statements:** Many SQL statements support declared temporary tables. To refer to a declared temporary table in an SQL statement other than DECLARE GLOBAL TEMPORARY TABLE, you must qualify the table name with SESSION. You can either specify SESSION explicitly in the table name or use the QUALIFIER bind option to specify SESSION as the qualifier for all SQL statements in the plan or package.

## DECLARE GLOBAL TEMPORARY TABLE

If you use SESSION as the qualifier for a table name but the application process does not include a DECLARE GLOBAL TEMPORARY TABLE statement for the table name, DB2 assumes that you are not referring to a declared temporary table. DB2 resolves such table references to a table whose definition is persistent and appears in the DB2 catalog tables.

With the exception of the DECLARE GLOBAL TEMPORARY TABLE statement, any static SQL statement that references a declared temporary table is incrementally bound at run time. This is because the definition of the declared temporary table does not exist until the DECLARE GLOBAL TEMPORARY statement is executed in the application process that contains those SQL statements and the definition does not persist when the application process finishes running.

When a plan or package is bound, any static SQL statement (other than the DECLARE GLOBAL TEMPORARY TABLE statement) that references a *table-name* that is qualified by SESSION, regardless of whether the reference is for a declared temporary table, is not completely bound. However, the bind of the plan or package succeeds if there are no other errors. These static SQL statements are then incrementally bound at run time when the static SQL statement is issued. This is necessary because:

- The definition of the declared temporary table does not exist until the DECLARE GLOBAL TEMPORARY TABLE statement for the table is executed in the same application process that contains those SQL statements. Therefore, DB2 must wait until the plan or package is run to determine if SESSION.*table-name* refers to a base table or a declared temporary table.
- The definition of a declared temporary table does not persist after the table it is explicitly dropped (DROP statement) or the application process that defined it finishes running. When the application process terminates or is re-used as a reusable application thread, the instantiated rows of the table are deleted and the definition of the declared temporary table is dropped if it has not already been explicitly dropped.

After the plan or package is bound, any static SQL statement that refers to a *table-name* that is qualified by SESSION has a new statement status of M in the DB2 catalog table (STATUS column of SYSIBM.SYSSTMT or SYSIBM.SYSPACKSTMT).

**Parallelism support:** Only I/O and CP parallelism are supported. Any query that involves a declared temporary table is limited to parallel tasks on a single CPC.

**Restrictions on the use of declared temporary tables:** Declared temporary tables cannot:

- Be specified in referential constraints.
- Be referenced in any SQL statements that are defined in a trigger body (CREATE TRIGGER statement). If you refer a table name that is qualified with SESSION in a trigger body, DB2 assumes that you are referring to a base table.

In addition, do not refer to a declared temporary table in any of the following statements.

## DECLARE GLOBAL TEMPORARY TABLE

|                               |                        |
|-------------------------------|------------------------|
| ALTER INDEX                   | CREATE VIEW            |
| ALTER TABLE                   | GRANT (table or view)  |
| COMMENT ON                    | LABEL ON               |
| CREATE ALIAS                  | LOCK TABLE             |
| CREATE FUNCTION (TABLE LIKE)  | RENAME TABLE           |
| CREATE PROCEDURE (TABLE LIKE) | REVOKE (table or view) |
| CREATE TRIGGER                |                        |

## Examples

*Example 1:* Define a declared temporary table with column definitions for an employee number, salary, commission, and bonus.

```
DECLARE GLOBAL TEMPORARY TABLE SESSION.TEMP_EMP
 (EMPNO CHAR(6) NOT NULL,
 SALARY DECIMAL(9, 2),
 BONUS DECIMAL(9, 2),
 COMM DECIMAL(9, 2)
 CCSID EBCDIC
 ON COMMIT PRESERVE ROWS;
```

*Example 2:* Assume that base table USER1.EMPTAB exists and that it contains three columns, one of which is an identity column. Declare a temporary table that has the same column names and attributes (including identity attributes) as the base table.

```
DECLARE GLOBAL TEMPORARY TABLE TEMPTAB1
 LIKE USER1.EMPTAB
 INCLUDING IDENTITY
 ON COMMIT PRESERVE ROWS;
```

In the above example, DB2 uses SESSION as the implicit qualifier for TEMPTAB1.

## DECLARE STATEMENT

The DECLARE STATEMENT statement is used for application program documentation. It declares names that are used to identify prepared SQL statements.

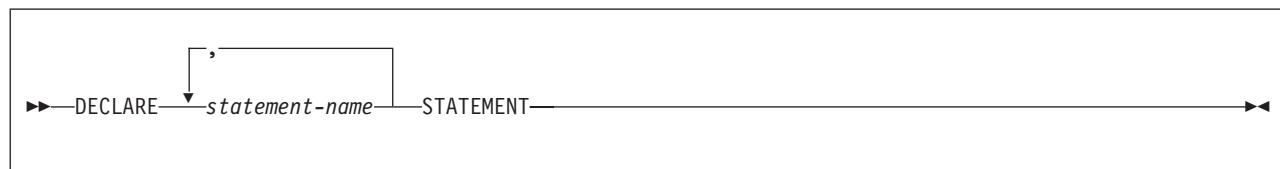
### Invocation

This statement can only be embedded in an application program. It is not an executable statement.

### Authorization

None required.

### Syntax



### Description

#### *statement-name* STATEMENT

Lists one or more names that are used in your application program to identify prepared SQL statements.

### Example

This example shows the use of the DECLARE STATEMENT statement in a PL/I program.

```

EXEC SQL DECLARE OBJECT_STATEMENT STATEMENT;

EXEC SQL INCLUDE SQLDA;
EXEC SQL DECLARE C1 CURSOR FOR OBJECT_STATEMENT;

(SOURCE_STATEMENT IS "SELECT DEPTNO, DEPTNAME,
 MGRNO FROM DSN8710.DEPT WHERE ADMRDEPT = 'A00'")

EXEC SQL PREPARE OBJECT_STATEMENT FROM SOURCE_STATEMENT;
EXEC SQL DESCRIBE OBJECT_STATEMENT INTO SQLDA;

(Examine SQLDA)

EXEC SQL OPEN C1;

DO WHILE (SQLCODE = 0);
 EXEC SQL FETCH C1 USING DESCRIPTOR SQLDA;

 (Print results)

END;

EXEC SQL CLOSE C1;

```

## DECLARE TABLE

## DECLARE TABLE

The DECLARE TABLE statement is used for application program documentation. It also provides the precompiler with information used to check your embedded SQL statements. (The DCLGEN subcommand can be used to generate declarations for tables and views described in any accessible DB2 catalog. For more on DCLGEN, see Part 2 of *DB2 Application Programming and SQL Guide* and Chapter 2 of *DB2 Command Reference*.)

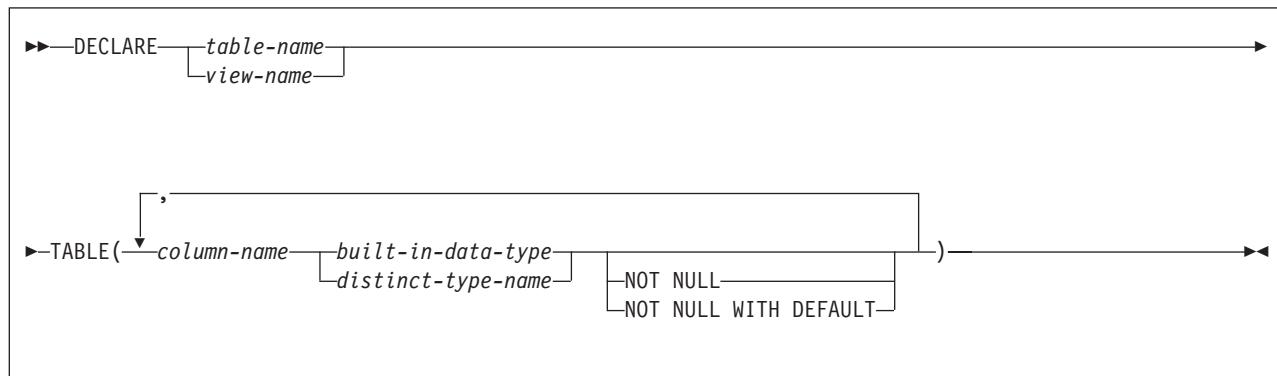
## Invocation

This statement can only be embedded in an application program. It is not an executable statement.

## Authorization

None required.

## Syntax



## Description

*table-name* or *view-name*

Is the name of the table or view you want to document. If the table is defined in your application program, the description of the table in the SQL statement in which it is defined (for example, CREATE TABLE or DECLARE GLOBAL TEMPORARY TABLE statement) and the DECLARE TABLE statement must be identical.

*column-name*

Is the name of a column of the table or view.

The precompiler uses these names to check for consistency of names within your SQL statements. It also uses the data type to check for consistency of types within your SQL statements.

## *built-in-data-type*

Is the built-in data type of the column. Use one of the built-in data types. See “built-in-data-type” on page 658 for details.

*distinct-type-name*

Is the distinct type (user-defined data type) of the column. An implicit or explicit schema name qualifies the name.

**NOT NULL**

Is used for a column that does not allow null values, and does not provide a default value.

**NOT NULL WITH DEFAULT**

Is used for a column that does not allow null values, but provides a default value.

**Notes**

**Error handling during processing:** If an error occurs during the processing of the DECLARE TABLE statement, a warning message is issued, and the precompiler continues processing your source program.

**Documenting a distinct type column:** Although you can specify the name of a distinct type as the data type of a column in the DECLARE TABLE statement, we recommend that you use the built-in data type on which the distinct type is sourced instead. Using the source type enables the precompiler to check the embedded SQL statements for errors; otherwise, error checking is deferred until bind time.

To determine the source data type of the distinct type, query column SOURCETYPE in catalog table SYSDATATYPES.

**Examples**

*Example 1:* Declare the sample employee table, DSN8710.EMP.

```
EXEC SQL DECLARE DSN8710.EMP TABLE
 (EMPNO CHAR(6) NOT NULL,
 FIRSTNME VARCHAR(12) NOT NULL,
 MIDINIT CHAR(1) NOT NULL,
 LASTNAME VARCHAR(15) NOT NULL,
 WORKDEPT CHAR(3) ,
 PHONENO CHAR(4) ,
 HIREDATE DATE ,
 JOB CHAR(8) ,
 EDLEVEL SMALLINT ,
 SEX CHAR(1) ,
 BIRTHDATE DATE ,
 SALARY DECIMAL(9,2) ,
 BONUS DECIMAL(9,2) ,
 COMM DECIMAL(9,2));
```

*Example 2:* Assume that table CANADIAN\_SALES keeps information for your company's sales in Canada. The table was created with the following definition:

```
CREATE TABLE CANADIAN_SALES
 (PRODUCT_ITEM INTEGER,
 MONTH INTEGER,
 YEAR INTEGER,
 TOTAL CANADIAN_DOLLAR);
```

CANADIAN\_DOLLAR is a distinct type that was created with the following statement:

```
CREATE DISTINCT TYPE CANADIAN_DOLLAR
 AS DECIMAL(9,2) WITH COMPARISONS;
```

Declare the CANADIAN\_SALES table, using the source type for CANADIAN\_DOLLAR instead of the distinct type name.

## DECLARE TABLE

```
DECLARE TABLE CANADIAN_SALES
 (PRODUCT_ITEM INTEGER,
 MONTH INTEGER,
 YEAR INTEGER,
 TOTAL DECIMAL(9,2);
```

## DECLARE VARIABLE

The DECLARE VARIABLE statement defines a CCSID for a host variable and the subtype of the variable. When it appears in an application program, the DECLARE VARIABLE statement causes the DB2 precompiler to tag a host variable with a specific CCSID. When the host variable appears in a SQL statement, the DB2 precompiler places this CCSID into the structures that it generates for the SQL statement.

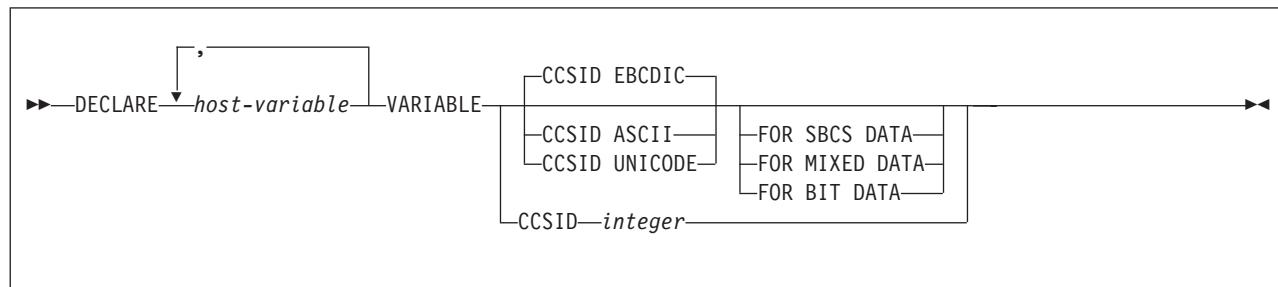
### Invocation

This statement can only be embedded in an application program. It is not an executable statement.

### Authorization

None required.

### Syntax



### Description

#### *host-variable*

Identifies a character or graphic-string host variable defined in the program. An indicator variable cannot be specified for the *host-variable*.

#### CCSID ASCII, EBCDIC, or UNICODE

Indicates that the appropriate default CCSID for the specified encoding scheme of the server should be used. If this clause is not specified, the CCSID of the variable is the appropriate default EBCDIC CCSID of the server.

#### CCSID ASCII

Indicates that the default ASCII CCSID for the type of the variable at the server should be used.

#### CCSID EBCDIC

Indicates that the default EBCDIC CCSID for the type of the variable at the server should be used. CCSID EBCDIC is the default if this option is not specified.

#### CCSID UNICODE

Indicates that the default UNICODE CCSID for the type of the variable at the server should be used.

#### FOR SBCS DATA, FOR MIXED DATA, or FOR BIT DATA

Indicates the type of data contained in the variable *host-variable*. The FOR clause cannot be specified when declaring a graphic host variable.

## DECLARE VARIABLE

For ASCII or EBCDIC data, if this clause is not specified when declaring a character host variable, the default is FOR SBCS DATA if MIXED DATA = NO on the install panel DSNTIPF. The default is FOR MIXED DATA if MIXED DATA = YES on the install panel DSNTIPF.

For UNICODE data, the default is always FOR MIXED DATA, regardless of the setting of MIXED DATA on the install panel DSNTIPF.

### FOR SBCS DATA

Indicates that the values of the host variable can contain only SBCS (single-byte character set) data.

### FOR MIXED DATA

Indicates that the values of the host variable can contain both SBCS data and DBCS data.

### FOR BIT DATA

Indicates that the values of the host-variable are not associated with a coded character set and, therefore, are never converted. The CCSID of a FOR BIT DATA host variable is 65535.

### CCSID *integer*

Indicates that the values of the host variable contain data that is encoded using CCSID *integer*. If the integer is an SBCS CCSID, the host variable is SBCS data. If the integer is a mixed data CCSID, the host variable is mixed data. For character host variables, the CCSID specified must be an SBCS, mixed CCSID, or UNICODE (UTF-8) CCSID. For graphic host variables, the CCSID specified must be a DBCS or UNICODE (UTF-16) CCSID. The valid range of values for the integer is 1 - 65533.

## Notes

**Placement of statement:** The DECLARE VARIABLE statement can be specified anywhere in an application program that SQL statements are valid with the following exception. The DECLARE VARIABLE statement must occur before an SQL statement that refers to a host variable specified in the DECLARE VARIABLE statement.

**CCSID exceptions for EXECUTE IMMEDIATE or PREPARE:** When the host variable appears in an SQL statement, the DB2 precompiler places the appropriate numeric CCSID into the structures it generates for the SQL statement. This placement of the CCSID occurs for any SQL statement other than the EXECUTE IMMEDIATE or PREPARE statements. The placement of the CCSID also occurs for a *host-variable* in an EXECUTE IMMEDIATE or PREPARE statement, but it does not occur for a variable in a *string-expression* in an EXECUTE IMMEDIATE or PREPARE statement.

If a PL/I application program contains at least one DECLARE VARIABLE statement, a *string-expression* in any EXECUTE IMMEDIATE or PREPARE statement cannot be preceded by a colon. An expression that consists of just a variable name preceded by a colon is interpreted as a *host-variable*.

**Specific host languages:** If a DECLARE VARIABLE statement is used in an assembler source program, the ONEPASS precompiler option must not be used. If a DECLARE VARIABLE statement is used in a C, C++, or PL/I source program, the TWOPASS precompiler option must be used. For those languages, or COBOL, the host-variable definition can either precede or follow a DECLARE VARIABLE

statement that refers to that variable. If a DECLARE VARIABLE statement is used in a FORTRAN source program, then the host-variable definition must precede the DECLARE VARIABLE statement.

## Example

*Example:* Define the following host variables using PL/I data types: FRED as fixed length bit data, JEAN as fixed length UTF-8 (mixed) data, DAVE as varying length UTF-8 (mixed) data, PETE as fixed length graphic UTF-16 data, and AMBER as varying length graphic UTF-16 data.

Use the DECLARE VARIABLE statement to specify a data subtype or CCSID for these host variables: FRED as CCSID EBCDIC, JEAN as CCSID 1208 or CCSID UNICODE, DAVE as CCSID 1208 or CCSID UNICODE, PETE as CCSID 1200 or CCSID UNICODE, and AMBER as CCSID 1200 or CCSID UNICODE.

```
EXEC SQL BEGIN DECLARE SECTION;
 DCL FRED CHAR(10);
 EXEC SQL DECLARE :FRED VARIABLE CCSID EBCDIC FOR BIT DATA;
 DCL JEAN CHAR(30);
 EXEC SQL DECLARE :JEAN VARIABLE CCSID 1208;
 DCL DAVE CHAR(9) VARYING;
 EXEC SQL DECLARE :DAVE VARIABLE CCSID UNICODE;
 DCL PETE GRAPHIC(10);
 EXEC SQL DECLARE :PETE VARIABLE CCSID 1200;
 DCL AMBER VARGRAPHIC(20);
 EXEC SQL DECLARE :AMBER VARIABLE CCSID UNICODE;
EXEC SQL END DECLARE SECTION;
```

## **DELETE**

---

## **DELETE**

The **DELETE** statement deletes rows from a table or view. The table or view can be at the current server or any DB2 subsystem with which the current server can establish a connection. Deleting a row from a view deletes the row from the table on which the view is based.

There are two forms of this statement:

- The *searched* **DELETE** form is used to delete one or more rows, optionally determined by a search condition.
- The *positioned* **DELETE** form is used to delete exactly one row, as determined by the current position of a cursor.

## **Invocation**

This statement can be embedded in an application program or issued interactively. A positioned **DELETE** is embedded in an application program. Both the embedded and interactive forms are executable statements that can be dynamically prepared.

## **Authorization**

Authority requirements depend on whether the object identified in the statement is a user-defined table, a catalog table, or a view, and whether the statement is a *searched* **DELETE** and SQL standard rules are in effect:

***When a user-defined table is identified:*** The privilege set must include at least one of the following:

- The **DELETE** privilege on the table
- Ownership of the table
- **DBADM** authority on the database that contains the table
- **SYSADM** authority

***When a catalog table is identified:*** The privilege set must include at least one of the following:

- **DBADM** authority on the catalog database
- **SYSCTRL** authority
- **SYSADM** authority

***When a view is identified:*** The privilege set must include at least one of the following:

- The **DELETE** privilege on the view
- **SYSADM** authority

In a *searched* delete, the **SELECT** privilege is required in addition to the **DELETE** privilege when the option for the SQL standard is set as follows:

### ***Searched DELETE and SQL standard rules:***

If SQL standard rules are in effect and the search-condition in a *searched* **DELETE** contains a reference to a column of the table or view, the privilege set must include at least one of the following:

- The **SELECT** privilege on the table or view
- **SYSADM** authority

SQL standard rules are in effect as follows:

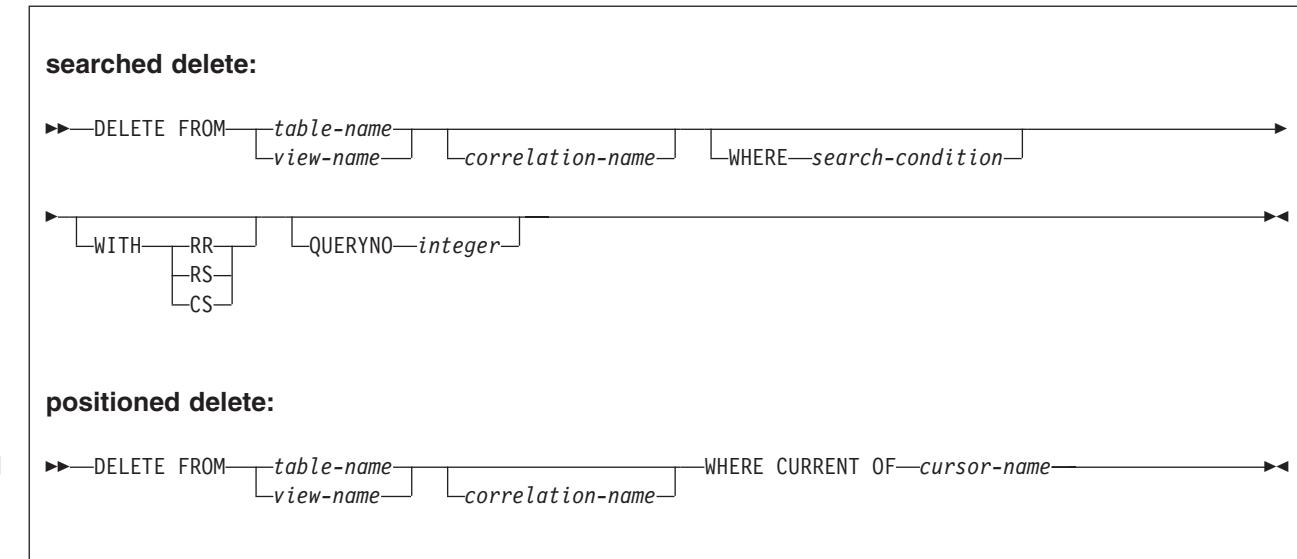
- For static SQL statements, if the **SQLRULES(STD)** bind option was specified
- For dynamic SQL statements, if the **CURRENT RULES** special register is set to '**STD**'

The owner of a view, unlike the owner of a table, might not have DELETE authority on the view (or might have DELETE authority without being able to grant it to others). The nature of the view itself can preclude its use for DELETE. For more information, see the description of authority in “CREATE VIEW” on page 711.

If a subselect is specified, the privilege set must include authority to execute the subselect. For more information about the subselect authorization rules, see “Authorization” on page 348.

If the statement is embedded in an application program, the privilege set is the privileges that are held by the authorization ID of the owner of the plan or package. If the statement is dynamically prepared, the privilege set is determined by the DYNAMICRULES behavior in effect (run, bind, define, or invoke) and is summarized in Table 40 on page 382. (For more information on these behaviors, including a list of the DYNAMICRULES bind option values that determine them, see “Authorization IDs and dynamic SQL” on page 43.)

## Syntax



## Description

**FROM** *table-name or view-name*

Identifies the object of the DELETE statement. The name must identify a table or view that exists at the DB2 subsystem identified by the implicitly or explicitly specified location name. The name must not identify:

- An auxiliary table
  - A catalog table for which deletes are not allowed
  - A view of such a catalog table
  - A read-only view (For a description of a read-only view, see “CREATE VIEW” on page 711.)

In an IMS or CICS application, the DB2 subsystem that contains the identified table or view must not be a remote DB2 Version 2 Release 3 subsystem.

*correlation-name*

Can be used within the *search-condition* or positioned DELETE to qualify references to columns of the table or view. (For an explanation of correlation names, see “Correlation names” on page 95.)

## DELETE

### WHERE

Specifies the rows to be deleted. You can omit the clause, give a search condition or name a cursor. For a created temporary table or a view of a created temporary table, you must omit the clause. When the clause is omitted, all the rows of the table or view are deleted.

#### *search-condition*

Is any search condition as described in Chapter 2, “Language elements,” on page 27. Each *column-name* in the search condition, other than in a subquery, must identify a column of the table or view.

The search condition is applied to each row of the table or view and the deleted rows are those for which the result of the search condition is true.

If the search condition contains a subquery, the subquery can be thought of as being executed each time the search condition is applied to a row, and the results used in applying the search condition. In actuality, a subquery with no correlated references is executed just once, whereas it is possible that a subquery with a correlated reference must be executed once for each row.

Let T2 denote the object table of a DELETE statement and let T1 denote a table that is referred to in the FROM clause of a subquery of that statement. T1 must not be a table that can be affected by the DELETE on T2. Thus, the following rules apply:

- T1 must not be a dependent of T2 in a relationship with a delete rule of CASCADE or SET NULL, unless the result of the subquery is materialized before the DELETE action is executed.
- T1 must not be a dependent of T3 in a relationship with a delete rule of CASCADE or SET NULL if deletes of T2 cascade to T3.

### CURRENT OF *cursor-name*

Identifies the cursor to be used in the delete operation. *cursor-name* must identify a declared cursor as explained in the description of the DECLARE CURSOR statement in “DECLARE CURSOR” on page 718. If the DELETE statement is embedded in a program, the DECLARE CURSOR statement must include *select-statement* rather than *statement-name*.

The table or view named must also be named in the FROM clause of the SELECT statement of the cursor, and the result table of the cursor must not be read-only. For an explanation of read-only result tables, see “Read-only cursors” on page 721. Note that the object of the DELETE statement must not be identified as the object of the subquery in the WHERE clause of the SELECT statement of the cursor.

If the cursor is ambiguous and the plan or package was bound with CURRENTDATA(NO), DB2 might return an error to the application if DELETE WHERE CURRENT OF is attempted for any of the following:

- A cursor that is using block fetching
- A cursor that is using query parallelism
- A cursor that is positioned on a row that has been modified by this or another application process

When the DELETE statement is executed, the cursor must be positioned on a row; that row is the one deleted. After the deletion, the cursor is positioned before the next row of its result table. If there is no next row, the cursor is positioned after the last row.

**WITH**

Specifies the isolation level used when locating the rows to be deleted by the statement.

**RR**    Repeatable read

**RS**    Read stability

**CS**    Cursor stability

The default isolation level of the statement is the isolation level of the package or plan in which the statement is bound, with the package isolation taking precedence over the plan isolation. When a package isolation is not specified, the plan isolation is the default.

**QUERYNO** *integer*

Specifies the number to be used for this SQL statement in EXPLAIN output and trace records. The number is used for the QUERYNO column of the plan table for the rows that contain information about this SQL statement. This number is also used in the QUERYNO column of the SYSIBM.SYSSTMT and SYSIBM.SYSPACKSTMT catalog tables.

If the clause is omitted, the number associated with the SQL statement is the statement number assigned during precompilation. Thus, if the application program is changed and then precompiled, that statement number might change.

Using the QUERYNO clause to assign unique numbers to the SQL statements in a program is helpful:

- For simplifying the use of optimization hints for access path selection
- For correlating SQL statement text with EXPLAIN output in the plan table

For information on using optimization hints, such as enabling the system for optimization hints and setting valid hint values, and for information on accessing the plan table, see Part 5 (Volume 2) of *DB2 Administration Guide*.

## Notes

**Delete operation errors:** If an error occurs during the execution of any delete operation, no changes are made. If an error occurs during the execution of a positioned delete, the position of the cursor is unchanged. However, it is possible for an error to make the position of the cursor invalid, in which case the cursor is closed. It is also possible for a delete operation to cause a rollback, in which case the cursor is closed.

**Position of cursor:** If an application process deletes a row on which any of its cursors are positioned, those cursors are positioned before the next row of the result table. Let C be a cursor that is positioned before row R (as a result of an OPEN, a DELETE through C, a DELETE through some other cursor, or a searched DELETE). In the presence of INSERT, UPDATE, and DELETE operations that affect the base table from which R is derived, the next FETCH operation referencing C does not necessarily position C on R. For example, the operation can position C on R', where R' is a new row that is now the next row of the result table.

**Locking:** Unless appropriate locks already exist, one or more exclusive locks are acquired during the execution of a successful delete operation. Until the locks are released by a commit or rollback operation, the effect of the DELETE operation can only be perceived by the application process that performed the deletion and the locks can prevent other application processes from performing operations on the table. Locks are not acquired when rows are deleted from a declared temporary table unless all the rows are deleted (DELETE FROM T). When all the rows are

## DELETE

deleted from a declared temporary table, a segmented table lock is acquired on the pages for the table and no other table in the table space is affected.

**Referential integrity:** If the object table of the delete operation is a parent table:

- The rows selected for deletion must have no dependents in a relationship governed by a delete rule of RESTRICT or NO ACTION.
- The delete operation must not cascade to descendent rows that are dependents in a relationship governed by a delete rule of RESTRICT or NO ACTION.

If the delete operation is not prevented by a RESTRICT or NO ACTION delete rule, the selected rows are deleted and:

- The columns of foreign keys in any rows that are their dependents in a relationship governed by a delete rule of SET NULL and which allow nulls are set to the null value.
- Any rows that are their dependents in a relationship governed by a delete rule of CASCADE are also deleted, and these rules apply, in turn, to those rows.

The only difference between NO ACTION and RESTRICT is when the referential constraint is enforced. RESTRICT (IBM SQL rules) enforces the rule immediately, and NO ACTION (SQL standard rules) enforces the rule at the end of the statement. This difference matters only in the case of a searched DELETE involving a self-referencing constraint that deletes more than one row. NO ACTION might allow the DELETE to be successful where RESTRICT (if it were allowed) would prevent it.

A check constraint can prevent the deletion of a row in a parent table when there are dependents in a relationship with a delete rule of SET NULL. If deleting a row in the parent table would cause a column in a dependent table to be set to null and there is a check constraint that specifies that the column must not be null, the row is not deleted.

**Nesting user-defined functions or stored procedures:** A DELETE statement can implicitly or explicitly refer to user-defined functions or stored procedures. This is known as *nesting* of SQL statements. A user-defined function or stored procedure that is nested within the DELETE must not access the table from which you are deleting rows.

**Triggers:** If the identified table or the base table of the identified view has a delete trigger, the trigger is fired for each row deleted.

**Number of rows deleted:** Except as noted below, a DELETE operation sets SQLERRD(3) to the number of deleted rows. This number does not include any rows that were deleted as a result of a CASCADE delete rule.

**DELETE FROM T without a WHERE clause** deletes all rows of T. If a table T is contained in a segmented table space and is not a parent table, this deletion will be performed without accessing T. If a table T is from a CREATE GLOBAL TEMPORARY TABLE statement, the SQLERRD(3) field is set to -1. (For a complete description of the SQLCA, including exceptions to the above, see "SQL communication area (SQLCA)" on page 979.)

If the object table is SYSIBM.SYSSTRINGS, the rows selected for delete must be rows provided by the user (the value of the IBMREQD column is N).

**Rules for positioned DELETE with SENSITIVE STATIC scrollable cursor:** When a SENSITIVE STATIC scrollable cursor has been declared, the following rules apply:

- *Delete attempt of delete holes or update holes.* If, with a positioned delete against a SENSITIVE STATIC scrollable cursor, an attempt is made to delete a row that has been identified as a delete hole (that is, a row in the result table whose corresponding row has been deleted from the base table), an error occurs.  
If an attempt is made to delete a row that has been identified as an update hole (that is, a row in the result table whose corresponding row has been updated so that it no longer satisfies the predicate of the SELECT statement), an error occurs.
- *Delete operations.* Positioned delete operations with SENSITIVE STATIC scrollable cursors perform as follows:
  1. The SELECT list items in the target row of the base table of the cursor are compared with the values in the corresponding row of the result table (that is, the result table must still agree with the base table). If the values are not identical, the delete operation is rejected and an error occurs. The operation can be attempted again after a successful FETCH SENSITIVE has occurred for the target row.
  2. The WHERE clause of the SELECT statement is re-evaluated to determine whether the current values in the base table still satisfy the search criteria. The values in the SELECT list are compared to determine that these values have not changed. If the WHERE clause evaluates as true, and the values in the SELECT list have not changed, the delete operation is allowed to proceed. Otherwise, an error occurs, the delete operation is rejected, and an update hole appears in the cursor.
  3. After the base table row is successfully deleted, the temporary result table is updated and the row is marked as a delete hole.
- *Rollback of delete holes.* Delete holes are usually permanent. Once a delete hole is identified, it remains a delete hole until the cursor is closed. However, if a positioned delete using *this* cursor actually caused the creation of the hole (that is, this cursor was used to make the changes that resulted in the hole) and the delete was subsequently rolled back, then the row is no longer considered a delete hole.
- *Result table.* Any deletes, either positioned or searched, to rows of the base table on which a SENSITIVE STATIC scrollable cursor is defined are reflected in the result table if a positioned update or positioned delete is attempted with the scrollable cursor. A SENSITIVE STATIC scrollable cursor sees these deletes when a FETCH SENSITIVE is attempted.

## Examples

Assume that the statements in these examples are embedded in PL/I programs.

*Example 1:* From the table DSN8710.EMP delete the row on which the cursor C1 is currently positioned.

```
EXEC SQL DELETE FROM DSN8710.EMP WHERE CURRENT OF C1;
```

*Example 2:* From the table DSN8710.EMP, delete all rows for departments E11 and D21.

```
EXEC SQL DELETE FROM DSN8710.EMP
 WHERE WORKDEPT = 'E11' OR WORKDEPT = 'D21';
```

## **DELETE**

|      *Example 3:* From employee table X, delete the employees who have the most  
|      absences for the department to which they belong.

```
| EXEC SQL DELETE FROM EMP X
| WHERE ABSENT = (SELECT MAX(ABSENT) FROM EMP Y
| WHERE X.WORKDEPT = Y.WORKDEPT);
|
```

## DESCRIBE (prepared statement or table)

The DESCRIBE statement obtains information about a prepared statement or a designated table or view. For an explanation of prepared statements, see “PREPARE” on page 846 and “DESCRIBE PROCEDURE” on page 760.

### Invocation

This statement can only be embedded in an application program. It is an executable statement that cannot be dynamically prepared.

### Authorization

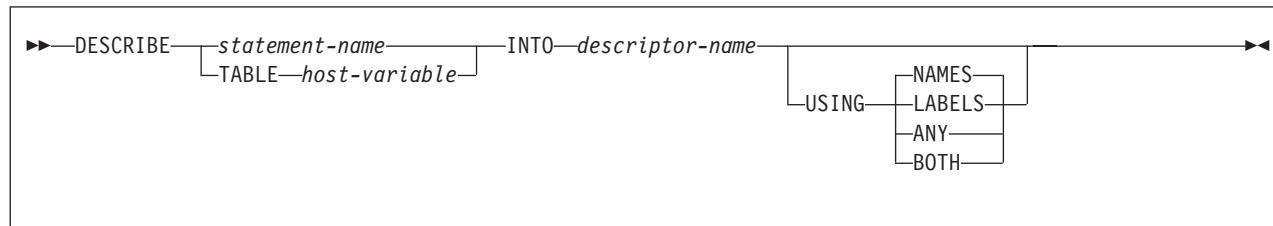
None required if the statement is used for a prepared statement. When it is used instead for a table or view, the privileges that are held by the authorization ID that owns the plan or package must include at least one of the following (if there is a plan, authorization checking is done only against the plan owner):

- Ownership of the table or view
- The SELECT, INSERT, UPDATE, DELETE, or REFERENCES privilege on the object
- The ALTER or INDEX privilege on the object (tables only)
- DBADM authority over the database that contains the object (tables only)
- SYSADM or SYSCTRL authority

For an RRSAF application that does not have a plan and in which the requester and the server are DB2 for OS/390 and z/OS systems, authorization to execute the package is performed against the primary or secondary authorization ID of the process.

See “PREPARE” on page 846 for the authorization required to create a prepared statement.

### Syntax



### Description

#### *statement-name*

Identifies the prepared statement. When the DESCRIBE statement is executed, the name must identify a statement that has been prepared by the application process at the current server.

#### **TABLE** *host-variable*

Identifies the table or view. The name must not identify an auxiliary table. When the DESCRIBE statement is executed, the host variable must contain a name which identifies a table or view that exists at the current server. This variable must be a fixed- or varying-length character string with a length attribute less than 256. The name must be followed by one or more blanks if the length of the name is less than the length of the variable. It cannot contain a period as the first character and it cannot contain embedded blanks. In addition, the quotation

## DESCRIBE

mark is the escape character regardless of the value of the string delimiter option. An indicator variable must not be specified for the host variable.

### **INTO** *descriptor-name*

Identifies an SQL descriptor area (SQLDA), which is described in Appendix C, “SQLCA and SQLDA,” on page 979. See “Identifying an SQLDA in C or C++” on page 1003 for how to represent *descriptor-name* in C.

*For languages other than REXX:* Before the DESCRIBE statement is executed, the user must set the following variable in the SQLDA and the SQLDA must be allocated.

### **SQLN** Indicates the number of SQLVAR occurrences provided in the SQLDA.

DB2 does not change this value. For techniques to determine the number of required occurrences, see “Allocating the SQLDA” on page 751.

*For REXX:* The SQLDA is not allocated before it is used. An SQLDA consists of a set of stem variables. There is one occurrence of variable *stem.SQLD*, followed by zero or more occurrences of a set of variables that is equivalent to an SQLVAR structure. Those variables begin with *stem.n*.

After the DESCRIBE statement is executed, all the fields in the SQLDA except SQLN are either set by DB2 or ignored. For information on the contents of the fields, see “The SQLDA contents returned after DESCRIBE” on page 752.

### **USING**

Indicates what value to assign to each SQLNAME variable in the SQLDA. If the requested value does not exist, SQLNAME is set to a length of 0.

### **NAMES**

Assigns the name of the column. This is the default.

### **LABELS**

Assigns the label of the column. (Column labels are defined by the LABEL ON statement.)

### **ANY**

Assigns the column label, and if the column has no label, the column name.

### **BOTH**

Assigns both the label and name of the column. In this case, two or three occurrences of SQLVAR per column, depending on whether the result set contains distinct types, are needed to accommodate the additional information. To specify this expansion of the SQLVAR array, set SQLN to  $2xn$  or  $3xn$ , where  $n$  is the number of columns in the object being described. For each of the columns, the first  $n$  occurrences of SQLVAR, which are the base SQLVAR entries, contain the column names. Either the second or third  $n$  occurrences of SQLVAR, which are the extended SQLVAR entries, contain the column labels. If there are no distinct types, the labels are returned in the second set of SQLVAR entries. Otherwise, the labels are returned in the third set of SQLVAR entries.

For a declared temporary table, the name of the column is assigned regardless of the value specified in the USING clause because declared temporary tables cannot have labels.

## Notes

Information about a prepared statement can also be obtained by using the INTO clause of the PREPARE statement.

**Allocating the SQLDA:** Before the DESCRIBE or PREPARE INTO statement is executed, the value of SQLN must be set to a value greater than or equal to zero to indicate how many occurrences of SQLVAR are provided in the SQLDA. Also, enough storage must be allocated to contain the number of occurrences that SQLN specifies. To obtain the description of the columns of the result table of a prepared SELECT statement, the number of occurrences of SQLVAR must be at least equal to the number of columns. Furthermore, if USING BOTH is specified, or if the columns include LOBs or distinct types, the number of occurrences of SQLVAR should be two or three times the number of columns. See “Determining how many SQLVAR occurrences are needed” on page 991 for more information.

*First technique:* Allocate an SQLDA with enough occurrences of SQLVAR to accommodate any select list that the application will have to process. At the extreme, the number of SQLVARs could equal three times the maximum number of columns allowed in a result table. After the SQLDA is allocated, the application can use the SQLDA repeatedly.

This technique uses a large amount of storage that is never deallocated, even when most of this storage is not used for a particular select list.

*Second technique:* Repeat the following two steps for every processed select list:

1. Execute a DESCRIBE statement with an SQLDA that has no occurrences of SQLVAR; that is, an SQLDA for which SQLN is zero.
2. Allocate a new SQLDA with enough occurrences of SQLVAR. Use the values that are returned in SQLD and SQLCODE to determine the number of SQLVAR entries that are needed. The value of SQLD is the number of columns in the result table, which is either the required number of occurrences of SQLVAR or a fraction of the required number (see “Determining how many SQLVAR occurrences are needed” on page 991 for details). If the SQLCODE is +236, +237, +238, or +239, the number of SQLVAR entries that is needed is two or three times the value in SQLD, depending on whether USING BOTH was specified. Set SQLN to reflect the number of SQLVAR entries that have been allocated.
3. Execute the DESCRIBE statement again, using the new SQLDA.

This technique allows better storage management than the first technique, but it doubles the number of DESCRIBE statements.

*Third technique:* Allocate an SQLDA that is large enough to handle most (hopefully, all) select lists but is also reasonably small. If an execution of DESCRIBE fails because SQLDA is too small, allocate a larger SQLDA and execute the DESCRIBE statement again.

For the new larger SQLDA, use the values that are returned in SQLD and SQLCODE from the failing DESCRIBE statement to calculate the number of occurrences of SQLVAR that are needed, as described in technique two.

Remember to check for SQLCODEs +236, +237, +238, and +239, which indicate whether *extended* SQLVAR entries are needed because the data includes LOBs or distinct types.

## DESCRIBE

This third technique is a compromise between the first two techniques. Its effectiveness depends on a good choice of size for the original SQLDA.

**The SQLDA contents returned on DESCRIBE:** After a DESCRIBE statement is executed, the following list describes the contents of the SQLDA fields as they are set by DB2 or ignored. These descriptions do not necessarily apply to the uses of an SQLDA in other SQL statements (EXECUTE, OPEN, FETCH). For more on the other uses, see Appendix C, “SQLCA and SQLDA,” on page 979.

### SQLDAID

DB2 sets the first 6 bytes to 'SQLDA' (5 letters followed by the space character) and the eighth byte to a space character. The seventh byte is set to indicate the number of SQLVAR entries that are needed to describe each column of the result table as follows:

**space** The value of space occurs when:

- USING BOTH was not specified and the columns being described do not include LOBs or distinct types. Each column only needs one SQLVAR entry. If the SQL standard option is yes, DB2 sets SQLCODE to warning code +236. Otherwise, SQLCODE is zero.
- USING BOTH was specified and the columns being described do not include LOBs or distinct types. Each column needs two SQLVAR entries. DB2 sets SQLD to two times the number of columns of the result table. The second set of SQLVARs is used for the labels.

**2** Each column needs two SQLVAR entries. Two entries per column are required when:

- USING BOTH was not specified and the columns being described include LOBs or distinct types or both. DB2 sets the second set of SQLVAR entries with information for the LOBs or distinct types being described.
- USING BOTH was specified and the columns include LOBs but not distinct types. DB2 sets the second set of SQLVAR entries with information for the LOBs and labels for the columns being described.

**3** Each column needs three SQLVAR entries. Three entries are required only when USING BOTH is specified and the columns being described include distinct types. The presence of LOB data does not matter. It is the distinct types and not the LOBs that cause the need for three SQLVAR entries per column when labels are also requested. DB2 sets the second set of SQLVAR entries with information for the distinct types (and LOBs, if any) and the third set of SQLVAR entries with the labels of the columns being described.

A REXX SQLDA does not contain this field.

### SQLDABC

The length of the SQLDA in bytes. DB2 sets the value to SQLNx44+16.

A REXX SQLDA does not contain this field.

**SQLD** If the prepared statement is a query, DB2 sets the value to the number of columns in the object being described (the value is actually twice the number of columns in the case where USING BOTH was specified and the result table does not include LOBs or distinct types). Otherwise, if the statement is not a query, DB2 sets the value to 0.

**SQLVAR**

An array of field description information for the column being described. There are two types of SQLVAR entries—the base SQLVAR and the extended SQLVAR.

If the value of SQLD is 0, or is greater than the value of SQLN, no values are assigned to any occurrences of SQLVAR. If the value of SQLN was set so that there are enough SQLVAR occurrences to describe the specified columns (columns with LOBs or distinct types and a request for labels increase the number of SQLVAR entries that are needed), the values are assigned to the first *n* occurrences of SQLVAR so that the first occurrence of SQLVAR contains a description of the first column, the second occurrence of SQLVAR contains a description of the second column, and so on. This first set of SQLVAR entries are referred to as *base SQLVAR* entries. Each column always has a base SQLVAR entry.

If the DESCRIBE statement included the USING BOTH clause, or the columns being described include LOBs or distinct types, additional SQLVAR entries are needed. These additional SQLVAR entries are referred to as the *extended SQLVAR* entries. There can be up to two sets of extended SQLVAR entries for each column.

For REXX, the SQLVAR is a set of stem variables that begin with *stem.n*, instead of a structure. The REXX SQLDA uses only a base SQLVAR. The way in which DB2 assigns values to the SQLVAR variables is the same as for other languages. That is, the *stem.1* variables describe the first column in the result table, the *stem.2* variables describe the second column in the result table, and so on. If USING BOTH is specified, the *stem.n+1* variables also describe the first column in the result table, the *stem.n+2* variables also describe the second column in the result table, and so on.

*The base SQLVAR:*

**SQLTYPE**

A code that indicates the data type of the column and whether the column can contain null values. For the possible values of SQLTYPE, see Table 80 on page 996.

**SQLLEN**

A length value depending on the data type of the result columns. SQLLEN is 0 for LOB data types. For the other possible values of SQLLEN, see Table 80 on page 996.

In a REXX SQLDA, for DECIMAL or NUMERIC columns, DB2 sets the SQLPRECISION and SQLSCALE fields instead of the SQLLEN field.

**SQLDATA**

The CCSID of a string column. For possible values, see Table 81 on page 997.

In a REXX SQLDA, DB2 sets the SQLCCSID field instead of the SQLDATA field.

**SQLIND**

Reserved.

**SQLNAME**

The unqualified name or label of the column, depending on the value of USING (NAMES, LABELS, ANY, or BOTH). The field is a string of length 0 if the column does not have a name or label. For

## DESCRIBE

more details on unnamed columns, see the discussion of the names of result columns under “select-clause” on page 349. This value is returned in the encoding scheme specified by the ENCODING bind option for the plan or package that contains the statement.

*The extended SQLVAR:*

### **SQLLONGLEN**

The length attribute of a BLOB, CLOB, or DBCLOB column.

- \* Reserved.

### **SQLDATALEN**

Not Used.

### **SQLDATATYPE-NAME**

For a distinct type, the fully qualified distinct type name. Otherwise, the value is the fully qualified name of the built-in data type.

For a label, the label for the column.

This value is returned in the encoding scheme specified by the ENCODING bind option for the plan or package that contains this statement.

The REXX SQLDA does not use the extended SQLVAR.

**Performance considerations:** Although DB2 does not change the value of SQLN, you might want to reset this value after the DESCRIBE statement is executed. If the contents of SQLDA from the DESCRIBE statement is used in a later FETCH statement, set SQLN to *n* (where *n* is the number of columns of the result table) before executing the FETCH statement. For details, see “Preparing the SQLDA for data retrieval.”

**Preparing the SQLDA for data retrievals:** This note is relevant if you are applying DESCRIBE to a prepared query and you intend to use the SQLDA in the FETCH statements you employ to retrieve the result table rows. To prepare the SQLDA for that task, you must set the SQLDATA field of SQLVAR. SQLIND must be set if SQLTYPE is odd, and SQLNAME must be set when overriding the CCSID. For the meaning of those fields in that context, see “SQL descriptor area (SQLDA)” on page 986.

Also, SQLN and SQLDABC should be reset (if necessary) to *n* and *n*×44+16, where *n* is the number of columns in the result table. Doing so can improve performance when the rows of the result table are fetched.

**Support for extended dynamic SQL in a distributed environment:** In a distributed environment where DB2 for OS/390 and z/OS is the server and the requester supports extended dynamic SQL, such as DB2 Server for VSE & VM, a DESCRIBE statement that is executed against an SQL statement in the extended dynamic package appears to DB2 as a DESCRIBE statement against a static SQL statement in the DB2 package. A DESCRIBE statement cannot normally be issued against a static SQL statement. However, a DESCRIBE against a static SQL statement that is generated by extended dynamic SQL executes without error if the package has been rebound after field DESCRIBE FOR STATIC on installation panel DSNTIPF has been set to YES.

YES indicates that DB2 generates an SQLDA for the DESCRIBE at bind time so that DESCRIBE requests for static SQL statements can be satisfied at execution time. For more information, see Part 3 of *DB2 Installation Guide*.

**Avoiding double preparation when using REOPTVAR:** If bind option REOPT(VARS) is in effect, DESCRIBE causes the statement to be prepared if it is not already prepared. If issued before an OPEN or an EXECUTE, the DESCRIBE causes the statement to be prepared without input variable values. If the statement has input variable values, it must then be prepared again when it is opened or executed. To avoid preparing statements twice, issue the DESCRIBE after the OPEN. For non-cursor statements, open and fetch processing are performed on the EXECUTE. So, if a DESCRIBE must be issued, the statement will be prepared twice.

**Errors occurring on DESCRIBE:** In local and remote processing, the DEFER(PREPARE) and REOPT(VARS) bind options can cause some errors that are normally issued during PREPARE processing to be issued on DESCRIBE.

**Using host variables:** If the DESCRIBE statement contains host variables, the contents of the host variables are assumed to be in the encoding scheme that was specified in the ENCODING parameter when the package or plan that contains the statement was bound.

## Example

In a PL/I program, execute a DESCRIBE statement with an SQLDA that has no occurrences of SQLVAR. If SQLD is greater than zero, use the value to allocate an SQLDA with the necessary number of occurrences of SQLVAR and then execute a DESCRIBE statement using that SQLDA. This is the second technique described in “Allocating the SQLDA” on page 751.

```

EXEC SQL BEGIN DECLARE SECTION;
 DCL STMT1_STR CHAR(200) VARYING;
EXEC SQL END DECLARE SECTION;
EXEC SQL INCLUDE SQLDA;
EXEC SQL DECLARE DYN_CURSOR CURSOR FOR STMT1_NAME;

... /* code to prompt user for a query, then to generate */
 /* a select-statement in the STMT1_STR */
EXEC SQL PREPARE STMT1_NAME FROM :STMT1_STR;

... /* code to set SQLN to zero and to allocate the SQLDA */
EXEC SQL DESCRIBE STMT1_NAME INTO :SQLDA;

... /* code to check that SQLD is greater than zero, to set */
 /* SQLN to SQLD, then to re-allocate the SQLDA */
EXEC SQL DESCRIBE STMT1_NAME INTO :SQLDA;

... /* code to prepare for the use of the SQLDA */
EXEC SQL OPEN DYN_CURSOR;

... /* loop to fetch rows from result table */
EXEC SQL FETCH DYN_CURSOR USING DESCRIPTOR :SQLDA;
.
.
.
```

## DESCRIBE CURSOR

### DESCRIBE CURSOR

The DESCRIBE CURSOR statement gets information about the result set that is associated with the cursor. The information, such as column information, is put into a descriptor. Use DESCRIBE CURSOR for result set cursors from stored procedures. The cursor must be defined with the ALLOCATE CURSOR statement.

### Invocation

This statement can only be embedded in an application program. It is an executable statement that cannot be dynamically prepared.

### Authorization

None required.

### Syntax

```
►►—DESCRIBE CURSOR—cursor-name—INTO—descriptor-name—►►
 |host-variable|
```

### Description

#### *cursor-name* or *host-variable*

Identifies a cursor by the specified cursor-name or the cursor name contained in the host-variable. The name must identify a cursor that has already been allocated in the source program.

A cursor name is a long identifier.

If a host variable is used:

- It must be a character string variable with a length attribute that is not greater than 18 bytes (A C NUL-terminated character string can be up to 19 bytes).
- It must not be followed by an indicator variable.
- The cursor name must be left justified within the host variable and must not contain embedded blanks.
- If the length of the cursor name is less than the length of the host variable, it must be padded on the right with blanks.

#### **INTO** *descriptor-name*

Identifies an SQL descriptor area (SQLDA). The information returned in the SQLDA describes the columns in the result set associated with the named cursor.

The considerations for allocating and initializing the SQLDA are similar to those of a varying-list SELECT statement. For more information, see Part 6 of *DB2 Application Programming and SQL Guide*.

| *For REXX:* The SQLDA is not allocated before it is used.

After the DESCRIBE CURSOR statement is executed, the contents of the SQLDA are the same as after a DESCRIBE for a SELECT statement, with the following exceptions:

- The first 5 bytes of the SQLDAID field are set to 'SQLRS'.

- Bytes 6 to 8 of the SQLDAID field are reserved. If the cursor is declared WITH HOLD in a stored procedure, the high-order bit of the 8th byte is set to 1.

These exceptions do not apply to a REXX SQLDA, which does not include the SQLDAID field.

## Notes

Column names are included in the information that DESCRIBE CURSOR obtains when the statement that generates the result set is either:

- Dynamic
- Static and the value of field DESCRIBE FOR STATIC on installation panel DSNTIPF was YES when the package or stored procedure was bound. If the value of the field was NO, the returned information includes only the data type and length of the columns.

**Using host variables:** If the DESCRIBE CURSOR statement contains host variables, the contents of the host variables are assumed to be in the encoding scheme that was specified in the ENCODING parameter when the package or plan that contains the statement was bound.

## Examples

The statements in the following examples are assumed to be in PL/I programs.

*Example 1:* Place information about the result set associated with cursor C1 into the descriptor named by :sqlda1.

```
EXEC SQL DESCRIBE CURSOR C1 INTO :sqlda1
```

*Example 2:* Place information about the result set associated with the cursor named by :hv1 into the descriptor named by :sqlda2.

```
EXEC SQL DESCRIBE CURSOR :hv1 INTO :sqlda2
```

## DESCRIBE INPUT

The DESCRIBE INPUT statement obtains information about the input parameter markers of a prepared statement. For an explanation of prepared statements, see “PREPARE” on page 846 and “DESCRIBE PROCEDURE” on page 760.

### Invocation

This statement can only be embedded in an application program. It is an executable statement that cannot be dynamically prepared.

### Authorization

None required if the statement is used for a prepared statement.

### Syntax

```
►►—DESCRIBE INPUT—statement-name—INTO—descriptor-name—►►
```

### Description

#### *statement-name*

Identifies the prepared statement. When the DESCRIBE INPUT statement is executed, the name must identify a statement that has been prepared by the application process at the current server.

#### **INTO** *descriptor-name*

Identifies an SQL descriptor area (SQLDA), which is described in Appendix C, “SQLCA and SQLDA,” on page 979. See “Identifying an SQLDA in C or C++” on page 1003 for how to represent *descriptor-name* in C. The information returned in the SQLDA describes the parameter markers.

Before the DESCRIBE INPUT statement is executed, the user must set the SQLN field in the SQLDA and the SQLDA must be allocated. Considerations for initializing and allocating the SQLDA are similar to those for the DESCRIBE statement (see “DESCRIBE (prepared statement or table)” on page 749). An occurrence of an extended SQLVAR is needed for each parameter in addition to the required base SQLVAR only if the input data contains LOBs.

*For REXX:* The SQLDA is not allocated before it is used.

After the DESCRIBE INPUT statement is executed, all the fields in the SQLDA except SQLN are either set by DB2 or ignored. The SQLDA contents are similar to the contents returned for the DESCRIBE statement (see page 752) with these exceptions:

- In the SQLDAID, DB2 sets the value of the seventh byte only to the space character or '2'. A value of '3' is never used. The value '2' indicates that two SQLVAR entries (an occurrence of both a base SQLVAR and an extended SQLVAR) are required for each parameter because the input data contains LOBs. The seventh byte is a space character when either of the following conditions is true:
  - The input data does not contain LOBs. Only a base SQLVAR occurrence is needed for each parameter.
  - Only a base SQLVAR occurrence is needed for each column of the result, and the SQLDA is not large enough to contain the returned information.

- The SQLD field is set to the number of parameter markers being described. The value is 0 if the statement being described does not have input parameter markers.
- The SQLNAME field is not used.
- The SQLDATATYPE-NAME is not used if an extended SQLVAR entry is present. DESCRIBE INPUT does not return information about distinct types.

For complete information on the contents of the fields, see “SQL descriptor area (SQLDA)” on page 986.

## Notes

**Preparing the SQLDA for OPEN or EXECUTE:** This note is relevant if you are applying DESCRIBE INPUT to a prepared statement and you intend to use the SQLDA in an OPEN or EXECUTE statement. To prepare the SQLDA for that purpose:

- Set SQLDATA to a valid address.
- If SQLTYPE is odd, set SQLIND to a valid address.

For the meaning of those fields in that context, see “SQL descriptor area (SQLDA)” on page 986.

**Support for extended dynamic SQL in a distributed environment:** Unlike the DESCRIBE statement, which can be used in a distributed environment to describe static SQL statements generated by extended dynamic SQL, you cannot describe host variables in static SQL statements that are generated by extended dynamic SQL. A DESCRIBE INPUT statement issued against such static SQL statements always fails.

For information on how the DESCRIBE statement supports extended dynamic SQL, see “Support for extended dynamic SQL in a distributed environment” on page 754.

**Using host variables:** If the DESCRIBE INPUT statement contains host variables, the contents of the host variables are assumed to be in the encoding scheme that was specified in the ENCODING parameter when the package or plan that contains the statement was bound.

## Example

Execute a DESCRIBE INPUT statement with an SQLDA that has enough SQLVAR occurrences to describe any number of input parameters a prepared statement might have. Assume that five parameter markers at most will need to be described and that the input data does not contain LOBs.

```
/* STMT1_STR contains INSERT statement with VALUES clause */
EXEC SQL PREPARE STMT1_NAME FROM :STMT1_STR;

... /* code to set SQLN to 5 and to allocate the SQLDA */ */
EXEC SQL DESCRIBE INPUT STMT1_NAME INTO :SQLDA;
.
.
```

This example uses the first technique described in “Allocating the SQLDA” on page 751 to allocate the SQLDA.

## DESCRIBE PROCEDURE

# DESCRIBE PROCEDURE

The DESCRIBE PROCEDURE statement gets information about the result sets returned by a stored procedure. The information, such as the number of result sets, is put into a descriptor.

## Invocation

This statement can only be embedded in an application program. It is an executable statement that cannot be dynamically prepared.

## Authorization

None required.

## Syntax

```
►—DESCRIBE PROCEDURE—procedure-name—INTO—descriptor-name—►
 └host-variable┘
```

## Description

### *procedure-name* or *host-variable*

Identifies the stored procedure to describe by the specified procedure name or the procedure name contained in the host variable.

A procedure name is a qualified or unqualified name. Each part of the name must be composed of SBCS characters:

- A fully qualified procedure name is a three-part name. The first part is a long identifier that contains the location name that identifies the DBMS at which the procedure is stored. The second part is a short identifier that contains the schema name of the stored procedure. The last part is a long identifier that contains the name of the stored procedure. A period must separate each of the parts. Any or all of the parts can be a delimited identifier.
- A two-part procedure name has one implicit qualifier. The implicit qualifier is the location name of the current server. The two parts identify the schema name and the name of the stored procedure. A period must separate the two parts.
- An unqualified procedure name is a one-part name with one implicit qualifier. The implicit qualifier is the location name of the current server. An implicit schema name is not needed as a qualifier. Successful execution of the ASSOCIATE LOCATOR statement only requires that the unqualified procedure name in the statement is the same as the procedure name in the most recently executed CALL statement that was specified with an unqualified procedure name. (The implicit schema name for the unqualified name in the CALL statement is not considered in the match.) The rules for how the procedure name must be specified are described below.

If a host variable is used:

- It must be a character string variable with a length attribute that is not greater than 254.
- It must not be followed by an indicator variable.
- The value of the host variable is a specification that depends on the database server. Regardless of the server, the specification must:

- Be left justified within the host variable
- Not contain embedded blanks
- Be padded on the right with blanks if its length is less than that of the host variable

When the DESCRIBE PROCEDURE statement is executed, the procedure name or specification must identify a stored procedure that the requester has already invoked using the CALL statement.

The procedure name in the DESCRIBE PROCEDURE statement must be specified the same way that it was specified on the CALL statement. For example, if a two-part name was specified on the CALL statement, you must use a two-part name in the DESCRIBE PROCEDURE statement. However, there is one condition under which the names do not have to match. If the CALL statement was made with a three-part name and the current server is the same as the location in the three-part name, you can omit the location name and specify a two-part name.

#### **INTO *descriptor-name***

Identifies an SQL descriptor area (SQLDA). The information returned in the SQLDA describes the result sets returned by the stored procedure.

Considerations for allocating and initializing the SQLDA are similar to those for DESCRIBE TABLE.

The contents of the SQLDA after executing a DESCRIBE PROCEDURE statement are:

- The first 5 bytes of the SQLDAID field are set to 'SQLPR'.  
A REXX SQLDA does not contain SQLDAID.
- Bytes 6 to 8 of the SQLDAID field are reserved.
- The SQLD field is set to the total number of result sets. A value of 0 in the field indicates there are no result sets.
- There is one SQLVAR entry for each result set.
- The SQLDATA field of each SQLVAR entry is set to the result set locator value associated with the result set.  
For a REXX SQLDA, SQLLOCATOR is set to the result set locator value.
- The SQLIND field of each SQLVAR entry is set to the estimated number of rows in the result set
- The SQLNAME field is set to the name of the cursor used by the stored procedure to return the result set. This value is returned in the encoding scheme specified by the ENCODING bind option for the plan or package that contains this statement.

## Notes

A value of -1 in the SQLIND field indicates that an estimated number of rows in the result set is not provided. DB2 for OS/390 and z/OS always sets SQLIND to -1.

DESCRIBE PROCEDURE does not return information about the parameters expected by the stored procedure.

If the DESCRIBE PROCEDURE statement contains host variables, the contents of the host variables are assumed to be in the encoding scheme that was specified in the ENCODING parameter when the package or plan that contains the statement was bound.

## DESCRIBE PROCEDURE

### Examples

The statements in the following examples are assumed to be in PL/I programs.

*Example 1:* Place information about the result sets returned by stored procedure P1 into the descriptor named by SQLDA1. Assume that the stored procedure is called with a one-part name from current server SITE2.

```
EXEC SQL CONNECT TO SITE2;
EXEC SQL CALL P1;
EXEC SQL DESCRIBE PROCEDURE P1 INTO :SQLDA1;
```

*Example 2:* Repeat the scenario in Example 1, but use a two-part name to specify an explicit schema name for the stored procedure to ensure that stored procedure P1 in schema MYSCHHEMA is used.

```
EXEC SQL CONNECT TO SITE2;
EXEC SQL CALL MYSCHHEMA.P1;
EXEC SQL DESCRIBE PROCEDURE MYSCHHEMA.P1 INTO :SQLDA1;
```

*Example 3:* Place information about the result sets returned by the stored procedure identified by host variable HV1 into the descriptor named by SQLDA2. Assume that host variable HV1 contains the value SITE2.MYSCHHEMA.P1 and the stored procedure is called with a three-part name.

```
EXEC SQL CALL SITE2.MYSCHHEMA.P1;
EXEC SQL DESCRIBE PROCEDURE :HV1 INTO :SQLDA2;
```

The preceding example would be invalid if host variable HV1 had contained the value MYSCHHEMA.P1, a two-part name. For the example to be valid with that two-part name in host variable HV1, the current server must be the same as the location name that is specified on the CALL statement as the following statements demonstrate. This is the only condition under which the names do not have to be specified the same way and a three-part name on the CALL statement can be used with a two-part name on the DESCRIBE PROCEDURES statement.

```
EXEC SQL CONNECT TO SITE2;
EXEC SQL CALL SITE2.MYSCHHEMA.P1;
EXEC SQL ASSOCIATE LOCATORS (:LOC1, :LOC2)
 WITH PROCEDURE :HV1;
```

---

## DROP

The DROP statement deletes an object at the current server. Except for storage groups, any objects that are directly or indirectly dependent on that object are deleted. Whenever an object is deleted, its description is deleted from the catalog at the current server, and any plans or packages that refer to the object are invalidated.

### Invocation

This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is implicitly or explicitly specified.

### Authorization

To drop a table, table space, or index, the privilege set that is defined below must include at least one of the following:

- Ownership of the object (for an index, the owner is the owner of the table or index)
- DBADM authority
- SYSADM or SYSCTRL authority

To drop an alias, storage group, or view, the privilege set that is defined below must include at least one of the following:

- Ownership of the object
- SYSADM or SYSCTRL authority

To drop a database, the privilege set that is defined below must include at least one of the following:

- The DROP privilege on the database
- DBADM or DBCTRL authority for the database
- SYSADM or SYSCTRL authority

To drop a package, the privilege set that is defined below must include at least one of the following:

- Ownership of the package
- The BINDAGENT privilege granted from the package owner
- PACKADM authority for the collection or for all collections
- SYSADM or SYSCTRL authority

To drop a synonym, the privilege set that is defined below must include ownership of the synonym.

To drop a distinct type, stored procedure, trigger, or user-defined function, the privilege set that is defined below must include at least one of the following:

- Ownership of the object <sup>36</sup>
- The DROPIN privilege for the schema or all schemas
- SYSADM or SYSCTRL authority

The authorization ID that matches the schema name implicitly has the DROPIN privilege on the schema.

**Privilege set:** If the statement is embedded in an application program, the privilege set is the privileges that are held by the authorization ID of the owner of the plan or

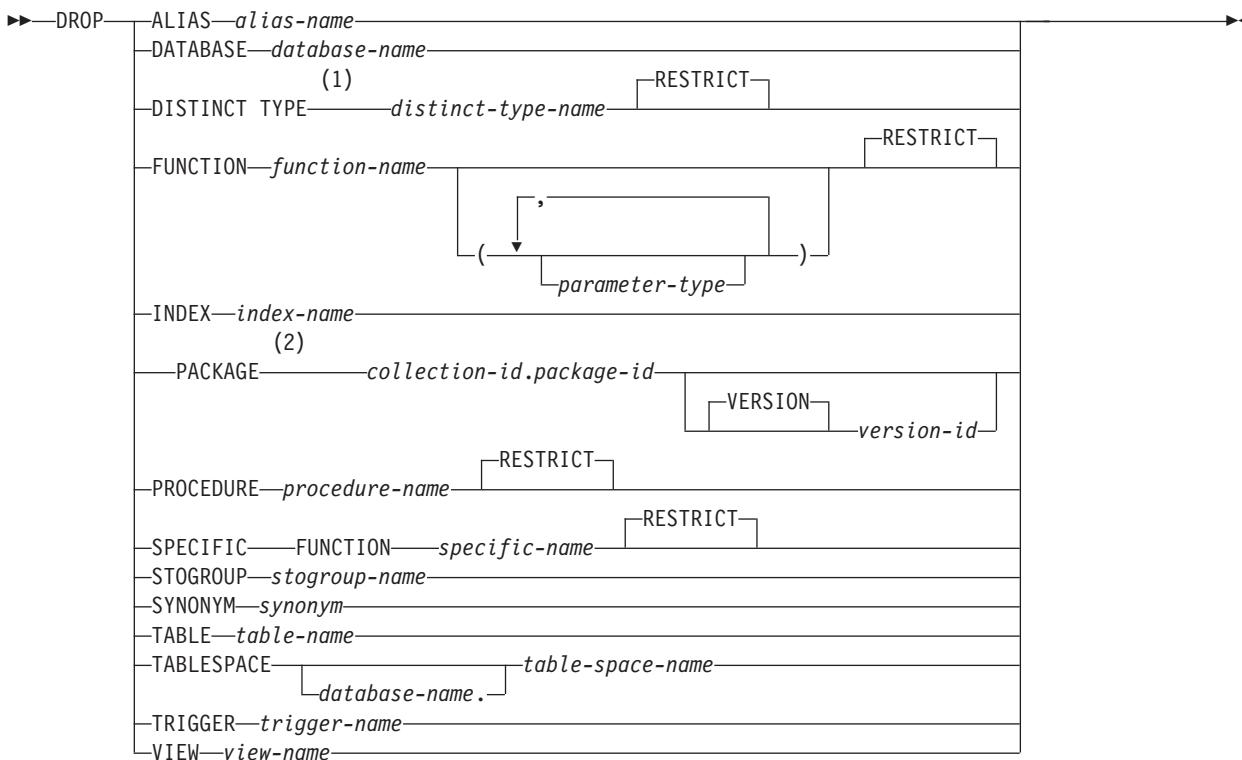
---

36. Not applicable for stored procedures defined in releases of DB2 for OS/390 prior to Version 6.

## DROP

package. If the statement is dynamically prepared, the privilege set is the union of the privilege sets that are held by each authorization ID of the process.

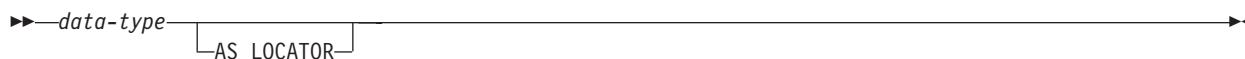
## Syntax



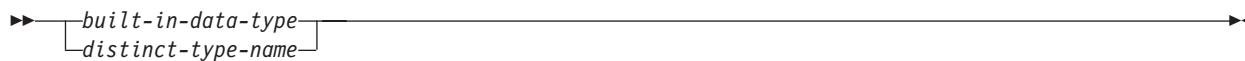
### Notes:

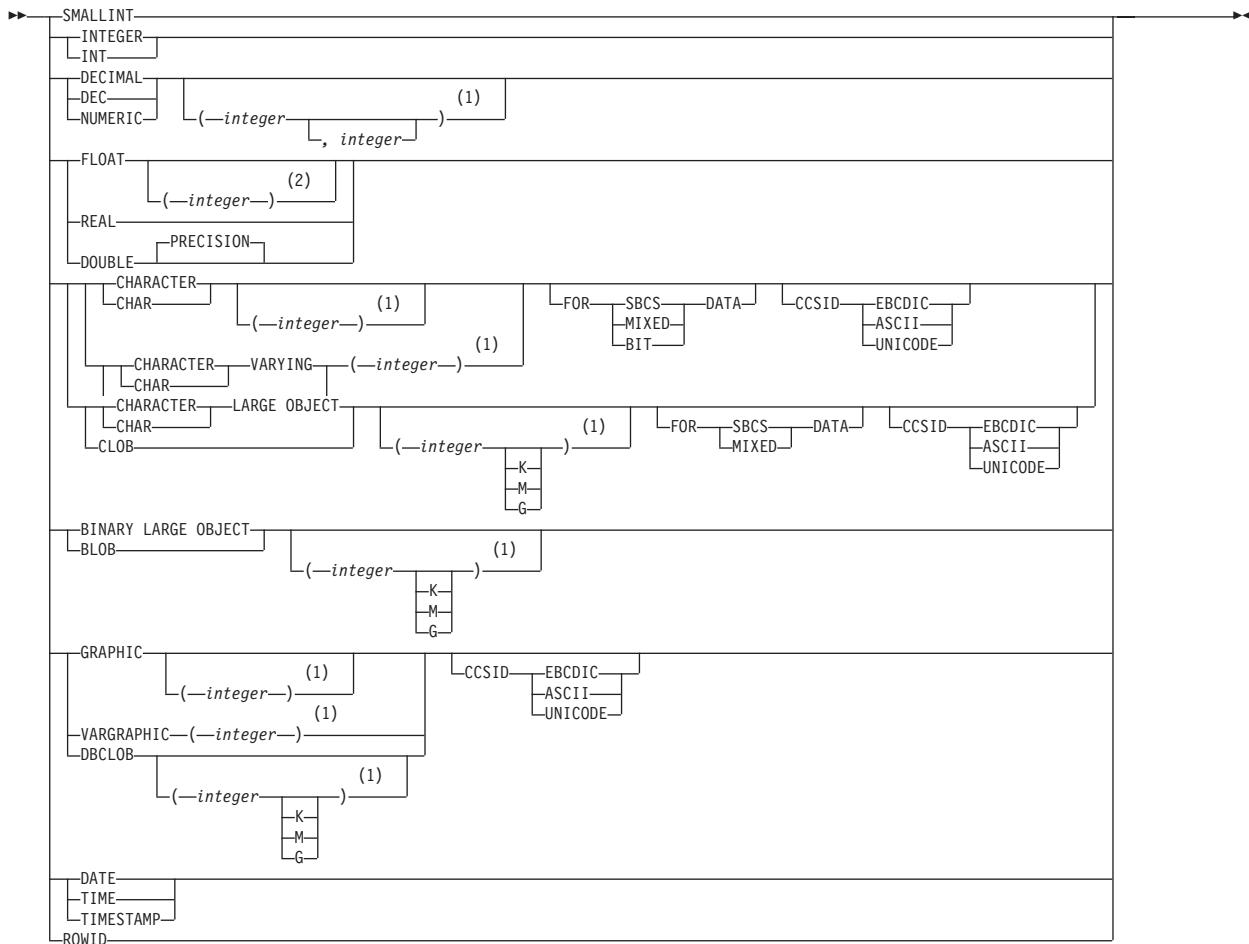
- 1 DATA can be used as a synonym for DISTINCT.
- 2 PROGRAM can be used as a synonym for PACKAGE.

### parameter type:



### data type:



**built-in-data-type:****Notes:**

- 1 The values that are specified for length, precision, or scale attributes must match the values that were specified when the function was created. Coding specific values is optional. Empty parentheses, (), can be used instead to indicate that DB2 ignores the attributes when determining whether data types match.
- 2 The value that is specified does not have to match the value that was specified when the function was created because matching is based on data type (REAL or DOUBLE).  $1 \leq \text{integer} \leq 21$  indicates REAL and  $22 \leq \text{integer} \leq 53$  indicates DOUBLE. Coding a specific value is optional. Empty parentheses cannot be used.

**Description****ALIAS alias-name**

Identifies the alias to be dropped. The name must identify an alias that exists at the current server. Dropping an alias has no effect on any view or synonym that was defined using the alias.

## DROP

### **DATABASE** *database-name*

Identifies the database to be dropped. The name must identify a database that exists at the current server. DSNDB04 or DSNDB06 must not be specified. The privilege set must include SYSADM authority. A TEMP database can be dropped only if none of the table spaces or index spaces that it contains are actively being used.

Whenever a database is dropped, all of its table spaces, tables, index spaces, and indexes are also dropped.

### **DISTINCT TYPE** *distinct-type-name*

Identifies the distinct type to be dropped. The name must identify a distinct type that exists at the current server. RESTRICT enforces the rule that the distinct type is not dropped if any of the following dependencies exist:

- The definition of a column of a table uses the distinct type.
- The definition of an input or result parameter of a user-defined function uses the distinct type.
- The definition of a parameter of a stored procedure uses the distinct type.

Whenever a distinct type is dropped, all privileges on the distinct type are also dropped. In addition, the cast functions that were generated when the distinct type was created and the privileges on those cast functions are also dropped.

### **FUNCTION**

Identifies the user-defined function to be dropped. The name must identify a function that has been defined with the CREATE FUNCTION statement at the current server. The name must not identify a cast function that was generated for a distinct type or a function that is in the SYSIBM schema. RESTRICT enforces the rule that the function is not dropped if any of the following dependencies exist:

- Another function is sourced on the function.
- A view uses the function.
- A trigger package uses the function.

Whenever a function is dropped, all privileges on the user-defined function are also dropped. Any plans or packages that are dependent on the function dropped are made inoperative.

You can identify the particular function to be dropped by its name, function signature, or specific name. If the function was defined with a table parameter (the LIKE TABLE was specified in the CREATE FUNCTION statement to indicate that one of the input parameters is a transition table), the function signature cannot be used to identify the function. Instead, identify the function with its function name, if unique, or with its specific name.

### **FUNCTION** *function-name*

Identifies the function by its name. There must be exactly one function with *function-name* in the implicitly or explicitly specified schema; otherwise, an error occurs.

### **FUNCTION** *function-name (parameter-type,...)*

Provides the function signature, which uniquely identifies the function. If *function-name()* is specified, the function that is identified must have zero parameters. There must be exactly one function with the function signature in the implicitly or explicitly specified schema; otherwise, an error occurs.

If the function was defined with a table parameter (the LIKE TABLE was specified in the CREATE FUNCTION statement to indicate that one of the input parameters is a transition table), the function signature cannot be

#  
#  
#

used to uniquely identify the function. Instead, use one of the other syntax variations to identify the function with its function name, if unique, or its specific name.

*function-name*

Identifies the name of the function.

*(parameter-type,...)*

Identifies the parameters of the function.

If an unqualified distinct type name is specified, DB2 searches the SQL path to resolve the schema name for the distinct type.

The data types of the parameters must match the data types that were specified on the CREATE FUNCTION statement in the corresponding position. The number of data types and the logical concatenation of the data types are used to identify the function.

For data types that have a length, precision, or scale attribute, you can specify a value or use a set of empty parentheses:

- Empty parentheses indicate that DB2 ignores the attribute when determining whether the data types match.  
FLOAT cannot be specified with empty parentheses because its parameter value indicates different data types (REAL or DOUBLE).
- If you use a specific value for a length, precision, or scale attribute, the value must exactly match the value that was specified (implicitly or explicitly) in the CREATE FUNCTION statement.

The specific value for FLOAT(*n*) does not have to exactly match the defined value of the source function because 1<=n<= 21 indicates REAL and 22<=n<=53 indicates DOUBLE. Matching is based on whether the data type is REAL or DOUBLE.

- If length, precision, or scale is not explicitly specified, and empty parentheses are not specified, the default length of the data type is implied. For example:

|                |                      |
|----------------|----------------------|
| <b>CHAR</b>    | CHAR(1)              |
| <b>GRAPHIC</b> | GRAPHIC(1)           |
| <b>DECIMAL</b> | DECIMAL(5,0)         |
| <b>FLOAT</b>   | DOUBLE (length of 8) |

The implicit length must exactly match the value that was specified (implicitly or explicitly) in the CREATE FUNCTION statement. For a complete list of the default lengths of data types, see “CREATE TABLE” on page 653.

For data types with a subtype or encoding scheme attribute, specifying the FOR DATA clause or CCSID clause is optional. Omission of either clause indicates that DB2 ignores the attribute when determining whether the data types match. If you specify either clause, it must match the value that was implicitly or explicitly specified in the CREATE FUNCTION statement.

**SPECIFIC FUNCTION** *specific-name*

Identifies the function by its specific name, which was explicitly specified or implicitly created when the function was created.

**INDEX** *index-name*

Identifies the index to be dropped. The name must identify a user-defined index that exists at the current server but must not identify a partitioning index, or a

## DROP

populated index on an auxiliary table. (For details on dropping user-defined indexes on catalog tables, see “SQL statements allowed on the catalog” on page 1011.) A partitioning index on table T can only be dropped by dropping the table space that contains T. A populated index on an auxiliary table can only be dropped by dropping the base table.

Whenever an index is directly or indirectly dropped, its index space is also dropped. The name of a dropped index space cannot be reused until a commit operation is performed.

If the index is a unique index used to enforce a unique constraint (primary or unique key), the unique constraint must be dropped before the index can be dropped. In addition, if a unique constraint supports a referential constraint, the index cannot be dropped unless the referential constraint is dropped.

However, a unique index (for a unique key only) can be dropped without first dropping the unique key constraint if the unique key was created in a release of DB2 before Version 7 and if the unique key constraint has no associated referential constraints. For information about dropping constraints, see “ALTER TABLE” on page 447.

If a unique index is dropped and that index was defined on a ROWID column that is defined as GENERATED BY DEFAULT, the table can still be used, but rows cannot be inserted into that table.

If an empty index on an auxiliary table is dropped, the base table is marked incomplete.

### **PACKAGE** *collection-id.package-id*

Identifies the package version to be dropped. The name plus the implicitly or explicitly specified *version-id* must identify a package version that exists at the current server. Omission of the *version-id* is an implicit specification of the null version. The name must not identify a trigger package. A trigger package can only be dropped by dropping the associated trigger or subject table.

Whenever the last or only version of a package is dropped, all privileges on the package are dropped and all plans that are dependent on the execute privilege of the package are invalidated.

### *version-id* or **VERSION** *version-id*

*version-id* is the version identifier that was assigned to the package’s DBRM when the DBRM was created. If *version-id* is not specified, a null string is used as the version identifier.

Delimit the version identifier when it:

- Is generated by the VERSION(AUTO) precompiler option
- Begins with a digit
- Contains lowercase or mixed-case letters

For more on version identifiers, see the section on preparing an application program for execution in Part 5 of *DB2 Application Programming and SQL Guide*.

### **PROCEDURE** *procedure-name*

Identifies the stored procedure to be dropped. The name must identify a stored procedure that has been defined with the CREATE PROCEDURE statement at the current server. For dropping a procedure with a schema name SISIBM, the privilege set must include SYSADM or SYSCTRL authority.

#  
#  
#  
#

#  
#  
# When a procedure is directly or indirectly dropped, all privileges on the procedure are also dropped. In addition, any plans or packages that are dependent on the procedure are made invalid.

**STOGROUP** *stogroup-name*

Identifies the storage group to be dropped. The name must identify a storage group that exists at the current server but not a storage group that is used by any table space or index space.

For information on the effect of dropping the default storage group of a database, see “Dropping a default storage group” on page 770.

**SYNONYM** *synonym*

Identifies the synonym to be dropped. In a static DROP SYNONYM statement, the name must identify a synonym that is owned by the owner of the plan or package. In a dynamic DROP SYNONYM statement, the name must identify a synonym that is owned by the SQL authorization ID. Thus, using interactive SQL, a user with SYSADM authority can drop any synonym by first setting CURRENT SQLID to the owner of the synonym.

Dropping a synonym has no effect on any view or alias that was defined using the synonym, nor does it invalidate any plans or packages that use such views or aliases.

**TABLE** *table-name*

Identifies the table to be dropped. The name must identify a table that exists at the current server but must not identify a catalog table, a table in a partitioned table space, or a populated auxiliary table. A table in a partitioned table space can only be dropped by dropping the table space. A populated auxiliary table can only be dropped by dropping the associated base table.

Whenever a table is directly or indirectly dropped, all privileges on the table, all referential constraints in which the table is a parent or dependent, and all synonyms, views, and indexes defined on the table are also dropped. If the table space for the table was implicitly created, it is also dropped.

If a table with LOB columns is dropped, the auxiliary tables associated with the table and the indexes on the auxiliary tables are also dropped. Any LOB table spaces that were implicitly created for the auxiliary tables are also dropped.

If an empty auxiliary table is dropped, the definition of the base table is marked incomplete.

**TABLESPACE** *database-name.table-space-name*

Identifies the table space to be dropped. The name must identify a table space that exists at the current server. The database name must not be DSNDB06. Omission of the database name is an implicit specification of DSNDB04.

Whenever a table space is directly or indirectly dropped, all the tables in the table space are also dropped. The name of a dropped table space cannot be reused until a commit operation is performed.

A table space in a TEMP database can be dropped only if it does not contain an active declared temporary table. A LOB table space can be dropped only if it does not contain an auxiliary table.

Whenever a base table space that contains tables with LOB columns is dropped, all the auxiliary tables and indexes on those auxiliary tables that are associated with the base table space are also dropped.

## **DROP**

|  |                                                                                                                                                                                                                                                              |
|--|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|  | <b>TRIGGER</b> <i>trigger-name</i>                                                                                                                                                                                                                           |
|  | Identifies the trigger to be dropped. The name must identify a trigger that exists at the current server.                                                                                                                                                    |
|  | Whenever a trigger is directly or indirectly dropped, all privileges on the trigger are also dropped and the associated trigger package is freed. The name of that trigger package is the same as the trigger name and the collection ID is the schema name. |
|  | <b>VIEW</b> <i>view-name</i>                                                                                                                                                                                                                                 |
|  | Identifies the view to be dropped. The name must identify a view that exists at the current server.                                                                                                                                                          |
|  | Whenever a view is directly or indirectly dropped, all privileges on the view and all synonyms and views that are defined on the view are also dropped.                                                                                                      |

## **Notes**

**Restrictions on *DROP*:** *DROP* is subject to these restrictions:

- *DROP DATABASE* cannot be performed while a DB2 utility has control of any part of the database.
- *DROP INDEX* cannot be performed while a DB2 utility has control of the index or its associated table space.
- *DROP TABLE* cannot be performed while a DB2 utility has control of the table space that contains the table.
- *DROP TABLESPACE* cannot be performed while a DB2 utility has control of the table space.

In a data sharing environment, the following restrictions also apply:

- If any member has an active resource limit specification table (RLST) you cannot drop the database or table space that contains the table, the table itself, or any index on the table.
- If the member executing the drop cannot access the DB2-managed data sets, only the catalog and directory entries for those data sets are removed.

Objects that have certain dependencies cannot be dropped. For information on these restrictions, see Table 55 on page 773.

***Dropping a parent table:*** *DROP* is not *DELETE* and therefore does not involve delete rules.

***Dropping a default storage group:*** If you drop the default storage group of a database, the database no longer has a legitimate default. You must then specify *USING* in any statement that creates a table space or index in the database. You must do this until you either:

- Create another storage group with the same name using the *CREATE STOGROUP* statement, or
- Designate another default storage group for the database using the *ALTER DATABASE* statement.

***Dropping a table space or index:*** To drop a table space or index, the size of the buffer pool associated with the table space or index must not be zero.

***Dropping a table space in a work file database:*** If one member of a data sharing group drops a table space in a work file database, or an entire work file database,

that belongs to another member, DB2-managed data sets that the executing member cannot access are not dropped. However, the catalog and directory entries for those data sets are removed.

**Dropping resource limit facility (governor) indexes, tables, and table spaces:**

While the RLST is active, you cannot issue a DROP DATABASE, DROP INDEX, DROP TABLE, or DROP TABLESPACE statement for an object associated with an RLST that is active on any member of a data sharing group. See Part 5 (Volume 2) of *DB2 Administration Guide* for details.

**Dropping a temporary table:** To drop a created temporary table or a declared temporary table, use the DROP TABLE statement.

**Dropping an alias:** Dropping a table or view does not drop its aliases. To drop an alias, use the DROP ALIAS statement.

**Dropping a generated procedure:** When you issue DROP PROCEDURE to drop a generated procedure, such as a procedure that is created using the DB2 Development Center, rows for the procedure in SYSIBM.SYSROUTINES\_OPTS and SYSIBM.SYSROUTINES\_SRC will not be dropped if the DROP PROCEDURE statement is issued outside of the DB2 Development Center. These rows must be deleted by using the DELETE statement.

**Dropping an index on an auxiliary table and an auxiliary table:** You can explicitly drop an empty index on an auxiliary table with the DROP INDEX statement. An empty or populated index on an auxiliary table is implicitly dropped when:

- The auxiliary table is empty and it is explicitly dropped (empty indexes only).
  - The associated base table for the auxiliary table is dropped.
  - The base table space that contains the associated base table is dropped.

You can explicitly drop an empty auxiliary table with the `DROP TABLE` statement. An empty or populated auxiliary table is implicitly dropped when:

- The associated base table for the auxiliary table is dropped.
  - The base table space that contains the associated base table is dropped.

Table 54 shows which DROP statements implicitly or explicitly cause an auxiliary table and the index on that table to be dropped, as indicated by the 'D' in the column.

Table 54. Effect of various *DROP* statements on auxiliary tables and indexes

| Statement                                | Auxiliary table |       | Index on auxiliary table |       |
|------------------------------------------|-----------------|-------|--------------------------|-------|
|                                          | Populated       | Empty | Populated                | Empty |
| DROP TABLESPACE<br>(base table space)    | D               | D     | D                        | D     |
| DROP TABLE<br>(base table)               | D               | D     | D                        | D     |
| DROP TABLE<br>(auxiliary table)          | -               | D     | -                        | D     |
| DROP INDEX (index<br>on auxiliary table) | -               | -     | -                        | D     |

---

**Note:** D indicates that the table or index is dropped.

## DROP

**Dropping a migrated index or table space:** Here, “migration” means migrated by the Hierarchical Storage Manager (DFSMSShsm™). DB2 does not wait for any recall of the migrated data sets. Hence, recall is not a factor in the time it takes to execute the statement.

**Deleting SYSLGRNG records for dropped table spaces:** After dropping a table space, you cannot delete the associated records. If you want to remove the records, you must quiesce the table space, and then run the MODIFY RECOVERY utility *before* dropping the table space. If you delete the SYSLGRNG records and drop the table space, you cannot reclaim the table space.

**Dependencies when dropping objects:** Whenever an object is directly or indirectly dropped, other objects that depend on the dropped object might also be dropped. (The catalog stores information about the dependencies of objects on each other.) The following semantics determine what happens to a dependent object when the object that it depends on (the underlying object) is dropped:

- Cascade (D) Dropping the underlying object causes the dependent object to be dropped. However, if the dependent object cannot be dropped because it has a restrict dependency on another object, the drop of the underlying object fails.
- Restrict (D) The underlying object cannot be dropped if a dependent object exists.
- Inoperative (O)
  - Dropping the underlying object causes the dependent object to become inoperative.
- Invalidation (V)
  - Dropping the underlying object causes the dependent object to become invalidated.

For objects that directly depend on others, Table 55 on page 773 uses the letter abbreviations above to summarize what happens to a dependent object when its underlying object is specified in a DROP statement. Additional objects can be indirectly affected, too.

To determine the indirect effects of a DROP statement, assess what happens to the dependent object and whether the dependent object has objects that depend on it. For example, assume that view B is defined on table A and view C is defined on view B. In Table 55 on page 773, the 'D' in the VIEW column of the DROP TABLE row indicates that view B is dropped when table A is dropped. Next, because view C is dependent on view B, check the VIEW column for DROP VIEW. The 'D' in the column indicates that view C will be dropped, too.

Table 55. Effect of dropping objects that have dependencies

| DROP statement                | Type of object |   |   |   |                |                |   |                |   |   |   |   |   |   |                 |   |   |   |   |   |   |   |   |
|-------------------------------|----------------|---|---|---|----------------|----------------|---|----------------|---|---|---|---|---|---|-----------------|---|---|---|---|---|---|---|---|
|                               | D              | I | S | T | I              | F              | P | R              | S | P | O | T | S | A | B               | L | E | T | R | V | I | G | E |
| DROP ALIAS                    | -              | - | - | - | -              | -              | V | -              | - | - | - | - | - | - | -               | - | - | - | - | - | - | - | - |
| DROP DATABASE                 | -              | - | - | - | -              | D <sup>2</sup> | - | -              | - | - | - | - | - | D | D               | - | - | D | - | - | - | - | - |
| DROP DISTINCT TYPE            | -              | - | - | - | R <sup>3</sup> | -              | - | R <sup>4</sup> | - | - | - | - | - | R | -               | - | - | - | - | - | - | - | - |
| DROP FUNCTION                 | -              | - | - | - | R <sup>5</sup> | -              | O | -              | - | - | - | - | - | - | -               | - | - | R | R | - | - | - | - |
| DROP INDEX <sup>2,6</sup>     | -              | - | - | - | -              | V              | - | -              | - | - | - | - | - | - | -               | - | - | V | - | - | - | - | - |
| DROP PACKAGE <sup>7</sup>     | -              | - | - | - | -              | -              | - | -              | - | - | - | - | - | - | -               | - | - | - | - | - | - | - | - |
| DROP PROCEDURE                | -              | - | - | - | -              | -              | O | -              | - | - | - | - | - | - | -               | - | - | R | - | - | - | - | - |
| DROP STOGROUP                 | -              | - | - | - | R <sup>8</sup> | -              | - | -              | - | - | - | - | - | - | R <sup>8</sup>  | - | - | - | - | - | - | - | - |
| DROP SYNONYM                  | -              | - | - | - | -              | -              | - | -              | - | - | - | - | - | - | -               | - | - | - | - | - | - | - | - |
| DROP TABLE <sup>9,10</sup>    | -              | - | - | - | D              | V              | - | -              | - | D | - | - | - | - | D <sup>11</sup> | D | - | - | - | - | - | - | - |
| DROP TABLESPACE <sup>12</sup> | -              | - | - | - | D              | V              | - | -              | - | D | - | - | - | D | -               | - | - | - | - | - | - | - | - |
| DROP TRIGGER                  | -              | - | - | - | -              | -              | - | -              | - | - | - | - | - | - | -               | - | - | - | - | - | - | - | - |
| DROP VIEW                     | -              | - | - | - | -              | -              | V | -              | - | D | - | - | - | D | -               | - | - | - | D | - | - | - | - |

**Legend:**

- D** Dependent object is dropped.
- O** Dependent object is made inoperative.
- V** Dependent object is invalidated.
- R** DROP statement fails.

**Notes:**

1. The PACKAGE column represents packages for user-defined functions, procedures, and triggers, as well as other packages. The PACKAGE column also applies for plans.
2. The index space associated with the index is dropped.
3. If a function is dependent on the distinct type being dropped, the distinct type cannot be dropped unless the function is one of the cast functions that was created for the distinct type.
4. If the definition of a parameter of a stored procedure uses the distinct type, the distinct type cannot be dropped.
5. If other user-defined functions are sourced on the user-defined function being dropped, the function cannot be dropped.
6. An index on an auxiliary table cannot be explicitly dropped.
7. A trigger package cannot be explicitly dropped with DROP PACKAGE. A trigger package is implicitly dropped when the associated trigger or subject table is dropped.
8. A storage group cannot be dropped if it is used by any table space or index space.
9. An auxiliary table cannot be explicitly dropped with DROP TABLE. An auxiliary table is implicitly dropped when the associated base table is dropped.
10. If an implicit table space was created when the table was created, the table space is also dropped.
11. When a subject table is dropped, the associated trigger and trigger package are also dropped.
12. A LOB table space cannot be dropped until the base table with the LOB columns is dropped. A table space in a TEMP database cannot be dropped if it contains an active declared temporary table.

## DROP

### Examples

*Example 1:* Drop table DSN8710.DEPT.

```
DROP TABLE DSN8710.DEPT;
```

*Example 2:* Drop table space DSN8S71D in database DSN8D71A.

```
DROP TABLESPACE DSN8D71A.DSN8S71D;
```

*Example 3:* Drop the view DSN8710.VPROJRE1:

```
DROP VIEW DSN8710.VPROJRE1;
```

*Example 4:* Drop the package DSN8CC0 with the version identifier VERSZZZZ. The package is in the collection DSN8CC61. Use the version identifier to distinguish the package to be dropped from another package with the same name in the same collection.

```
DROP PACKAGE DSN8CC61.DSN8CC0 VERSION VERSZZZZ;
```

*Example 5:* Drop the package DSN8CC0 with the version identifier "1994-07-14-09.56.30.196952". When a version identifier is generated by the VERSION(AUTO) precompiler option, delimit the version identifier.

```
DROP PACKAGE DSN8CC61.DSN8CC0 VERSION "1994-07-14-09.56.30.196952";
```

*Example 6:* Drop the distinct type DOCUMENT, if it is not currently in use:

```
DROP DISTINCT TYPE DOCUMENT;
```

*Example 7:* Assume that you are SMITH and that ATOMIC\_WEIGHT is the only function with that name in schema CHEM. Drop ATOMIC\_WEIGHT.

```
DROP FUNCTION CHEM.ATOMIC_WEIGHT;
```

*Example 8:* Assume that you are SMITH and that you created the function CENTER in schema SMITH. Drop CENTER, using the function signature to identify the function instance to be dropped.

```
DROP FUNCTION CENTER(INTEGER, FLOAT);
```

*Example 9:* Assume that you are SMITH and that you created another function named CENTER, which you gave the specific name FOCUS97, in schema JOHNSON. Drop CENTER, using the specific name to identify the function instance to be dropped.

```
DROP SPECIFIC FUNCTION JOHNSON.FOCUS97;
```

*Example 10:* Assume that you are SMITH and that stored procedure OSMOSIS is in schema BIOLOGY. Drop OSMOSIS.

```
DROP PROCEDURE BIOLOGY.OSMOSIS;
```

*Example 11:* Assume that you are SMITH and that trigger BONUS is in your schema. Drop BONUS.

```
DROP TRIGGER BONUS;
```

## END DECLARE SECTION

The END DECLARE SECTION statement marks the end of a host variable declare section.

### Invocation

This statement can only be embedded in an application program. It is not an executable statement.

### Authorization

None required.

### Syntax

```
►►—END DECLARE SECTION————►►
```

### Description

The END DECLARE SECTION statement can be coded in the application program wherever declarations can appear in accordance with the rules of the host language. It is used to indicate the end of a host variable declaration section. A host variable section starts with a BEGIN DECLARE SECTION statement described in “BEGIN DECLARE SECTION” on page 482.

The following rules are enforced by the precompiler only if the host language is C or the STDSQL(YES) precompiler option is specified:

- A variable referred to in an SQL statement must be declared within a host variable declaration section of the source program.
- BEGIN DECLARE SECTION and END DECLARE SECTION statements must be paired and must not be nested.
- Host variable declaration sections can contain only host variable declarations, SQL INCLUDE statements that include host variable declarations, or DECLARE VARIABLE statements.

### Notes

Host variable declaration sections are only required if the STDSQL(YES) option is specified or the host language is C. However, declare sections can be specified for any host language so that the source program can conform to IBM SQL. If declare sections are used, but not required, variables declared outside a declare section should not have the same name as variables declared within a declare section.

### Example

```
EXEC SQL BEGIN DECLARE SECTION;
 (host variable declarations)
EXEC SQL END DECLARE SECTION;
```

## EXECUTE

## EXECUTE

The EXECUTE statement executes a prepared SQL statement.

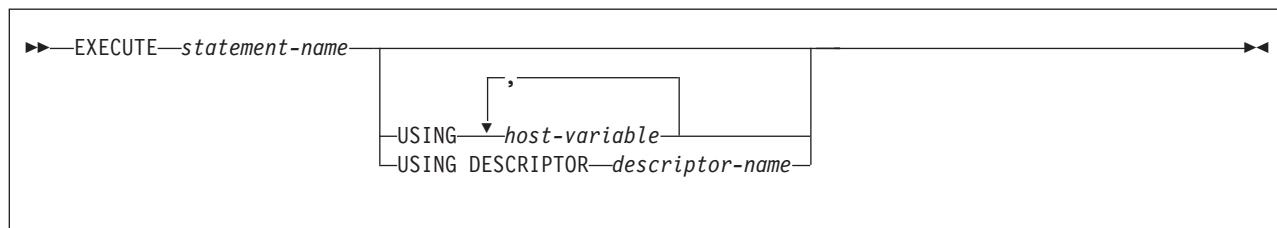
### Invocation

This statement can only be embedded in an application program. It is an executable statement that cannot be dynamically prepared.

### Authorization

See “PREPARE” on page 846 for the authorization required to create a prepared statement.

### Syntax



### Description

#### *statement-name*

Identifies the prepared statement to be executed. *statement-name* must identify a statement that was previously prepared within the unit of work and the prepared statement must not be a SELECT statement.

#### USING

Introduces a list of host variables whose values are substituted for the parameter markers (question marks) in the prepared statement. (For an explanation of parameter markers, see “PREPARE” on page 846.) If the prepared statement includes parameter markers, you must include USING in the EXECUTE statement. USING is ignored if there are no parameter markers.

For more on the substitution of values for parameter markers, see “Parameter marker replacement” on page 777.

#### *host-variable*,...

Identifies structures or variables that must be described in the application program in accordance with the rules for declaring host structures and variables. In the operational form of the clause, a reference to a structure is replaced by a reference to each of its variables. After all the replacements, the number of variables must be the same as the number of parameter markers in the prepared statement. The *n*th variable supplies the value for the *n*th parameter marker in the prepared statement.

#### USING DESCRIPTOR *descriptor-name*

Identifies an SQLDA that contains a valid description of the input host variables.

Before the EXECUTE statement is processed, the user must set the following fields in the SQLDA:

- SQLN to indicate the number of SQLVAR occurrences provided in the SQLDA

A REXX SQLDA does not contain this field.

- SQLD to indicate the number of variables used in the SQLDA when processing the statement
- SQLVAR occurrences to indicate the attributes of the variables

The SQLDA must have enough storage to contain all SQLVAR occurrences. If LOBs or distinct types are present in the results, there must be additional SQLVAR entries for each parameter. For more information on the SQLDA, which includes a description of the SQLVAR and an explanation on how to determine the number of SQLVAR occurrences, see “SQL descriptor area (SQLDA)” on page 986.

SQLD must be set to a value greater than or equal to zero and less than or equal to SQLN. It must be the same as the number of parameter markers in the prepared statement.

See “Identifying an SQLDA in C or C++” on page 1003 for how to represent *descriptor-name* in C.

## Notes

DB2 can stop the execution of a prepared SQL statement if the statement is taking too much processor time to finish. When this happens, an error occurs. The application that issued the statement is not terminated; it is allowed to issue another SQL statement.

**Parameter marker replacement:** Before the prepared statement is executed, each parameter marker in the statement is effectively replaced by its corresponding host variable. The replacement is an assignment operation in which the source is the value of the host variable and the target is a variable within DB2. The assignment rules are those described for assignment to a column in “Assignment and comparison” on page 64. For a typed parameter marker, the attributes of the target variable are those specified by the CAST specification. For an untyped parameter marker, the attributes of the target variable are determined according to the context of the parameter marker. For the rules that affect parameter markers, see “Parameter markers” on page 852.

Let V denote a host variable that corresponds to parameter marker P. The value of V is assigned to the target variable for P in accordance with the rules for assigning a value to a column:

- V must be compatible with the target.
- If V is a string, its length must not be greater than the length attribute of the target.
- If V is a number, the absolute value of its integral part must not be greater than the maximum absolute value of the integral part of the target.
- If the attributes of V are not identical to the attributes of the target, the value is converted to conform to the attributes of the target.
- If the target cannot contain nulls, V must not be null.

When the prepared statement is executed, the value used in place of P is the value of the target variable for P. For example, if V is CHAR(6) and the target is CHAR(8), the value used in place of P is the value of V padded on the right with two blanks.

## EXECUTE

**Errors occurring on EXECUTE:** In local and remote processing, the DEFER(PREPARE) and REOPT(VARS) bind options can cause some errors that are normally issued during PREPARE processing to be issued on EXECUTE.

### Example

In this example, an INSERT statement with parameter markers is prepared and executed. S1 is a structure that corresponds to the format of DSN8710.DEPT.

```
EXEC SQL PREPARE DEPT_INSERT FROM
 'INSERT INTO DSN8710.DEPT VALUES(?, ?, ?, ?)';

(Check for successful execution and read values into S1)

EXEC SQL EXECUTE DEPT_INSERT USING :S1;
```

## EXECUTE IMMEDIATE

The EXECUTE IMMEDIATE statement:

- Prepares an executable form of an SQL statement from a string form of the statement
- Executes the SQL statement
- Destroys the executable form

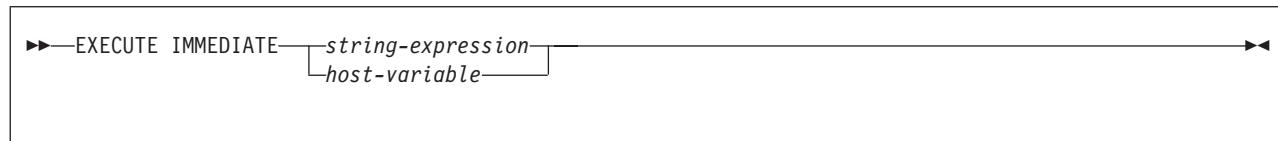
### Invocation

This statement can only be embedded in an application program. It is an executable statement that cannot be dynamically prepared.

### Authorization

The authorization rules are those defined for the dynamic preparation of the SQL statement specified by EXECUTE IMMEDIATE. For example, see “INSERT” on page 832 for the authorization rules that apply when an INSERT statement is executed using EXECUTE IMMEDIATE.

### Syntax



### Description

#### *string-expression*

*string-expression* is any PL/I expression that yields a string. If the source program does not include any DECLARE VARIABLE statements, an optional colon can precede the *string-expression*. The colon introduces PL/I syntax. Therefore, host variables within a *string-expression* that includes operators or functions should not be preceded with a colon. However, if the source program includes at least one DECLARE VARIABLE statement, a *string-expression* cannot be preceded by a colon and an expression that consists of just a variable name preceded by a colon is interpreted as a *host-variable*, not as a *string-expression*. The precompiler-generated structures for a *string-expression* use an EBCDIC CCSID.

#### *host-variable*

For languages other than PL/I, *host-variable* must be specified. It must identify a host variable that is described in the application program in accordance with the rules for declaring character string variables. The host variable must not have a CLOB data type, and an indicator variable must not be specified. In Assembler language, C, and COBOL, the host variable must be a varying-length string variable. In C, it must not be a NUL-terminated string.

In PL/I, if the source program includes at least one DECLARE VARIABLE statement, a host variable (preceded by a colon) is considered a *host-variable* and must be a varying-length string variable. The host variable may be either a fixed-length or varying-length string variable if the source program does not include any DECLARE VARIABLE statements. It is then considered a

## EXECUTE IMMEDIATE

|  
|       *string-expression*. When a *string-expression* is used, the precompiler-generated  
| structures for it use an EBCDIC CCSID and an informational message is  
| issued.

### Notes

The value of the identified host variable or the specified *string-expression* is called the *statement string*.

The statement string must be one of the following SQL statements:

|                                |                               |
|--------------------------------|-------------------------------|
| ALTER                          | RENAME                        |
| COMMENT ON                     | REVOKE                        |
| COMMIT                         | ROLLBACK                      |
| CREATE                         | SET CURRENT DEGREE            |
| DECLARE GLOBAL TEMPORARY TABLE | SET CURRENT LOCALE LC_CTYPE   |
| DELETE                         | SET CURRENT OPTIMIZATION HINT |
| DROP                           | SET CURRENT PRECISION         |
| EXPLAIN                        | SET CURRENT RULES             |
| GRANT                          | SET CURRENT SQLID             |
| INSERT                         | SET PATH                      |
| LABEL ON                       | UPDATE                        |
| LOCK TABLE                     |                               |

The statement string must not include parameter markers or references to host variables, must not begin with EXEC SQL, and must not terminate with END-EXEC or a semicolon.

When an EXECUTE IMMEDIATE statement is executed, the specified statement string is parsed and checked for errors. If the SQL statement is invalid, it is not executed and the error condition that prevents its execution is reported in the SQLCA. If the SQL statement is valid, but an error occurs during its execution, that error condition is reported in the SQLCA.

DB2 can stop the execution of a prepared SQL statement if the statement is taking too much CPU time to finish. When this happens an error occurs. The application that issued the statement is not terminated; it is allowed to issue another SQL statement.

If the same SQL statement is to be executed more than once, it is more efficient to use the PREPARE and EXECUTE statements rather than the EXECUTE IMMEDIATE statement.

### Example

In this PL/I example, the EXECUTE IMMEDIATE statement is used to execute a DELETE statement in which the rows to be deleted are determined by a search-condition specified by the value of PREDS.

```
EXEC SQL EXECUTE IMMEDIATE 'DELETE FROM DSN8710.DEPT
WHERE' || PREDS;
```

## EXPLAIN

The information about this statement is Product-sensitive Programming Interface and Associated Guidance Information, as defined in Appendix H, “Notices,” on page 1173.

The EXPLAIN statement obtains information about access path selection for an *explainable statement*. A statement is explainable if it is a SELECT or INSERT statement, or the searched form of an UPDATE or DELETE statement. The information obtained is placed in a user-supplied *plan table*.

Optionally, EXPLAIN can also obtain and place information in two additional tables. A user-supplied *statement table* can be populated with information about the estimated cost of executing the explainable statement. A user-supplied *function table* can be populated with information about how DB2 resolves the user-defined functions that are referred to in the explainable statement.

### Invocation

This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared.

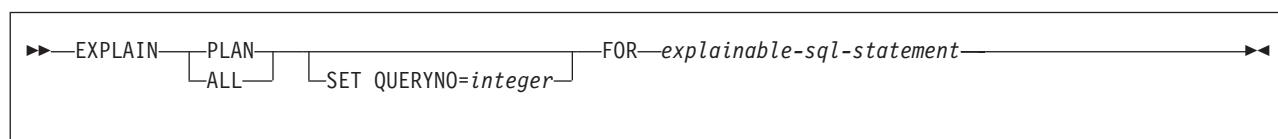
### Authorization

The authorization rules are those defined for the SQL statement specified in the EXPLAIN statement. For example, see the description of the DELETE statement for the authorization rules that apply when a DELETE statement is explained.

If the EXPLAIN statement is embedded in an application program, the authorization rules that apply are those defined for embedding the specified SQL statement in an application program. In addition, the authorization ID of the owner of the plan or package must also be the owner of a plan table named PLAN\_TABLE.

If the EXPLAIN statement is dynamically prepared, the authorization rules that apply are those defined for dynamically preparing the specified SQL statement. In addition, the SQL authorization ID of the process must also be the owner of a plan table named PLAN\_TABLE.

### Syntax



### Description

#### PLAN

Inserts one row into the plan table for each step used in executing *explainable-sql-statement*. The steps for enforcing referential constraints are not included. See “Creating a plan table” on page 783 and Table 56 on page 784 for the format and column descriptions of the plan table.

If a statement table exists, a row that provides a cost estimate of processing the explainable statement is inserted into the statement table. See “Creating a statement table” on page 789 and Table 57 on page 789 for the format and column descriptions of the statement table.

## EXPLAIN

If a function table exists, one row is inserted into the function table for each user-defined function that is referred to by the explainable statement. See “Creating a function table” on page 790 and Table 58 on page 791 for the format and column descriptions of the function table.

### ALL

Has the same effect as PLAN.

### SET QUERYNO = *integer*

Associates *integer* with *explainable-sql-statement*. The column QUERYNO is given the value *integer* in every row inserted into the plan table, statement table, or function table by the EXPLAIN statement. If QUERYNO is not specified, DB2 itself assigns a number. For an embedded EXPLAIN statement, the number is the statement number that was assigned by the precompiler and placed in the DBRM.

### FOR *explainable-sql-statement*

Specifies the SQL statement to be explained. *explainable-sql-statement* can be any explainable SQL statement. If EXPLAIN is embedded in a program, the statement can contain references to host variables. If EXPLAIN is dynamically prepared, the statement can contain parameter markers. Host variables that appear in the statement must be defined in the statement’s program.

The statement must refer to objects at the current server.

*explainable-sql-statement* must not contain a QUERYNO clause. To specify the value of the QUERYNO column in plan table for the statement being explained, use the SET QUERYNO = clause of the EXPLAIN statement.

*explainable-sql-statement* cannot be a statement-name or a host-variable. To use EXPLAIN to get information about dynamic SQL statements, you must prepare the entire EXPLAIN statement dynamically.

To obtain information about an explainable SQL statement that references a declared temporary table, the EXPLAIN statement must be executed in the same application process in which the table was declared. For static EXPLAIN statements, the information is not obtained at bind-time but at run-time when the EXPLAIN statement is incrementally bound.

## Notes

**Output from EXPLAIN:** Output from EXPLAIN is one or more rows of data inserted into the plan table. Rows are also inserted into the statement table and function table if the tables exist. The plan table must be created before the EXPLAIN statement is executed. Unless you need the information that the statement table or function table provides, it is not necessary to create either table to use EXPLAIN.

The tables have the following names:

|                 |                                   |
|-----------------|-----------------------------------|
| plan table      | <i>userid</i> .PLAN_TABLE         |
| statement table | <i>userid</i> .DSN_STATEMNT_TABLE |
| function table  | <i>userid</i> .DSN_FUNCTION_TABLE |

where *userid* is:

- The owner of the plan or package if the EXPLAIN statement is embedded in a plan or package.
- The SQL authorization ID of the process if the statement is dynamically prepared.

For information on using the plan table and the statement table, see Part 5 (Volume 2) of *DB2 Administration Guide*. For information on using the function table, see Part 3 of *DB2 Application Programming and SQL Guide*.

**Output from BIND or REBIND:** DB2 can also add rows to a plan table, statement table, and function table when a plan or package is bound or rebound. This addition of rows occurs when the BIND or REBIND subcommand is executed with the EXPLAIN(YES) option in effect. The option requires that rows be added for every explainable statement in the plan or package being bound. For a plan, these do not include statements in the packages that can be used with the plan. For either a package or plan, they do not include explainable statements within EXPLAIN statements nor do they include explainable statements that refer to declared temporary tables, which are incrementally bound at run-time.

The plan table must exist when the BIND or REBIND subcommand is executed. Unless you need the information that the statement table or function table provides, neither table has to exist. Only the tables that exist receive new rows. The tables have the following names:

|                 |                                  |
|-----------------|----------------------------------|
| plan table      | <i>userid.PLAN_TABLE</i>         |
| statement table | <i>userid.DSN_STATEMNT_TABLE</i> |
| function table  | <i>userid.DSN_FUNCTION_TABLE</i> |

where *userid* is the owner of the plan or package.

**Creating a plan table:** To create a plan table, execute the following SQL statement:

```
CREATE TABLE userid.PLAN_TABLE
 (QUERYNO INTEGER NOT NULL,
 QBLOCKNO SMALLINT NOT NULL,
 APPLNAME CHAR(8) NOT NULL,
 PROGNAME CHAR(8) NOT NULL,
 PLANNO SMALLINT NOT NULL,
 METHOD SMALLINT NOT NULL,
 CREATOR CHAR(8) NOT NULL,
 TNAME CHAR(18) NOT NULL,
 TABNO SMALLINT NOT NULL,
 ACESSTYPE CHAR(2) NOT NULL,
 MATCHCOLS SMALLINT NOT NULL,
 ACCESSCREATOR CHAR(8) NOT NULL,
 ACCESSNAME CHAR(18) NOT NULL,
 INDEXONLY CHAR(1) NOT NULL,
 SORTN_UNIQ CHAR(1) NOT NULL,
 SORTN_JOIN CHAR(1) NOT NULL,
 SORTN_ORDERBY CHAR(1) NOT NULL,
 SORTN_GROUPBY CHAR(1) NOT NULL,
 SORTC_UNIQ CHAR(1) NOT NULL,
 SORTC_JOIN CHAR(1) NOT NULL,
 SORTC_ORDERBY CHAR(1) NOT NULL,
 SORTC_GROUPBY CHAR(1) NOT NULL,
 TSLOCKMODE CHAR(3) NOT NULL,
 TIMESTAMP CHAR(16) NOT NULL,
 REMARKS VARCHAR(254) NOT NULL,
 PREFETCH CHAR(1) NOT NULL WITH DEFAULT,
 COLUMN_FN_EVAL CHAR(1) NOT NULL WITH DEFAULT,
 MIXOPSEQ SMALLINT NOT NULL WITH DEFAULT,
 VERSION VARCHAR(64) NOT NULL WITH DEFAULT,
 COLLID CHAR(18) NOT NULL WITH DEFAULT,
 ACCESS_DEGREE SMALLINT ,
 ACCESS_PGROUP_ID SMALLINT ,
 JOIN_DEGREE SMALLINT ,
 JOIN_PGROUP_ID SMALLINT ,
 SORTC_PGROUP_ID SMALLINT ,
 SORTN_PGROUP_ID SMALLINT ,
 PARALLELISM_MODE CHAR(1) ,
 MERGE_JOIN_COLS SMALLINT ,
 CORRELATION_NAME CHAR(18) ,
 PAGE_RANGE CHAR(1) NOT NULL WITH DEFAULT,
 JOIN_TYPE CHAR(1) NOT NULL WITH DEFAULT,
```

## EXPLAIN

```
GROUP_MEMBER CHAR(8) NOT NULL WITH DEFAULT,
IBM_SERVICE_DATA VARCHAR(254) NOT NULL WITH DEFAULT,
WHEN_OPTIMIZE CHAR(1) NOT NULL WITH DEFAULT,
QBLOCK_TYPE CHAR(6) NOT NULL WITH DEFAULT,
BIND_TIME TIMESTAMP NOT NULL WITH DEFAULT,
OPTHINT CHAR(8) NOT NULL WITH DEFAULT,
HINT_USED CHAR(8) NOT NULL WITH DEFAULT,
PRIMARY_ACESSTYPE CHAR(1) NOT NULL WITH DEFAULT,
PARENT_QBLOCKNO SMALLINT NOT NULL WITH DEFAULT,
TABLE_TYPE CHAR(1))
IN database-name.table-space-name;
```

where *database-name.table-space-name* identifies a database and table space you have authorization to use.

**Plan table column descriptions:** Table 56 explains the columns in PLAN\_TABLE. The explanations apply both to rows resulting from the execution of an EXPLAIN statement and to rows resulting from a bind or rebind.

Each row in a plan table describes a step in the execution of a query or subquery in an explainable statement. The column values for the row identify, among other things, the query or subquery, the tables involved, and the method used to carry out the step.

Table 56. Descriptions of columns in PLAN\_TABLE

| Column Name | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
|-------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| QUERYNO     | A number intended to identify the statement being explained. For a row produced by an EXPLAIN statement, specify the number in the QUERYNO clause. For a row produced by non-EXPLAIN statements, specify the number using the QUERYNO clause, which is an optional part of the SELECT, INSERT, UPDATE and DELETE statement syntax. Otherwise, DB2 assigns a number based on the line number of the SQL statement in the source program.<br><br>When the values of QUERYNO are based on the statement number in the source program, values greater than 32767 are reported as 0. Hence, in a very long program, the value is not guaranteed to be unique. If QUERYNO is not unique, the value of TIMESTAMP is unique. |
| QBLOCKNO    | The position of the query in the statement being explained (1 for the outermost query, 2 for the next query, and so forth). For better performance, DB2 might merge a query block into another query block. When that happens, the position number of the merged query block will not be in QBLOCKNO.                                                                                                                                                                                                                                                                                                                                                                                                                |
| APPLNAME    | The name of the application plan for the row. Applies only to embedded EXPLAIN statements executed from a plan or to statements explained when binding a plan. Blank if not applicable.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| PROGNAME    | The name of the program or package containing the statement being explained. Applies only to embedded EXPLAIN statements and to statements explained as the result of binding a plan or package. Blank if not applicable.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| PLANNO      | The number of the step in which the query indicated in QBLOCKNO was processed. This column indicates the order in which the steps were executed.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |

Table 56. Descriptions of columns in PLAN\_TABLE (continued)

| Column Name   | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
|---------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| METHOD        | A number (0, 1, 2, 3, or 4) that indicates the join method used for the step:<br><br>0 First table accessed, continuation of previous table accessed, or not used.<br>1 <i>Nested loop</i> join. For each row of the present composite table, matching rows of a new table are found and joined.<br>2 <i>Merge scan</i> join. The present composite table and the new table are scanned in the order of the join columns, and matching rows are joined.<br>3 Sorts needed by ORDER BY, GROUP BY, SELECT DISTINCT, UNION, a quantified predicate, or an IN predicate. This step does not access a new table.<br>4 <i>Hybrid</i> join. The current composite table is scanned in the order of the join-column rows of the new table. The new table is accessed using list prefetch. |
| CREATOR       | The creator of the new table accessed in this step, blank if METHOD is 3.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| TNAME         | The name of a table, created or declared temporary table, materialized view, or materialized table expression. The value is blank if METHOD is 3. The column can also contain the name of a table in the form DSNWFQB( <i>qblockno</i> ).<br>DSNWFQB( <i>qblockno</i> ) is used to represent the intermediate result of a UNION ALL or an outer join that is materialized. If a view is merged, the name of the view does not appear.                                                                                                                                                                                                                                                                                                                                             |
|               | A value of Q in TABLE_TYPE for the name of a view or nested table expression indicates that the materialization was virtual and not actual. Materialization can be virtual when the view or nested table expression definition contains a UNION ALL that is not distributed.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| TABNO         | Values are for IBM use only.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| ACCESSTYPE    | The method of accessing the new table:<br><br>I By an index (identified in ACCESSCREATOR and ACCESSNAME)<br>I1 By a one-fetch index scan<br>N By an index scan when the matching predicate contains the IN keyword<br>R By a table space scan<br>M By a multiple index scan (followed by MX, MI, or MU)<br>MX By an index scan on the index named in ACCESSNAME<br>MI By an intersection of multiple indexes<br>MU By a union of multiple indexes<br>T By a sparse index (star join work files)<br>blank Not applicable to the current row                                                                                                                                                                                                                                        |
| #             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| MATCHCOLS     | For ACESSTYPE I, I1, N, or MX, the number of index keys used in an index scan; otherwise, 0.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| ACCESSCREATOR | For ACESSTYPE I, I1, N, or MX, the creator of the index; otherwise, blank.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| ACCESSNAME    | For ACESSTYPE I, I1, N, or MX, the name of the index; otherwise, blank.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| INDEXONLY     | Whether access to an index alone is enough to carry out the step, or whether data too must be accessed. Y=Yes; N=No. For exceptions, see Part 5 (Volume 2) of DB2 Administration Guide.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| SORTN_UNIQ    | Whether the new table is sorted to remove duplicate rows. Y=Yes; N=No.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| SORTN_JOIN    | Whether the new table is sorted for join method 2 or 4. Y=Yes; N=No.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| SORTN_ORDERBY | Whether the new table is sorted for ORDER BY. Y=Yes; N=No.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| SORTN_GROUPBY | Whether the new table is sorted for GROUP BY. Y=Yes; N=No.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| SORTC_UNIQ    | Whether the composite table is sorted to remove duplicate rows. Y=Yes; N=No.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| SORTC_JOIN    | Whether the composite table is sorted for join method 1, 2 or 4. Y=Yes; N=No.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |

## EXPLAIN

Table 56. Descriptions of columns in PLAN\_TABLE (continued)

| Column Name                                                                                                                                                                                                                                                                      | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| SORTC_ORDERBY                                                                                                                                                                                                                                                                    | Whether the composite table is sorted for an ORDER BY clause or a quantified predicate. Y=Yes; N=No.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| SORTC_GROUPBY                                                                                                                                                                                                                                                                    | Whether the composite table is sorted for a GROUP BY clause. Y=Yes; N=No.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| TSLOCKMODE                                                                                                                                                                                                                                                                       | An indication of the mode of lock to be acquired on either the new table, or its table space or table space partitions. If the isolation can be determined at bind time, the values are:<br><b>IS</b> Intent share lock<br><b>IX</b> Intent exclusive lock<br><b>S</b> Share lock<br><b>U</b> Update lock<br><b>X</b> Exclusive lock<br><b>SIX</b> Share with intent exclusive lock<br><b>N</b> UR isolation; no lock<br>If the isolation cannot be determined at bind time, then the lock mode determined by the isolation at run time is shown by the following values.<br><b>NS</b> For UR isolation, no lock; for CS, RS, or RR, an S lock.<br><b>NIS</b> For UR isolation, no lock; for CS, RS, or RR, an IS lock.<br><b>NSS</b> For UR isolation, no lock; for CS or RS, an IS lock; for RR, an S lock.<br><b>SS</b> For UR, CS, or RS isolation, an IS lock; for RR, an S lock. |
|                                                                                                                                                                                                                                                                                  | The data in this column is right justified. For example, IX appears as a blank followed by I followed by X. If the column contains a blank, then no lock is acquired.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| TIMESTAMP                                                                                                                                                                                                                                                                        | Usually, the time at which the row is processed, to the last .01 second. If necessary, DB2 adds .01 second to the value to ensure that rows for two successive queries have different values.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| REMARKS                                                                                                                                                                                                                                                                          | A field into which you can insert any character string of 254 or fewer characters.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| PREFETCH                                                                                                                                                                                                                                                                         | Whether data pages are to be read in advance by prefetch. S = pure sequential prefetch; L = prefetch through a page list; blank = unknown or no prefetch.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| COLUMN_FN_EVAL                                                                                                                                                                                                                                                                   | When an SQL column function is evaluated. R = while the data is being read from the table or index; S = while performing a sort to satisfy a GROUP BY clause; blank = after data retrieval and after any sorts.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| MIXOPSEQ                                                                                                                                                                                                                                                                         | The sequence number of a step in a multiple index operation.<br><br><b>1, 2, ... n</b> For the steps of the multiple index procedure (ACCESSTYPE is MX, MI, or MU.)<br><br><b>0</b> For any other rows (ACCESSTYPE is I, I1, M, N, R, or blank.)                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| VERSION                                                                                                                                                                                                                                                                          | The version identifier for the package. Applies only to an embedded EXPLAIN statement executed from a package or to a statement that is explained when binding a package. Blank if not applicable.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| COLLID                                                                                                                                                                                                                                                                           | The collection ID for the package. Applies only to an embedded EXPLAIN statement executed from a package or to a statement that is explained when binding a package. Blank if not applicable.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| <b>Note:</b> The following nine columns, from ACCESS_DEGREE through CORRELATION_NAME, contain the null value if the plan or package was bound using a plan table with fewer than 43 columns. Otherwise, each of them can contain null if the method it refers to does not apply. |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| ACCESS_DEGREE                                                                                                                                                                                                                                                                    | The number of parallel tasks or operations activated by a query. This value is determined at bind time; the actual number of parallel operations used at execution time could be different. This column contains 0 if there is a host variable.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| ACCESS_PGROUP_ID                                                                                                                                                                                                                                                                 | The identifier of the parallel group for accessing the new table. A parallel group is a set of consecutive operations, executed in parallel, that have the same number of parallel tasks. This value is determined at bind time; it could change at execution time.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |

Table 56. Descriptions of columns in PLAN\_TABLE (continued)

| Column Name      | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
|------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| JOIN_DEGREE      | The number of parallel operations or tasks used in joining the composite table with the new table. This value is determined at bind time and can be 0 if there is a host variable. The actual number of parallel operations or tasks used at execution time could be different.                                                                                                                                                                                                                                                                                                                                                                                                                         |
| JOIN_PGROUP_ID   | The identifier of the parallel group for joining the composite table with the new table. This value is determined at bind time; it could change at execution time.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| SORTC_PGROUP_ID  | The parallel group identifier for the parallel sort of the composite table.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| SORTN_PGROUP_ID  | The parallel group identifier for the parallel sort of the new table.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| PARALLELISM_MODE | The kind of parallelism, if any, that is used at bind time:<br><b>I</b> Query I/O parallelism<br><b>C</b> Query CP parallelism<br><b>X</b> Sysplex query parallelism                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| MERGE_JOIN_COLS  | The number of columns that are joined during a merge scan join (Method=2).                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| CORRELATION_NAME | The correlation name of a table or view that is specified in the statement. If there is no correlation name, then the column is blank.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| PAGE_RANGE       | Whether the table qualifies for page range screening, so that plans scan only the partitions that are needed. Y = Yes; blank = No.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| JOIN_TYPE        | The type of join:<br><b>F</b> FULL OUTER JOIN<br><b>L</b> LEFT OUTER JOIN<br><b>S</b> STAR JOIN<br><b>blank</b> INNER JOIN or no join<br>RIGHT OUTER JOIN converts to a LEFT OUTER JOIN when you use it, so that JOIN_TYPE contains L.                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| GROUP_MEMBER     | The member name of the DB2 that executed EXPLAIN. The column is blank if the DB2 subsystem was not in a data sharing environment when EXPLAIN was executed.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| IBM_SERVICE_DATA | Values are for IBM use only.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| WHEN_OPTIMIZE    | When the access path was determined:<br><b>blank</b> At bind time, using a default filter factor for any host variables, parameter markers, or special registers.<br><b>B</b> At bind time, using a default filter factor for any host variables, parameter markers, or special registers; however, the statement is reoptimized at run time using input variable values for input host variables, parameter markers, or special registers. The bind option REOPT(VARS) must be specified for reoptimization to occur.<br><b>R</b> At run time, using input variables for any host variables, parameter markers, or special registers. The bind option REOPT(VARS) must be specified for this to occur. |

## EXPLAIN

Table 56. Descriptions of columns in PLAN\_TABLE (continued)

| Column Name       | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
|-------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| QBLOCK_TYPE       | For each query block, an indication of the type of SQL operation performed. For the outermost query, this column identifies the statement type. Possible values:<br><b>SELECT</b> SELECT<br><b>INSERT</b> INSERT<br><b>UPDATE</b> UPDATE<br><b>DELETE</b> DELETE<br><b>SELUPD</b> SELECT with FOR UPDATE OF<br><b>DELCUR</b> DELETE WHERE CURRENT OF CURSOR<br><b>UPDCUR</b> UPDATE WHERE CURRENT OF CURSOR<br><b>CORSUB</b> Correlated subselect or fullselect<br><b>NCOSUB</b> Noncorrelated subselect or fullselect<br><b>TABLEX</b> Table expression<br><b>UNION</b> UNION<br><b>UNIONA</b> UNION ALL |
| BIND_TIME         | The time at which the plan or package for this statement or query block was bound. For static SQL statements, this is a full-precision timestamp value. For dynamic SQL statements, this is the value contained in the TIMESTAMP column of PLAN_TABLE appended by 4 zeroes.                                                                                                                                                                                                                                                                                                                               |
| OPTHINT           | A string that you use to identify this row as an optimization hint for DB2. DB2 uses this row as input when choosing an access path.                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| HINT_USED         | If DB2 used one of your optimization hints, it puts the identifier for that hint (the value in OPTHINT) in this column.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| PRIMARY_ACESSTYPE | Indicates whether direct row access will be attempted first:<br><br><b>D</b> DB2 will try to use direct row access. If DB2 cannot use direct row access at runtime, it uses the access path described in the ACESSTYPE column of PLAN_TABLE.<br><br><b>blank</b> DB2 will not try to use direct row access.                                                                                                                                                                                                                                                                                               |
| PARENT_QBLOCKNO   | A number that indicates the QBLOCKNO of the parent query block.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| TABLE_TYPE        | The type of new table:<br><b>F</b> Table function<br><b>Q</b> Temporary intermediate result table (not materialized)<br><b>T</b> Table<br><b>W</b> Work file<br>The value of the column is null if the query uses GROUP BY, ORDER BY, or DISTINCT, which requires an implicit sort.                                                                                                                                                                                                                                                                                                                       |

**Plan table creation options:** A plan table can have a format with fewer columns than those shown in the foregoing CREATE statement. A plan table must include one of the following sets of columns:

- All the columns up to and including REMARKS (COLCOUNT = 25)
- All the columns up to and including MIXOPSEQ (COLCOUNT = 28)
- All the columns up to and including COLLID (COLCOUNT = 30)
- All the columns up to and including JOIN\_PGROUP\_ID (COLCOUNT = 34)
- All the columns up to and including IBM\_SERVICE\_DATA (COLCOUNT = 43)
- All the columns up to and including BIND\_TIME (COLCOUNT = 46)
- All the columns shown in the CREATE statement (COLCOUNT = 51)

Whichever set of columns you choose, the columns must appear in the order in which the CREATE statement indicates. You can add columns to an existing plan table with the ALTER TABLE statement only if the modified table satisfies one of the allowed sets of columns. For example, you cannot add column PREFETCH by

itself, but must add columns PREFETCH, COLUMN\_FN\_EVAL, and MIXOPSEQ. If you add any NOT NULL columns, give them the NOT NULL WITH DEFAULT attribute.

Missing columns are ignored when rows are added to a plan table.

**Plan table migration:** You can migrate existing plan tables to subsequent releases or fall back to prior releases. If you fall back to a prior release, the extra columns are simply ignored when EXPLAIN is executed. If you migrate to a subsequent release, the missing columns are likewise ignored.

**Creating a statement table:** To create a statement table, execute the following SQL statement:

```
CREATE TABLE userid.DSN_STATEMNT_TABLE
 (QUERYNO INTEGER NOT NULL WITH DEFAULT,
 APPLNAME CHAR(8) NOT NULL WITH DEFAULT,
 PROGNAME CHAR(8) NOT NULL WITH DEFAULT,
 COLLID CHAR(18) NOT NULL WITH DEFAULT,
 GROUP_MEMBER CHAR(8) NOT NULL WITH DEFAULT,
 EXPLAIN_TIME TIMESTAMP NOT NULL WITH DEFAULT,
 STMT_TYPE CHAR(6) NOT NULL WITH DEFAULT,
 COST_CATEGORY CHAR(1) NOT NULL WITH DEFAULT,
 PROCMS INTEGER NOT NULL WITH DEFAULT,
 PROCSU INTEGER NOT NULL WITH DEFAULT,
 REASON VARCHAR(254) NOT NULL WITH DEFAULT)
IN database-name.table-space-name;
```

where *database-name.table-space-name* identifies a database and table space you have authorization to use.

**Statement table column descriptions:** Table 57 explains the columns in DSN\_STATEMNT\_TABLE. The explanations apply both to rows resulting from the execution of an EXPLAIN statement and to rows resulting from a bind or rebind.

Each row in the table provides a cost estimate, in service units and milliseconds, of processing an explainable statement.

Notice that the first five columns of the table are the same as the first five columns of PLAN\_TABLE and DSN\_FUNCTION\_TABLE.

Table 57. Descriptions of columns in DSN\_STATEMNT\_TABLE

| Column name | Description                                                                                                                                                                                                         |
|-------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| QUERYNO     | A number intended to identify the statement being explained. See the description of the QUERYNO column in Table 56 on page 784 for more information. If QUERYNO is not unique, the value of EXPLAIN_TIME is unique. |
| APPLNAME    | The name of the application plan for the row, or blank. See the description of the APPLNAME column in Table 56 on page 784 for more information.                                                                    |
| PROGNAME    | The name of the program or package containing the statement being explained, or blank. See the description of the PROGNAME column in Table 56 on page 784 for more information.                                     |
| COLLID      | The collection ID for the package, or blank. See the description of the COLLID column in Table 56 on page 784 for more information.                                                                                 |

## EXPLAIN

Table 57. Descriptions of columns in DSN\_STATEMNT\_TABLE (continued)

| Column name                                                                                              | Description                                                                                                                                                                                                                                                                                                                                                                                                  |
|----------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| GROUP_MEMBER                                                                                             | The member name of the DB2 that executed EXPLAIN, or blank. See the description of the GROUP_MEMBER column in Table 56 on page 784 for more information.                                                                                                                                                                                                                                                     |
| EXPLAIN_TIME                                                                                             | The time at which the statement is processed. This time is the same as the BIND_TIME column in PLAN_TABLE.                                                                                                                                                                                                                                                                                                   |
| STMT_TYPE                                                                                                | The type of statement being explained. Possible values are:<br><b>SELECT</b> SELECT<br><b>INSERT</b> INSERT<br><b>UPDATE</b> UPDATE<br><b>DELETE</b> DELETE<br><b>SELUPD</b> SELECT with FOR UPDATE<br><b>DELCUR</b> DELETE WHERE CURRENT OF CURSOR<br><b>UPDCUR</b> UPDATE WHERE CURRENT OF CURSOR                                                                                                          |
| COST_CATEGORY                                                                                            | Indicates if DB2 was forced to use default values when making its estimates. Possible values:<br><b>A</b> Indicates that DB2 had enough information to make a cost estimate without using default values.<br><b>B</b> Indicates that some condition exists for which DB2 was forced to use default values. See the values in REASON to determine why DB2 was unable to put this estimate in cost category A. |
| PROCMS                                                                                                   | The estimated processor cost, in milliseconds, for the SQL statement. The estimate is rounded up to the next integer value. The maximum value for this cost is 2147483647 milliseconds, which is equivalent to approximately 24.8 days. If the estimated value exceeds this maximum, the maximum value is reported.                                                                                          |
| PROCSU                                                                                                   | The estimated processor cost, in service units, for the SQL statement. The estimate is rounded up to the next integer value. The maximum value for this cost is 2147483647 service units. If the estimated value exceeds this maximum, the maximum value is reported.                                                                                                                                        |
| REASON                                                                                                   | A string that indicates the reasons for putting an estimate into cost category B.                                                                                                                                                                                                                                                                                                                            |
| <b>HOST VARIABLES</b>                                                                                    |                                                                                                                                                                                                                                                                                                                                                                                                              |
| The statement uses host variables, parameter markers, or special registers.                              |                                                                                                                                                                                                                                                                                                                                                                                                              |
| <b>TABLE CARDINALITY</b>                                                                                 |                                                                                                                                                                                                                                                                                                                                                                                                              |
| The cardinality statistics are missing for one or more of the tables used in the statement.              |                                                                                                                                                                                                                                                                                                                                                                                                              |
| UDF                                                                                                      | The statement uses user-defined functions.                                                                                                                                                                                                                                                                                                                                                                   |
| <b>TRIGGERS</b>                                                                                          |                                                                                                                                                                                                                                                                                                                                                                                                              |
| Triggers are defined on the target table of an INSERT, UPDATE, or DELETE statement.                      |                                                                                                                                                                                                                                                                                                                                                                                                              |
| <b>REFERENTIAL CONSTRAINTS</b>                                                                           |                                                                                                                                                                                                                                                                                                                                                                                                              |
| Referential constraints of the type CASCADE or SET NULL exist on the target table of a DELETE statement. |                                                                                                                                                                                                                                                                                                                                                                                                              |

**Creating a function table:** To create a function table, execute the following SQL statement:

```

CREATE TABLE DSN_FUNCTION_TABLE
 (QUERYNO INTEGER NOT NULL WITH DEFAULT,
 QBLOCKNO INTEGER NOT NULL WITH DEFAULT,
 APPLNAME CHAR(8) NOT NULL WITH DEFAULT,
 PROGNAME CHAR(8) NOT NULL WITH DEFAULT,
 COLLID CHAR(18) NOT NULL WITH DEFAULT,
 GROUP_MEMBER CHAR(8) NOT NULL WITH DEFAULT,
 EXPLAIN_TIME TIMESTAMP NOT NULL WITH DEFAULT,
 SCHEMA_NAME CHAR(8) NOT NULL WITH DEFAULT,
 FUNCTION_NAME CHAR(18) NOT NULL WITH DEFAULT,
 SPEC_FUNC_NAME CHAR(18) NOT NULL WITH DEFAULT,
 FUNCTION_TYPE CHAR(2) NOT NULL WITH DEFAULT,
 VIEW_CREATOR CHAR(8) NOT NULL WITH DEFAULT,
 VIEW_NAME CHAR(18) NOT NULL WITH DEFAULT,
 PATH VARCHAR(254) NOT NULL WITH DEFAULT,
 FUNCTION_TEXT VARCHAR(254) NOT NULL WITH DEFAULT)
IN database-name.table-space-name;

```

where *database-name.table-space-name* identifies a database and table space you have authorization to use.

**Function table column descriptions:** Table 58 explains the columns in DSN\_FUNCTION\_TABLE. The explanations apply both to rows resulting from the execution of an EXPLAIN statement and to rows resulting from a bind or rebind.

For each user-defined function that is referred to by the explainable statement, each row in the function table describes how DB2 resolved the function reference.

Notice that the first five columns of the table are the same as the first five columns of PLAN\_TABLE and DSN\_STATEMNT\_TABLE.

Table 58. Descriptions of columns in DSN\_FUNCTION\_TABLE

| Column name   | Description                                                                                                                                                                                                         |
|---------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| QUERYNO       | A number intended to identify the statement being explained. See the description of the QUERYNO column in Table 56 on page 784 for more information. If QUERYNO is not unique, the value of EXPLAIN_TIME is unique. |
| APPLNAME      | The name of the application plan for the row, or blank. See the description of the APPLNAME column in Table 56 on page 784 for more information.                                                                    |
| PROGNAME      | The name of the program or package containing the statement being explained, or blank. See the description of the PROGNAME column in Table 56 on page 784 for more information.                                     |
| COLLID        | The collection ID for the package, or blank. See the description of the COLLID column in Table 56 on page 784 for more information.                                                                                 |
| GROUP_MEMBER  | The member name of the DB2 that executed EXPLAIN, or blank. See the description of the GROUP_MEMBER column in Table 56 on page 784 for more information.                                                            |
| EXPLAIN_TIME  | The time at which the statement is processed. This time is the same as the BIND_TIME column in PLAN_TABLE.                                                                                                          |
| SCHEMA_NAME   | The schema name of the function invoked in the explained statement.                                                                                                                                                 |
| FUNCTION_NAME | The name of the function invoked in the explained statement.                                                                                                                                                        |
| SPEC_FUNC_ID  | The specific name of the function invoked in the explained statement.                                                                                                                                               |

## EXPLAIN

Table 58. Descriptions of columns in DSN\_FUNCTION\_TABLE (continued)

| Column name   | Description                                                                                                                                                                                                                                                                                                                                                                                            |
|---------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| FUNCTION_TYPE | The type of function invoked in the explained statement. Possible values are:<br><b>S</b> Scalar function<br><b>T</b> Table function                                                                                                                                                                                                                                                                   |
| VIEW_CREATOR  | If the function specified in the FUNCTION_NAME column is referenced in a view definition, the creator of the view. Otherwise, blank.                                                                                                                                                                                                                                                                   |
| VIEW_NAME     | If the function specified in the FUNCTION_NAME column is referenced in a view definition, the name of the view. Otherwise, blank.                                                                                                                                                                                                                                                                      |
| PATH          | The value of the SQL path that was used to resolve the schema name of the function.                                                                                                                                                                                                                                                                                                                    |
| FUNCTION_TEXT | The text of the function reference (the function name and parameters). If the function reference is over 100 bytes, this column contains the first 100 bytes. For functions specified in infix notation, FUNCTION_TEXT contains only the function name. For example, for a function named /, which overloads the SQL divide operator, if the function reference is A/B, FUNCTION_TEXT contains only /. |

## Examples

*Example 1:* Determine the steps required to execute the query 'SELECT X.ACTNO...'. Assume that no set of rows in the PLAN\_TABLE has the value 13 for the QUERYNO column.

```
EXPLAIN PLAN SET QUERYNO = 13
FOR SELECT X.ACTNO, X.PROJNO, X.EMPNO, Y.JOB, Y.EDLEVEL
 FROM DSN8710.EMPPROJECT X, DSN8710.EMP Y
 WHERE X.EMPNO = Y.EMPNO
 AND X.EMPTIME > 0.5
 AND (Y.JOB = 'DESIGNER' OR Y.EDLEVEL >= 12)
 ORDER BY X.ACTNO, X.PROJNO;
```

*Example 2:* Retrieve the information returned in Example 1. Assume that a statement table exists, so also retrieve the estimated cost of processing the query. Use the following query, which joins the plan table and the statement table.

```
SELECT * FROM PLAN_TABLE A, DSN_STATEMNT_TABLE B
 WHERE A.QUERYNO = 13 and B.QUERYNO = 13
 ORDER BY A.QBLOCKNO, A.PLANNO, A.MIXOPSEQ;
```

## FETCH

The FETCH statement positions a cursor on the next row of its result table and assigns the values of that row to host variables.

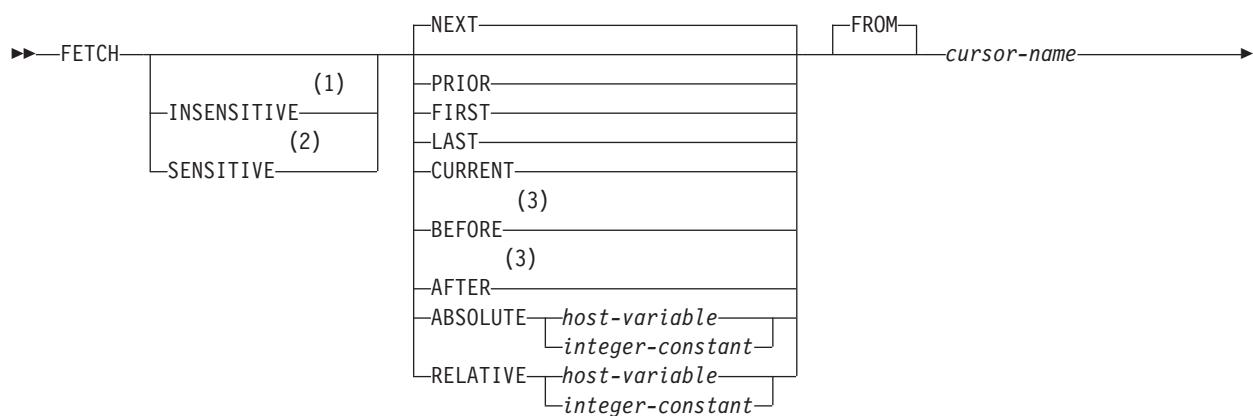
### Invocation

This statement can only be embedded in an application program. It is an executable statement that cannot be dynamically prepared.

### Authorization

See “DECLARE CURSOR” on page 718 for an explanation of the authorization required to use a cursor.

### Syntax



#### single-fetch-clause:



#### Notes:

- 1 The default depends on the sensitivity of the cursor. If INSENSITIVE is specified on the DECLARE CURSOR, then the default is INSENSITIVE and if SENSITIVE is specified on the DECLARE CURSOR, then the default is SENSITIVE.
- 2 If INSENSITIVE or SENSITIVE is specified, a *single-fetch-clause* must be specified.
- 3 If BEFORE or AFTER is specified, a *single-fetch-clause*, SENSITIVE, or INSENSITIVE must not be specified.
- 4 "USING DESCRIPTOR" may be used as a synonym for "INTO DESCRIPTOR".

## **FETCH**

### **Description**

#### **INSENSITIVE**

Returns the row from the result table as it is. If the row has been previously fetched with a `FETCH SENSITIVE`, it reflects changes made outside this cursor before the `FETCH SENSITIVE` statement was issued. Positioned updates and deletes are reflected with `FETCH INSENSITIVE` if the same cursor was used for the positioned update or delete.

`INSENSITIVE` can be specified for a cursor defined as `INSENSITIVE` or `SENSITIVE`. For an `INSENSITIVE` cursor, specifying `INSENSITIVE` is optional because it is the default.

#### **SENSITIVE**

Updates the fetched row in the result table from the corresponding row in the base table of the cursor's `SELECT` statement and returns the current values. Thus, it reflects changes made outside this cursor. `SENSITIVE` can only be specified for cursors that have been defined with the `SENSITIVE STATIC SCROLL` keywords; otherwise, an error occurs and the `FETCH` has no effect. For a `SENSITIVE` cursor, specifying `SENSITIVE` is optional because it is the default.

When a `FETCH SENSITIVE` is requested, the following steps are taken:

1. DB2 retrieves the row of the database that corresponds to the row of the result table that is about to be fetched.
2. If the corresponding row has been deleted, a *delete hole* occurs in the result table, a warning is issued, the cursor is repositioned on the hole, and no data is fetched. (DB2 marks a row in the result table as a delete hole when the corresponding row in the database is deleted.)
3. If the corresponding row has not been deleted, the predicate of the underlying `SELECT` statement is re-evaluated. If the row no longer satisfies the predicate, an *update hole* occurs in the result table, a warning is issued, the cursor is repositioned on the hole, and no data is fetched. (DB2 marks a row in the result table as an update hole when an update to the corresponding row in the database causes the row to no longer qualify for the result table.)
4. If the corresponding row does not result in a delete or an update hole in the result table, the cursor is repositioned on the row of the result table and the data is fetched.

#### **NEXT**

Positions the cursor on the next row of the result table relative to the current cursor position and fetches the row. `NEXT` is the default if no other cursor positioning is specified. `NEXT` may be used for cursors that have not been declared `SCROLL`.

Table 59 on page 795 lists situations in which `NEXT` does not result in positioning the cursor on the next row.

*Table 59. Situations in which NEXT does not position the cursor on the next row*

| <b>Current state of the cursor</b>    | <b>Result of FETCH NEXT</b>                                                                                                                |
|---------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------|
| Before the first row                  | Cursor is positioned on the first row (1).                                                                                                 |
| On the last row or after the last row | A warning occurs, values are not assigned to host variables, and the cursor is positioned after the last row.                              |
| Before a hole                         | A warning occurs for a delete hole or an update hole, values are not assigned to host variables, and the cursor is positioned on the hole. |
| Unknown                               | An error occurs, values are not assigned to host variables, and the cursor position remains unknown.                                       |

**Note:**

(1)This row is not applicable in the case of a forward-only cursor (that is when NO SCROLL was specified implicitly or explicitly).

**PRIOR**

Positions the cursor on the previous row of the result table relative to the current cursor position and fetches the row.

Table 60 lists situations in which PRIOR does not result in positioning the cursor on the previous row.

*Table 60. Situations in which PRIOR does not position the cursor on the previous row*

| <b>Current state of the cursor</b>       | <b>Result of FETCH PRIOR</b>                                                                                                               |
|------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------|
| Before the first row or on the first row | A warning occurs, values are not assigned to host variables, and the cursor is positioned before the first row.                            |
| After a hole                             | A warning occurs for a delete hole or an update hole, values are not assigned to host variables, and the cursor is positioned on the hole. |
| Unknown                                  | An error occurs, values are not assigned to host variables, and the cursor position remains unknown.                                       |

**FIRST**

Positions the cursor on the first row of the result table and fetches the row. If the first row of the result table is a hole, a warning occurs for a delete hole or an update hole and values are not assigned to host variables.

**LAST**

Positions the cursor on the last row of the result table and fetches the row. The number of rows of the result table is returned in the SQLERRD1 and SQLERRD2 fields of the SQLCA. If the last row of the result table is a hole, a warning occurs for a delete hole or an update hole and values are not assigned to host variables.

**CURRENT**

Fetches the current row. Does not reposition the cursor, but maintains current cursor position.

## FETCH

Table 61 lists situations in which errors occur with CURRENT.

Table 61. Situations in which errors occur with CURRENT

| Current state of the cursor                | Result of FETCH CURRENT                                                                                                                    |
|--------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------|
| Before the first row or after the last row | A warning occurs, values are not assigned to host variables.                                                                               |
| On a hole                                  | A warning occurs for a delete hole or an update hole, values are not assigned to host variables, and the cursor is positioned on the hole. |
| Unknown                                    | An error occurs, values are not assigned to host variables, and the cursor position remains unknown.                                       |

### BEFORE

Positions the cursor before the first row of the result table. Values are not assigned to host variables.

### AFTER

Positions the cursor after the last row of the result table. Values are not assigned to host variables. If the resulting cursor position is after the last row, the number of rows of the result table are returned in the SQLERRD1 and SQLERRD2 fields of the SQLCA.

### ABSOLUTE

Evaluates *host-variable* or *integer-constant* to an integral value *k*. If a *host-variable* is specified, it must be an exact numeric type with zero scale and must not include an indicator variable. The possible data types for the host variable are DECIMAL(*n*,0) or INTEGER. The DECIMAL data type is limited to DECIMAL(18,0). An *integer-constant* can be up to 31 digits, depending on the application language.

If *k*=0, the cursor is positioned before the first row of the result table. Otherwise, ABSOLUTE positions the cursor to row *k* of the result table if *k*>0, or to *k* rows from the bottom of the table if *k*<0. For example, "ABSOLUTE -1" is the same as "LAST".

If an absolute position is specified that is before the first row or after the last row of the result table, a warning occurs, values are not assigned to host variables, and the cursor is positioned either before the first row or after the last row. If the resulting cursor position is after the last row, the number of rows of the result table are returned in the SQLERRD1 and SQLERRD2 fields of the SQLCA. If row *k* of the result table is a hole, a warning occurs and values are not assigned to host variables.

FETCH ABSOLUTE 0 results in positioning before the first row and a warning is issued. FETCH BEFORE results in positioning before the first row and no warning is issued.

Table 62 lists some synonymous specifications.

*Table 62. Synonymous Scroll Specifications for ABSOLUTE*

| Specification                   | Alternative                |
|---------------------------------|----------------------------|
| ABSOLUTE 0 (but with a warning) | BEFORE (without a warning) |
| ABSOLUTE +1                     | FIRST                      |
| ABSOLUTE -1                     | LAST                       |
| ABSOLUTE $-m$ , $0 < m \leq n$  | ABSOLUTE $n+1-m$           |
| ABSOLUTE $n$                    | LAST                       |
| ABSOLUTE $-n$                   | FIRST                      |
| ABSOLUTE $x$ (with a warning)   | AFTER (without a warning)  |
| ABSOLUTE $-x$ (with a warning)  | BEFORE (without a warning) |

**Note:**

Assume:  $0 \leq m \leq n < x$  Where,  $n$  is the number of rows in the result table.

## RELATIVE

Evaluates *host-variable* or *integer-constant* to an integral value  $k$ . If a *host-variable* is specified, it must be an exact numeric type with zero scale and must not include an indicator variable. The possible data types for the host variable are DECIMAL( $n,0$ ) or INTEGER. The DECIMAL data type is limited to DECIMAL(18,0). An *integer-constant* can be up to 31 digits, depending on the application language.

RELATIVE positions the cursor to the row in the result table that is either  $k$  rows after the current row if  $k > 0$ , or  $ABS(k)$  rows before the current row if  $k < 0$ . For example, "RELATIVE -1" is the same as "PRIOR". If  $k=0$ , the position of the cursor does not change (that is, "RELATIVE 0" is the same as "CURRENT").

If a relative position is specified that results in positioning before the first row or after the last row, a warning is issued, values are not assigned to host variables, and the cursor is positioned either before the first row or after the last row. If the resulting cursor position is after the last row, the number of rows of the result table is returned in the SQLERRD1 and SQLERRD2 fields of the SQLCA. If the cursor is positioned on a hole and RELATIVE 0 is specified or if the target row is a hole, a warning occurs and values are not assigned to host variables. If the cursor position is unknown and RELATIVE 0 is specified, an error occurs.

Table 63 lists some synonymous specifications.

*Table 63. Synonymous Scroll Specifications for RELATIVE*

| Specification                  | Alternative                |
|--------------------------------|----------------------------|
| RELATIVE +1                    | NEXT                       |
| RELATIVE -1                    | PRIOR                      |
| RELATIVE 0                     | CURRENT                    |
| RELATIVE $+r$ (with a warning) | AFTER (without a warning)  |
| RELATIVE $-r$ (with a warning) | BEFORE (without a warning) |

**Note:**

$r$  has to be large enough to position the cursor beyond either end of the result table.

## FETCH

### *cursor-name*

Identifies the cursor to be used in the fetch operation. The cursor name must identify a declared cursor, as explained in the description of the DECLARE CURSOR statement in “Notes” on page 721, or an allocated cursor, as explained in “ALLOCATE CURSOR” on page 386. When the FETCH statement is executed, the cursor must be in the open state.

### *single-fetch-clause*

When *single-fetch-clause* is specified, SENSITIVE or INSENSITIVE must be specified. The single-fetch-clause must not be specified when FETCH BEFORE or FETCH AFTER position option is specified and it must be specified with all other position options of FETCH. If an individual fetch operation causes the cursor to be positioned or to remain positioned on a row, the values of the result table are assigned to host variables as specified by the *single-fetch-clause*.

### **INTO** *host-variable*,...

Specifies a list of host variables. Each *host-variable* must identify a structure or variable that is described in the application program in accordance with the rules for declaring host structures and variables. In the operational form of INTO, a reference to a structure is replaced by a reference to each of its variables. The first value in the result row is assigned to the first host variable, the second value to the second host variable, and so on.

### **INTO DESCRIPTOR** *descriptor-name*

Identifies an SQLDA that contains a valid description of the host output variables. Result values from the associated SELECT statement are returned to the application program in the output host variables.

Before the FETCH statement is processed, the user must set the following fields in the SQLDA:

- SQLN to indicate the number of SQLVAR occurrences provided in the SQLDA  
A REXX SQLDA does not contain this field.
- SQLD to indicate the number of variables used in the SQLDA when processing the statement
- SQLVAR occurrences to indicate the attributes of the variables

The SQLDA must have enough storage to contain all SQLVAR occurrences. Each SQLVAR occurrence describes a host variable or buffer into which a value in the result set is to be assigned. If LOBs are present in the results, there must be additional SQLVAR entries for each column of the result table. If the result table contains only base types and distinct types, multiple SQLVAR entries are not needed for each column. However, extra SQLVAR entries are needed for distinct types as well as for LOBs in DESCRIBE and PREPARE INTO statements. For more information on the SQLDA, which includes a description of the SQLVAR and an explanation on how to determine the number of SQLVAR occurrences, see “SQL descriptor area (SQLDA)” on page 986.

SQLD must be set to a value greater than or equal to zero and less than or equal to SQLN.

See “Identifying an SQLDA in C or C++” on page 1003 for how to represent *descriptor-name* in C.

## Notes

**Assignment to host variables:** The data type of a host variable must be compatible with its corresponding value. If the value is numeric, the variable must have the capacity to represent the whole part of the value. For a datetime value, the variable must be a character string variable of a minimum length as defined in “String representations of datetime values” on page 57. If the value is null, an indicator variable must be specified.

Assignments are made in sequence through the list. Each assignment to a variable is made according to the rules described in Chapter 2, “Language elements,” on page 27. If the number of variables is less than the number of values in the row, the SQLWARN3 field of the SQLCA is set to W. If an assignment error occurs, the value is not assigned to the variable and no more values are assigned to variables. Any values that have already been assigned to variables remain assigned.

**Providing indicator variable for error condition:** If an error occurs as the result of an arithmetic expression in the SELECT list of an outer SELECT statement (division by zero, or overflow) or a numeric conversion error occurs, the result is the null value. As in any other case of a null value, an indicator variable must be provided and the main variable is unchanged. In this case, however, the indicator variable is set to -2. Processing of the statement continues as if the error had not occurred. (However, this error causes a positive SQLCODE.) If you do not provide an indicator variable, a negative value is returned in the SQLCODE field of the SQLCA. Processing of the statement terminates when the error is encountered. No value is assigned to the host variable or to later variables, though any values that have already been assigned to variables remain assigned.

**Cursor positioning:** An open cursor has three possible positions:

- Before a row
- On a row
- After the last row

When a scrollable or non-scrollable cursor is opened, it is positioned before the first row in the result table. If a cursor is on a row, that row is called the current row of the cursor. A cursor referred to in an UPDATE or DELETE statement must be positioned on a row. A cursor can only be on a row as a result of a FETCH statement. If the cursor was declared SENSITIVE STATIC SCROLL, the row may be a *hole*, from which no values may be fetched.

The current row of a cursor cannot be updated or deleted by another application process if it is locked. Unless it is already locked because it was inserted or updated by the application process during the current unit of work, the current row of a cursor is not locked if:

- The isolation level is UR, or
- The isolation level is CS, and
  - The result table of the cursor is read-only
  - The bind option CURRENTDATA(NO) is in effect

For scrollable cursors, the cursor position after an error varies depending on the type of error:

- When an operation is attempted against an update or delete hole, or when an update or delete hole is detected, the cursor is positioned on the hole.
- When a FETCH operation is attempted past the end of file, the cursor is positioned after the last row.

## FETCH

- When a FETCH operation is attempted before the beginning of file, the cursor is positioned before the first row.
- When an error causes the cursor position to be invalid such as when a positioned update or positioned delete error occurs that causes a rollback, the cursor is closed.

**Cursor position after exception condition:** If an error occurs during the execution of a fetch operation, the position of the cursor and the result of any later fetch is unpredictable. It is possible for an error to occur that makes the position of the cursor invalid, in which case the cursor is closed.

If an individual fetch operation specifies a destination that is outside the range of the cursor, a warning is issued (except for FETCH BEFORE or FETCH AFTER), the cursor is positioned before or after the result table, and values are not assigned to host variables.

**Sensitivity of SENSITIVE STATIC SCROLL cursors to database changes:** When SENSITIVE STATIC SCROLL has been declared, the following rules apply:

- For the result of an update operation to be visible within a cursor after *open*, the update operation must be a positioned update executed against the cursor, or a FETCH SENSITIVE in a STATIC cursor must be executed against a row which has been updated by some other means (that is, a searched update, committed updates of others, or an update with another cursor in the same process).
- Another process can update the base table of the SELECT statement so that the current values no longer satisfy the WHERE clause. In this case, an *update hole* effectively exists during the time the values in the base table do not satisfy the WHERE clause, and the row is no longer accessible through the cursor. When an attempt is made to fetch a row that has been identified as an update hole, no values are returned, and a warning is issued.

Under SENSITIVE STATIC SCROLL cursors, update holes are only identified during positioned update, positioned delete, and FETCH SENSITIVE operations. Each positioned update, positioned delete, and FETCH SENSITIVE operation does the necessary tests to determine if an update hole exists.

- For the result of a delete operation to be visible within a SENSITIVE STATIC SCROLL cursor, the delete operation must be a positioned delete executed against the cursor or a FETCH SENSITIVE in a STATIC cursor must be executed against a row that has been deleted by some other means (that is, a searched delete, committed deletes of others, or a delete with another cursor in the same process).
- Another process, or the even the same process, may delete a row in the base table of the SELECT statement so that a row of the cursor no longer has a corresponding row in the base table. In this case, a *delete hole* effectively exists, and that row is no longer accessible through the cursor. When an attempt is made to fetch a row that has been identified as a delete hole, no values are returned, and a warning is issued.

Under SENSITIVE STATIC SCROLL cursors, delete holes are identified during positioned update, positioned delete, and FETCH SENSITIVE operations.

- Inserts into the base table or tables of SENSITIVE STATIC SCROLL cursors are not seen after the cursor is opened.

**LOB locators:** When information is retrieved into LOB locators and it is not necessary to retain the locator across FETCH statements, it is a good practice to issue a FREE LOCATOR statement before issuing another FETCH statement because locator resources are limited.

## Example

The FETCH statement fetches the results of the SELECT statement into the application program variables DNUM, DNAME, and MNUM. When no more rows remain to be fetched, the not found condition is returned.

```
EXEC SQL DECLARE C1 CURSOR FOR
 SELECT DEPTNO, DEPTNAME, MGRNO FROM DSN8710.DEPT
 WHERE ADMRDEPT = 'A00';

EXEC SQL OPEN C1;

DO WHILE (SQLCODE = 0);
 EXEC SQL FETCH C1 INTO :DNUM, :DNAME, :MNUM;

END;

EXEC SQL CLOSE C1;
```

## FREE LOCATOR

### FREE LOCATOR

The FREE LOCATOR statement removes the association between a LOB locator variable and its value.

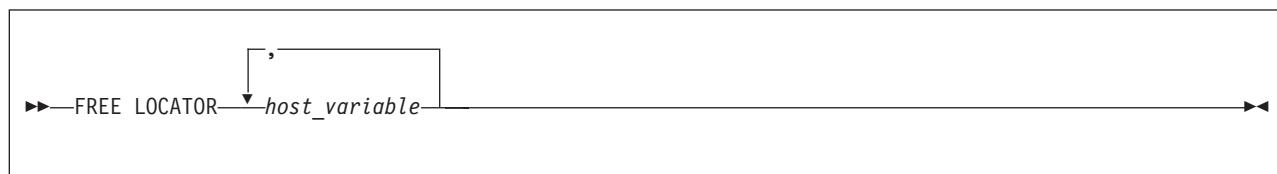
#### Invocation

This statement can only be embedded in an application program. It cannot be issued interactively. It is an executable statement that can be dynamically prepared. However, the EXECUTE statement with the USING clause must be used to execute the prepared statement. FREE LOCATOR cannot be used with the EXECUTE IMMEDIATE statement.

#### Authorization

None required.

#### Syntax



#### Description

*host\_variable*,...

Identifies a *host-variable* locator variable that must have been previously declared according to the rules for declaring host-variable locator variables. The locator variable type must be a binary large object locator, a character large object locator, or a double-byte character large object locator.

After the FREE LOCATOR statement is executed, each locator variable in the host-variable list is no longer associated with the string value it represented.

If a locator variable is not an established locator within the current unit of work, an invalid locator error occurs. When this error occurs and more than one host variable was specified in the FREE LOCATOR statement, only the locators up to the first invalid locator are freed. Locators listed after the first invalid locator are not freed.

#### Example

Assume that the employee table contains columns RESUME, HISTORY, and PICTURE and that locators have been established in a program to represent the column values. Free the CLOB locator variables LOCRES and LOCHIST, and the BLOB locator variable LOCPIC.

```
EXEC SQL FREE LOCATOR :LOCRES, :LOCHIST, :LOCPIC
```

---

## GRANT

The GRANT statement grants privileges to authorization IDs. There is a separate form of the statement for each of these classes of privilege:

- Collection
- Database
- Distinct type
- Function or stored procedure
- Package
- Plan
- Schema
- System
- Table or view
- Use

The applicable objects are always at the current server. The grants are recorded in the current server's catalog.

### Invocation

This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is implicitly or explicitly specified.

If the authorization mechanism was not activated when the DB2 subsystem was installed, an error condition occurs.

### Authorization

To grant a privilege P, the privilege set must include one of the following:

- The privilege P WITH GRANT OPTION
- Ownership of the object on which P is a privilege
- SYSADM authority

The presence of SYSCTRL authority in the privilege set allows the granting of all authorities except:

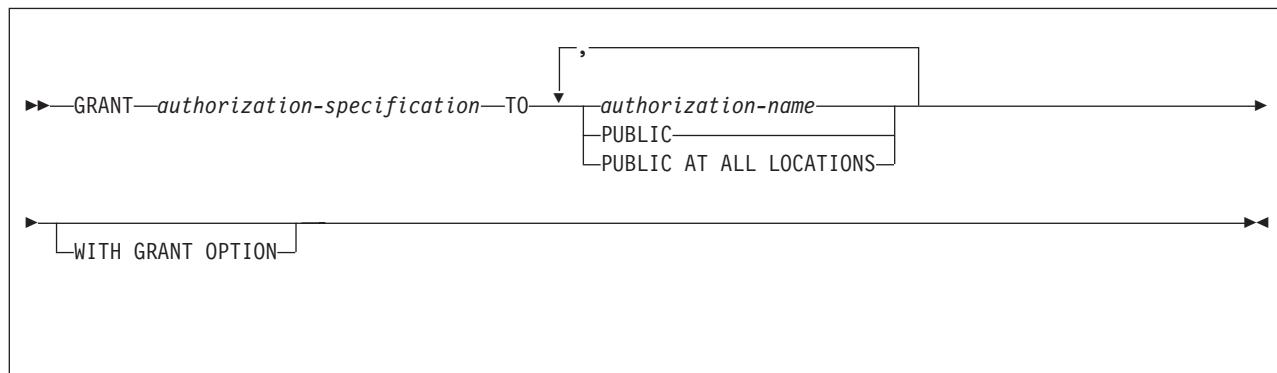
- DBADM on databases
- DELETE, INSERT, SELECT, and UPDATE on user tables or views
- EXECUTE on plans, packages, functions, or stored procedures
- PACKADM on collections
- SYSADM authority
- USAGE on distinct types and JARs

#

Except for views, the GRANT option for privileges on a table is also inherent in DBADM authority for its database, provided DBADM authority was acquired with the GRANT option. See "CREATE VIEW" on page 711 for a description of the rules that apply to views.

If the statement is embedded in an application program, the privilege set is the privileges that are held by the authorization ID of the owner of the plan or package. If the statement is dynamically prepared, the privilege set is the privileges that are held by the SQL authorization ID of the process.

## Syntax



## Description

### *authorization-specification*

Names one or more privileges in one of the formats described below. The same privilege must not be specified more than once.

### TO

Specifies to what authorization IDs the privileges are granted.

### *authorization-name*,...

Lists one or more authorization IDs.

The value of CURRENT RULES determines whether you can use the ID of the GRANT statement itself (to grant privileges to yourself). When CURRENT RULES is:

#### DB2

You cannot use the ID of the GRANT statement.

#### STD

You can use the ID of the GRANT statement.

### PUBLIC

Grants the privileges to all users at the current server, including database requesters using DRDA access.

### PUBLIC AT ALL LOCATIONS

Grants the privileges to all users in the network. Applies to table privileges only, excluding ALTER, INDEX, REFERENCES, and TRIGGER.

PUBLIC AT ALL LOCATIONS applies to DB2 private protocol access only.

### WITH GRANT OPTION

Allows the named users to grant the privileges to others. Granting an administrative authority with this option allows the user to specifically grant any privilege belonging to that authority. If you omit WITH GRANT OPTION, the named users cannot grant the privileges to others unless they have that authority from some other source.

GRANT authority cannot be passed to PUBLIC or to PUBLIC AT ALL LOCATIONS. When WITH GRANT OPTION is used with either of these, a warning is issued, and the named privileges are granted, but without GRANT authority.

## Notes

For more on DB2 privileges, read Part 3 (Volume 1) of *DB2 Administration Guide*. For information on access control authorization, see Appendix B (Volume 2) of *DB2 Administration Guide*.

A *grant* is the granting of a specific privilege by a specific grantor to a specific grantee. The grantor for a given GRANT statement is the authorization ID for the privilege set; that is, the SQL authorization ID of the process or the authorization ID of the owner of the plan or package. The grantee, as recorded in the catalog, is an authorization ID, PUBLIC, or PUBLIC\*, where PUBLIC\* denotes PUBLIC AT ALL LOCATIONS.

Duplicate grants from the same grantor are not recorded in the catalog. Otherwise, the result of executing a GRANT statement is recorded as one or more grants in the current server's catalog.

If more than one privilege or *authorization-name* is specified after the TO keyword and one of the grants is in error, execution of the statement is stopped and no grants are made. The status of the privilege or privileges granted is recorded in the catalog for each *authorization-name*.

Different grantors can grant the same privilege to a single grantee. The grantee retains that privilege as long as one or more of those grants are recorded in the catalog. Privileges that imply other privileges are also termed *authorities*. Grants are removed from the catalog by executing SQL REVOKE statements.

Whenever a grant is made for a database, distinct type, package, plan, schema, stored procedure, table, trigger, user-defined function, view, or USE privilege for an object that does not exist, an SQL return code is issued and the grant is not made.

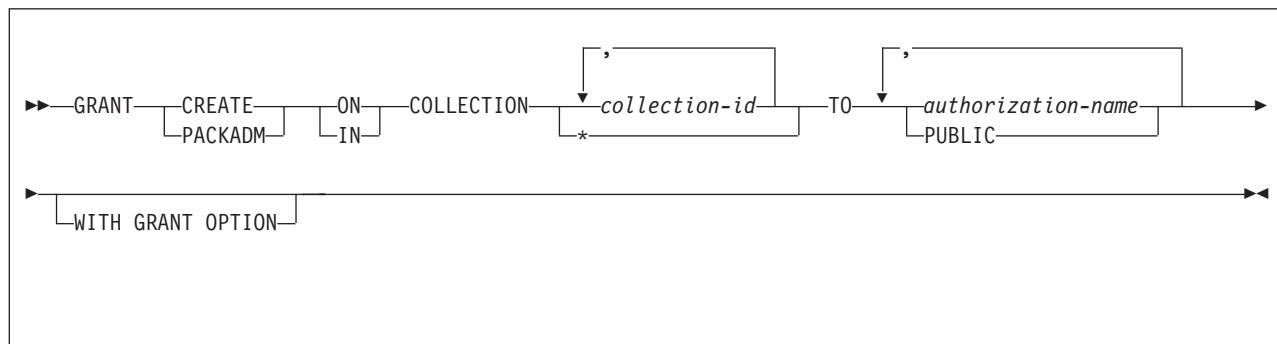
|  
|  
**PUBLIC AT ALL LOCATIONS:** PUBLIC AT ALL LOCATIONS can continue to be  
specified as an alternative to PUBLIC as in prior releases. PUBLIC AT ALL  
LOCATIONS was introduced and was intended for use only with DB2 private  
protocol access.

## GRANT (collection privileges)

### GRANT (collection privileges)

This form of the GRANT statement grants privileges on collections.

### Syntax



### Description

#### CREATE IN

Grants the privilege to use the BIND subcommand to create packages in the designated collections.

The word ON can be used instead of IN.

#### PACKADM ON

Grants package administrator authority for the designated collections.

The word IN can be used instead of ON.

#### COLLECTION *collection-id*...

Identifies the collections on which the specified privilege is granted. The collections do not have to exist.

#### COLLECTION \*

Indicates that the specified privilege is granted on all collections including those that do not currently exist.

#### TO

Refer to "GRANT" on page 803 for a description of the TO clause.

#### WITH GRANT OPTION

Refer to "GRANT" on page 803 for a description of the WITH GRANT OPTION clause.

### Example

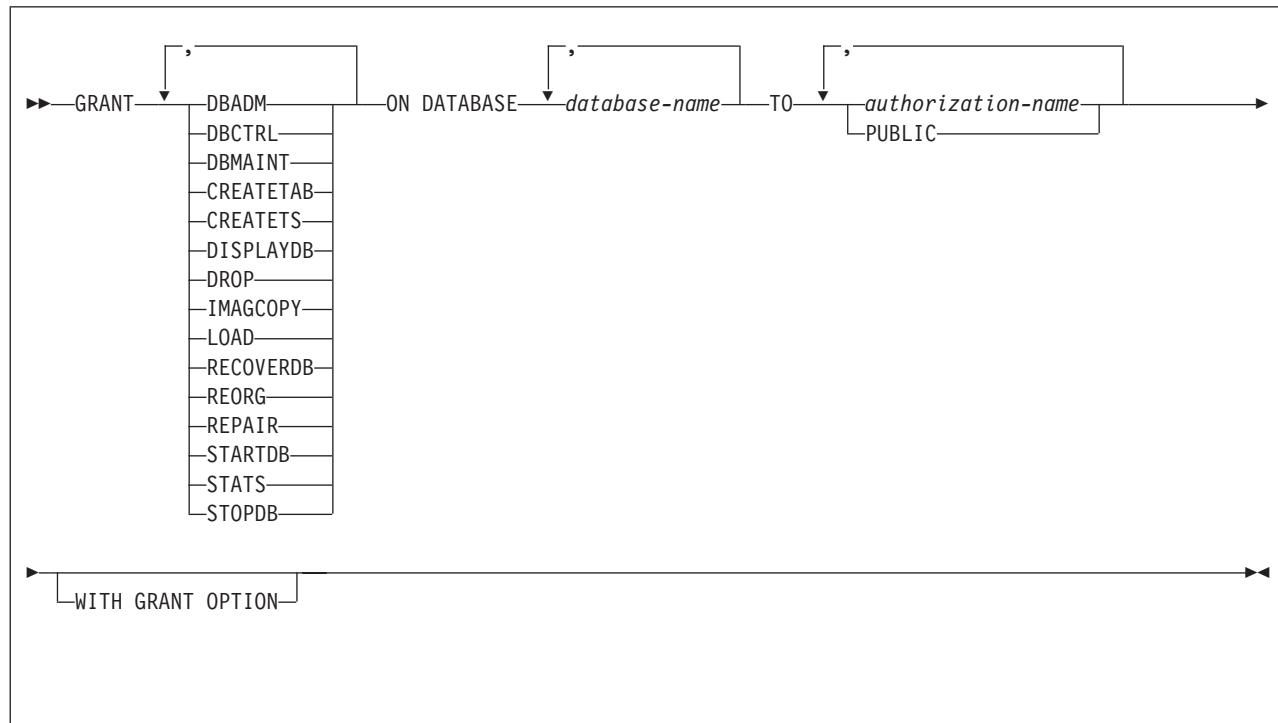
Grant the privilege to create new packages in collections QAACLONE and DSN8CC61 to CLARK.

```
GRANT CREATE IN COLLECTION QAACLONE, DSN8CC61 TO CLARK;
```

## GRANT (database privileges)

This form of the GRANT statement grants privileges on databases.

### Syntax



### Description

Each keyword listed grants the privilege described, but only as it applies to or within the databases named in the statement.

#### **DBADM**

Grants the database administrator authority.

#### **DBCTRL**

Grants the database control authority.

#### **DBMAINT**

Grants the database maintenance authority.

#### **CREATETAB**

Grants the privilege to create new tables. For a TEMP database, PUBLIC implicitly has the CREATETAB privilege (without GRANT authority) to define declared temporary tables; this privilege is not recorded in the DB2 catalog, and it cannot be revoked.

#### **CREATESTS**

Grants the privilege to create new table spaces.

#### **DISPLAYDB**

Grants the privilege to issue the DISPLAY DATABASE command.

#### **DROP**

Grants the privilege to issue the DROP or ALTER DATABASE statements for the designated databases.

## **GRANT (database privileges)**

### **IMAGCOPY**

Grants the privilege to run the COPY, MERGECOPY, and QUIESCE utilities against table spaces of the specified databases, and to run the MODIFY utility.

### **LOAD**

Grants the privilege to use the LOAD utility to load tables.

### **RECOVERDB**

Grants the privilege to use the RECOVER and REPORT utilities to recover table spaces and indexes.

### **REORG**

Grants the privilege to use the REORG utility to reorganize table spaces and indexes.

### **REPAIR**

Grants the privilege to use the REPAIR and DIAGNOSE utilities.

### **STARTDB**

Grants the privilege to issue the START DATABASE command.

### **STATS**

Grants the privilege to use the RUNSTATS utility to update statistics, and the CHECK utility to test whether indexes are consistent with the data they index.

### **STOPDB**

Grants the privilege to issue the STOP DATABASE command.

### **ON DATABASE *database-name*,...**

Identifies databases on which privileges are to be granted. For each named database, the grantor must have all the specified privileges with the GRANT option. Each name must identify a database that exists at the current server. DSNDB01 must not be identified; however, a grant of a privilege on DSNDB06 implies the granting of the same privilege on DSNDB01 for utility operations only.

### **TO**

Refer to “GRANT” on page 803 for a description of the TO clause.

### **WITH GRANT OPTION**

Refer to “GRANT” on page 803 for a description of the WITH GRANT OPTION clause.

## **Examples**

*Example 1:* Grant drop privileges on database DSN8D71A to user PEREZ.

```
GRANT DROP
 ON DATABASE DSN8D71A
 TO PEREZ;
```

*Example 2:* Grant repair privileges on database DSN8D71A to all local users.

```
GRANT REPAIR
 ON DATABASE DSN8D71A
 TO PUBLIC;
```

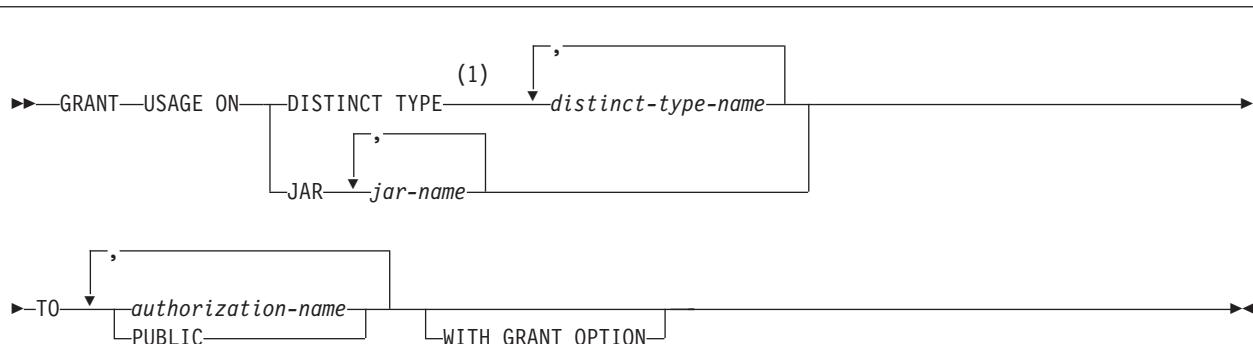
*Example 3:* Grant authority to create new tables and load tables in database DSN8D71A to users WALKER, PIANKA, and FUJIMOTO, and give them grant privileges.

```
GRANT CREATETAB,LOAD
 ON DATABASE DSN8D71A
 TO WALKER,PIANKA,FUJIMOTO
 WITH GRANT OPTION;
```

### **GRANT (distinct type or JAR privileges)**

This form of the GRANT statement grants the privilege to use distinct types (user-defined data types) or JARs.

## Syntax



## Notes:

- 1 DATA can be used as a synonym for DISTINCT. DISTINCT is the keyword that is used on CREATE.

## Description

## USAGE

Grants the privilege to use the identified distinct types or grants the privilege to use the identified JARs.

**DISTINCT TYPE** *distinct-type-name*

Identifies the distinct type. The name, including the implicit or explicit schema name, must identify a unique distinct type that exists at the current server. If you do not explicitly qualify the distinct type name, it is implicitly qualified with a schema name according to the following rules:

- If the statement is embedded in a program, the schema name is the authorization ID in the QUALIFIER option when the plan or package was created or last rebound. If QUALIFIER was not used, the schema name is the owner of the plan or package.
  - If the statement is dynamically prepared, the schema name is the SQL authorization ID in the CURRENT\_SQLID special register.

**JAR** *jar-name*

Identifies the JAR. The name, including the implicit or explicit schema name, must identify a unique JAR that exists at the current server. If you do not explicitly qualify the JAR name, it is implicitly qualified with a schema name according to the following rules:

- If the statement is embedded in a program, the schema name is the authorization ID in the QUALIFIER option when the plan or package was created or last rebound. If QUALIFIER was not used, the schema name is the owner of the plan or package.
  - If the statement is dynamically prepared, the schema name is the SQL authorization ID in the CURRENT\_SQLID special register.

## **GRANT (distinct type or JAR privileges)**

### **TO**

Refer to “GRANT” on page 803 for a description of the TO clause.

### **WITH GRANT OPTION**

Refer to “GRANT” on page 803 for a description of the WITH GRANT OPTION clause.

## **Examples**

*Example 1:* Grant the USAGE privilege on distinct type SHOE\_SIZE to user JONES. This GRANT statement does not give JONES the privilege to execute the cast functions that are associated with the distinct type SHOE\_SIZE.

```
GRANT USAGE ON DISTINCT TYPE SHOE_SIZE TO JONES;
```

*Example 2:* Grant the USAGE privilege on distinct type US\_DOLLAR to all users at the current server.

```
GRANT USAGE ON DISTINCT TYPE US_DOLLAR TO PUBLIC;
```

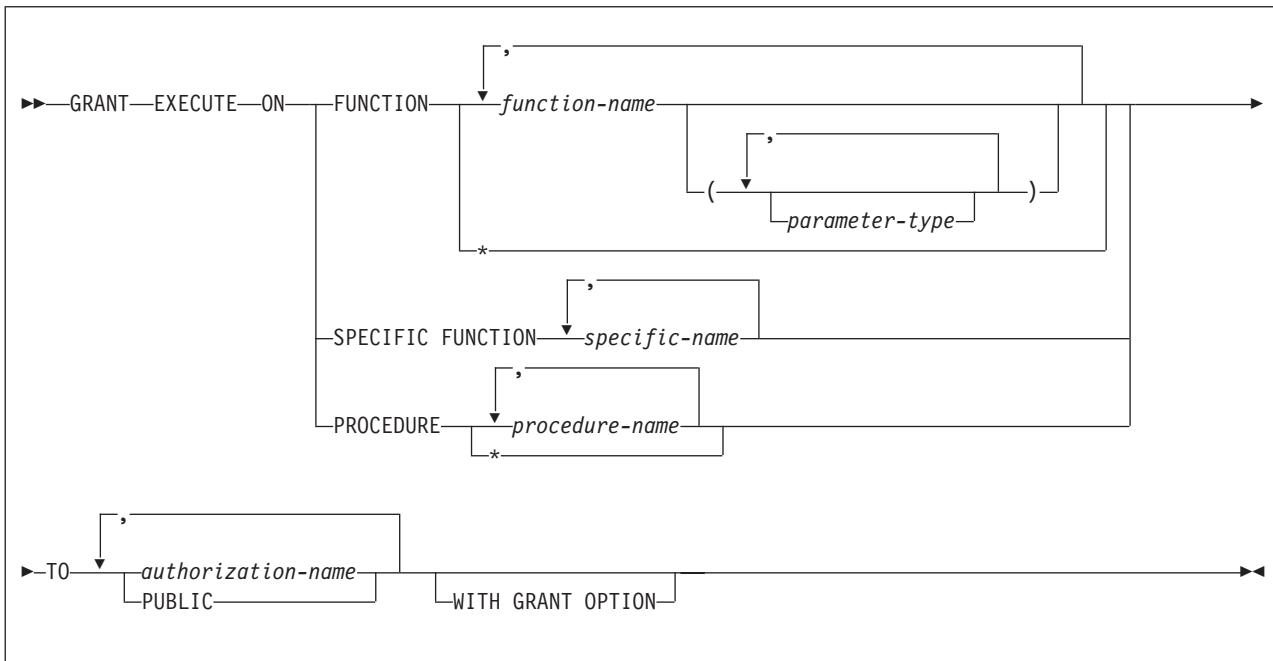
*Example 3:* Grant the USAGE privilege on distinct type CANADIAN\_DOLLAR to the administrative assistant (ADMIN\_A), and give this user the ability to grant the USAGE privilege on the distinct type to others. The administrative assistant cannot grant the privilege to execute the cast functions that are associated with the distinct type CANADIAN\_DOLLAR because WITH GRANT OPTION does not give the administrative assistant the EXECUTE authority on these cast functions.

```
GRANT USAGE ON DISTINCT TYPE CANADIAN_DOLLAR TO ADMIN_A
 WITH GRANT OPTION;
```

## GRANT (function or procedure privileges)

This form of the GRANT statement grants privileges on user-defined functions, cast functions that are generated for distinct types, and stored procedures.

### Syntax



#### parameter type:

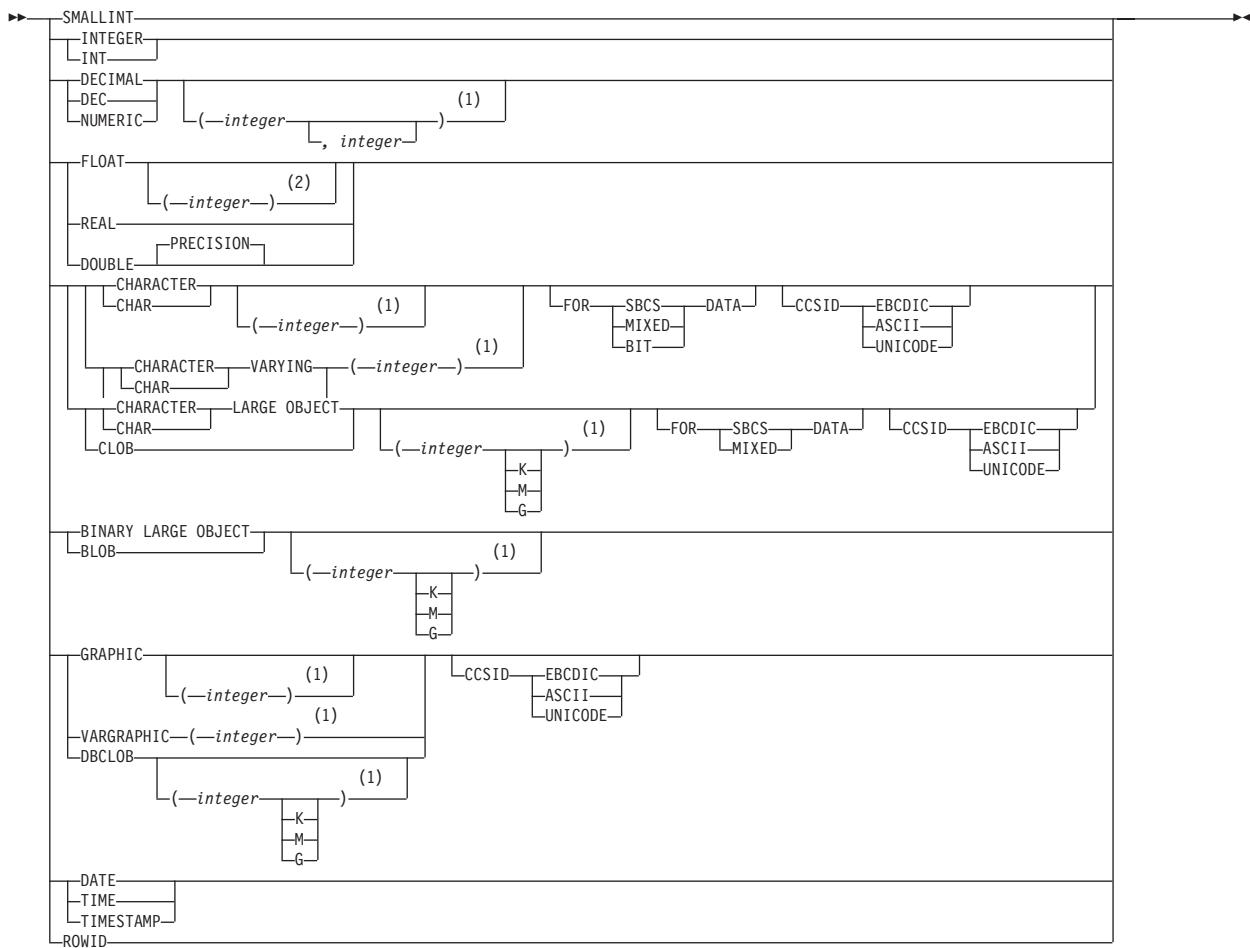


#### data type:



## GRANT (function or procedure privileges)

### built-in data type:



### Notes:

- 1 The values that are specified for length, precision, or scale attributes must match the values that were specified when the function was created. Coding specific values is optional. Empty parentheses, (), can be used instead to indicate that DB2 ignores the attributes when determining whether data types match.
- 2 The value that is specified does not have to match the value that was specified when the function was created because matching is based on data type (REAL or DOUBLE).  $1 \leq \text{integer} \leq 21$  indicates REAL and  $22 \leq \text{integer} \leq 53$  indicates DOUBLE. Coding a specific value is optional. Empty parentheses cannot be used.

## Description

### EXECUTE

Grants the privilege to run the identified user-defined function, cast function that was generated for a distinct type, or stored procedure.

### FUNCTION or SPECIFIC FUNCTION

Identifies the function on which the privilege is granted. The function must exist

at the current server, and it must be a function that was defined with the CREATE FUNCTION statement or a cast function that was generated by a CREATE DISTINCT TYPE statement.

If the function was defined with a table parameter (the LIKE TABLE was specified in the CREATE FUNCTION statement to indicate that one of the input parameters is a transition table), the function signature cannot be used to identify the function. Instead, identify the function with its function name, if unique, or with its specific name.

**FUNCTION *function-name***

Identifies the function by its name. You can identify a function by its name only if there is exactly one function with *function-name* in the schema. If you do not explicitly qualify the function name with a schema name, the function name is implicitly qualified with a schema name according to the following rules:

- If the statement is embedded in a program, the schema name is the authorization ID in the QUALIFIER option when the plan or package was created or last rebound. If QUALIFIER was not used, the schema name is the owner of the plan or package.
- If the statement is dynamically prepared, the schema name is the SQL authorization ID in the CURRENT SQLID special register.

An \* can be specified for a qualified or unqualified *function-name*. An \* (or *schema-name.\**) indicates that the privilege is granted on all the functions in the schema including those that do not currently exist. Specifying an \* does not affect any EXECUTE privileges that are already granted on a function.

**FUNCTION *function-name* (*parameter-type*,...)**

Identifies the function by its function signature, which uniquely identifies the function. If *function-name()* is specified, the function that is identified must have zero parameters.

***function-name***

Specifies the name of the function. If you do not explicitly qualify the function name with a schema name, the function name is implicitly qualified with a schema name as described in the preceding description for FUNCTION *function-name*.

**(*parameter-type*,...)**

Identifies the number of input parameters of the function and their data types.

The data type of each parameter must match the data type that was specified in the CREATE FUNCTION statement for the parameter in the corresponding position. The number of data types and the logical concatenation of the data types is used to uniquely identify the function.

For data types that have a length, precision, or scale attribute, you can specify a value or use a set of empty parentheses:

- Empty parentheses indicate that DB2 ignores the attribute when determining whether the data types match.

FLOAT cannot be specified with empty parentheses because its parameter value indicates different data types (REAL or DOUBLE).

- If you use a specific value for a length, precision, or scale attribute, the value must exactly match the value that was specified (implicitly or explicitly) in the CREATE FUNCTION statement.

#  
#

## GRANT (function or procedure privileges)

The specific value for FLOAT( $n$ ) does not have exactly match the defined value of the source function because  $1 \leq n \leq 21$  indicates REAL and  $22 \leq n \leq 53$  indicates DOUBLE. Matching is based on whether the data type is REAL or DOUBLE.

- If length, precision, or scale is not explicitly specified, and empty parentheses are not specified, the default length of the data type is implied. For example:

|                |                      |
|----------------|----------------------|
| <b>CHAR</b>    | CHAR(1)              |
| <b>GRAPHIC</b> | GRAPHIC(1)           |
| <b>DECIMAL</b> | DECIMAL(5,0)         |
| <b>FLOAT</b>   | DOUBLE (length of 8) |

The implicit length must exactly match the value that was specified (implicitly or explicitly) in the CREATE FUNCTION statement. For a complete list of the default lengths of data types, see “CREATE TABLE” on page 653.

For data types with a subtype or encoding scheme attribute, specifying the FOR DATA clause or CCSID clause is optional. Omission of either clause indicates that DB2 ignores the attribute when determining whether the data types match. If you specify either clause, it must match the value that was implicitly or explicitly specified in the CREATE FUNCTION statement.

### SPECIFIC FUNCTION *specific-name*

Identifies the function by its specific name.

### PROCEDURE *procedure-name*

Identifies a stored procedure that is defined at the current server. If you do not explicitly qualify the procedure name with a schema name, the procedure name is implicitly qualified with a schema name according to the following rules:

- If the statement is embedded in a program, the schema name is the authorization ID in the QUALIFIER option when the plan or package was created or last rebound. If QUALIFIER was not used, the schema name is the owner of the plan or package.
- If the statement is dynamically prepared, the schema name is the SQL authorization ID in the CURRENT SQLID special register.

An \* can be specified for a qualified or unqualified *procedure-name*. An \* (or *schema-name.\**) indicates that the privilege is granted on all the stored procedures in the schema including those that do not currently exist. Specifying an \* does not affect any EXECUTE privileges that are already granted on a stored procedure.

### TO

Refer to “GRANT” on page 803 for a description of the TO clause.

### WITH GRANT OPTION

Refer to “GRANT” on page 803 for a description of the WITH GRANT OPTION clause.

## Examples

*Example 1:* Grant the EXECUTE privilege on function CALC\_SALARY to user JONES. Assume that there is only one function in the schema with function name CALC\_SALARY.

```
GRANT EXECUTE ON FUNCTION CALC_SALARY TO JONES;
```

## **GRANT (function or procedure privileges)**

*Example 2:* Grant the EXECUTE privilege on procedure VACATION\_ACCR to all users at the current server.

```
GRANT EXECUTE ON PROCEDURE VACATION_ACCR TO PUBLIC;
```

*Example 3:* Grant the EXECUTE privilege on function DEPT\_TOTALS to the administrative assistant and give the assistant the ability to grant the EXECUTE privilege on this function to others. The function has the specific name DEPT85\_TOT. Assume that the schema has more than one function that is named DEPT\_TOTALS.

```
GRANT EXECUTE ON SPECIFIC FUNCTION DEPT85_TOT TO ADMIN_A
WITH GRANT OPTION;
```

*Example 4:* Grant the EXECUTE privilege on function NEW\_DEPT\_HIRES to HR (Human Resources). The function has two input parameters with data types of INTEGER and CHAR(10), respectively. Assume that the schema has more than one function that is named NEW\_DEPT\_HIRES.

```
GRANT EXECUTE ON FUNCTION NEW_DEPT_HIRES (INTEGER, CHAR(10))
TO HR;
```

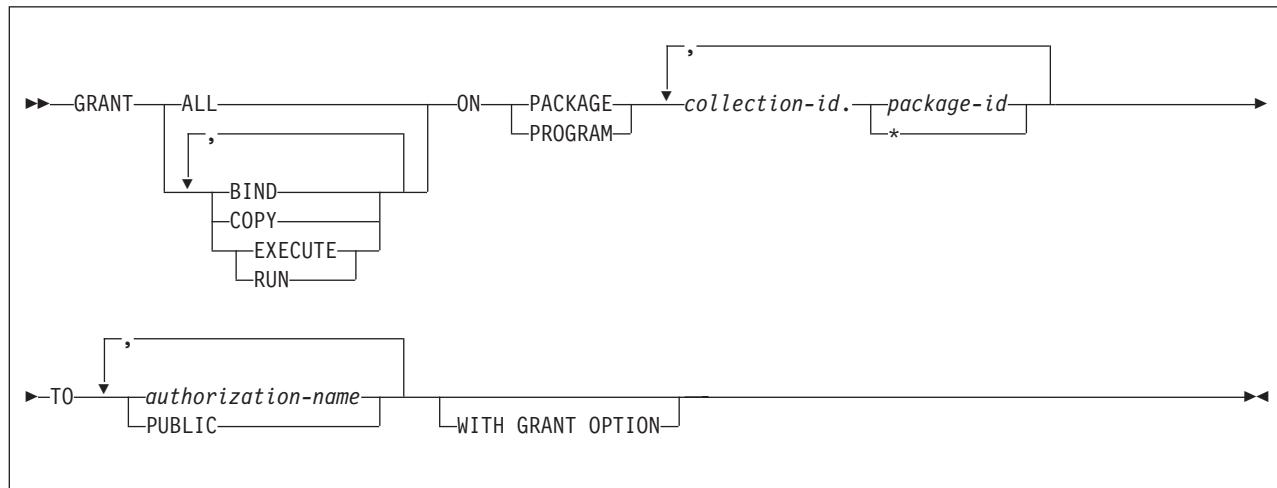
You can also code the CHAR(10) data type as CHAR().

## GRANT (package privileges)

### GRANT (package privileges)

This form of the GRANT statement grants privileges on packages.

### Syntax



### Description

#### BIND

Grants the privilege to use the BIND and REBIND subcommands for the designated packages.

The BIND package privilege is used to add a new version of an existing package. For details on the authorization required to create new packages and new versions of existing packages, see “Notes” on page 817.

#### COPY

Grants the privilege to use the COPY option of the BIND subcommand for the designated packages.

#### EXECUTE

Grants the privilege to run application programs that use the designated packages and to specify the packages following PKLIST for the BIND PLAN and REBIND PLAN commands. RUN is an alternate name for the same privilege.

#### ALL

Grants all package privileges for which you have GRANT authority for the packages named in the ON clause.

#### ON PACKAGE *collection-id.package-id*,...

Identifies packages for which you are granting privileges. The granting of a package privilege applies to all versions of a package. The list can simultaneously contain items of the following two forms:

- *collection-id.package-id* explicitly identifies a single package. The name must identify a package that exists at the current server.
- *collection-id.\** applies to every package in the indicated collection. This includes packages that currently exist and future packages. The grant applies to a collection at the current server, but the *collection-id* does not have to identify a collection that exists when the grant is made.

To grant a privilege in this form requires PACKADM with the WITH GRANT OPTION over the collection or all collections, SYSADM, or SYSCTRL authority. Because of this fact, WITH GRANT OPTION, if included in the statement, is ignored for grants of this form, but not for grants for specific packages.

The word PROGRAM can be used in place of PACKAGE.

**TO**

Refer to “GRANT” on page 803 for a description of the TO clause.

**WITH GRANT OPTION**

Refer to “GRANT” on page 803 for a description of the WITH GRANT OPTION clause.

## Notes

The authorization required to add a new package or a new version of an existing package depends on the value of field BIND NEW PACKAGE on installation panel DSNTIPP. The default value is BINDADD.

If the value of BIND NEW PACKAGE is BINDADD, the primary authorization ID must have one of the following to add a new package or a new version of an existing package to a collection:

- The BINDADD system privilege and either the CREATE IN privilege or PACKADM authority for the collection or for all collections
- SYSADM or SYSCTRL authority

If the value of BIND NEW PACKAGE is BIND, the primary authorization ID must have one of the following to add a new package or a new version of an existing package to a collection:

- The BINDADD system privilege and either the CREATE IN privilege or PACKADM authority for the collection or for all collections
- SYSADM or SYSCTRL authority
- PACKADM authority for the collection or for all collections
- The BIND package privilege (can only add a new version of an existing package)

## Examples

*Example 1:* Grant the privilege to copy all packages in collection DSN8CC61 to LEWIS.

```
GRANT COPY ON PACKAGE DSN8CC61.* TO LEWIS;
```

*Example 2:* You have the BIND privilege with GRANT authority over the package CLCT1.PKG1. You have the EXECUTE privilege with GRANT authority over the package CLCT2.PKG2. You have no other privileges with GRANT authority over any package in the collections CLCT1 AND CLCT2. Hence, the following statement, when executed by you, grants LEWIS the BIND privilege on CLCT1.PKG1 and the EXECUTE privilege on CLCT2.PKG2, and makes no other grant. The privileges granted include no GRANT authority.

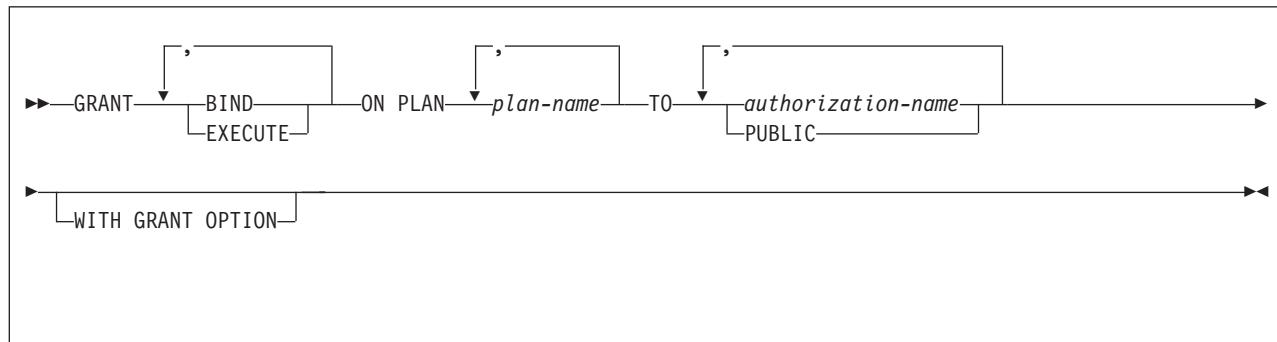
```
GRANT ALL ON PACKAGE CLCT1.PKG1, CLCT2.PKG2 TO JONES;
```

## GRANT (plan privileges)

### GRANT (plan privileges)

This form of the GRANT statement grants privileges on plans.

### Syntax



### Description

#### BIND

Grants the privilege to use the BIND, REBIND, and FREE subcommands for the identified plans. (The authority to create new plans using BIND ADD is a system privilege.)

#### EXECUTE

Grants the privilege to run programs that use the identified plans.

#### ON PLAN *plan-name*,...

Identifies the application plans on which the privileges are granted. For each identified plan, you must have all specified privileges with the GRANT option.

#### TO

Refer to "GRANT" on page 803 for a description of the TO clause.

#### WITH GRANT OPTION

Refer to "GRANT" on page 803 for a description of the WITH GRANT OPTION clause.

### Examples

*Example 1:* Grant the privilege to bind plan DSN8IP71 to user JONES.

```
GRANT BIND ON PLAN DSN8IP71 TO JONES;
```

*Example 2:* Grant privileges to bind and execute plan DSN8CP71 to all users at the current server.

```
GRANT BIND,EXECUTE ON PLAN DSN8CP71 TO PUBLIC;
```

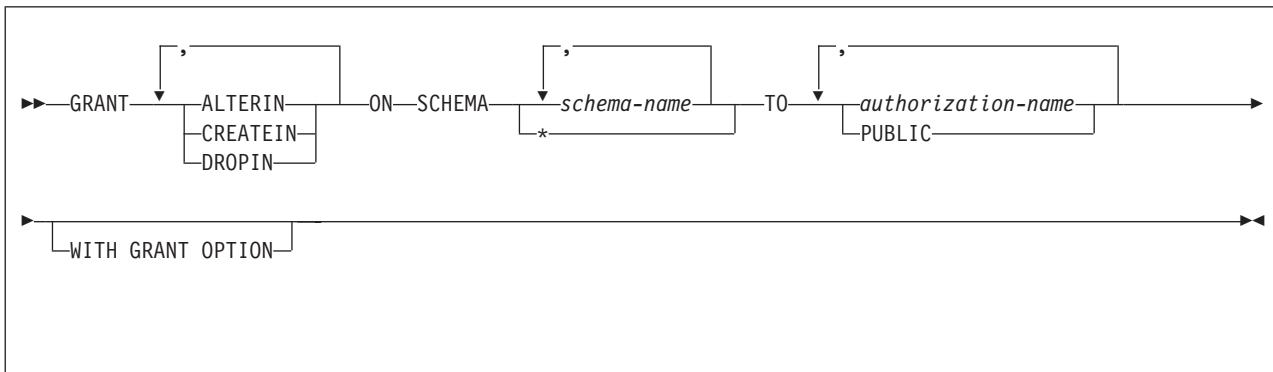
*Example 3:* Grant the privilege to execute plan DSN8CP71 to users ADAMSON and BROWN with grant option.

```
GRANT EXECUTE ON PLAN DSN8CP71 TO ADAMSON,BROWN WITH GRANT OPTION;
```

## GRANT (schema privileges)

This form of the GRANT statement grants privileges on schemas.

### Syntax



### Description

#### ALTERIN

Grants the privilege to alter stored procedures and user-defined functions, or specify a comment for distinct types, cast functions that are generated for distinct types, stored procedures, triggers, and user-defined functions in the designated schemas.

#### CREATEIN

Grants the privilege to create distinct types, stored procedures, triggers, and user-defined functions in the designated schemas.

#### DROPIN

Grants the privilege to drop distinct types, stored procedures, triggers, and user-defined functions in the designated schemas.

#### SCHEMA *schema-name*

Identifies the schemas on which the privilege is granted. The schemas do not need to exist when the privilege is granted.

#### SCHEMA \*

Indicates that the specified privilege is granted on all schemas including those that do not currently exist.

#### TO

Refer to "GRANT" on page 803 for a description of the TO clause.

#### WITH GRANT OPTION

Refer to "GRANT" on page 803 for a description of the WITH GRANT OPTION clause.

### Examples

*Example 1:* Grant the CREATEIN privilege on schema T\_SCORES to user JONES.

```
GRANT CREATEIN ON SCHEMA T_SCORES TO JONES;
```

*Example 2:* Grant the CREATEIN privilege on schema VAC to all users at the current server.

```
GRANT CREATEIN ON SCHEMA VAC TO PUBLIC;
```

## **GRANT (schema privileges)**

*Example 3:* Grant the ALTERIN privilege on schema DEPT to the administrative assistant and give the grantee the ability to grant ALTERIN privileges on this schema to others.

```
GRANT ALTERIN ON SCHEMA DEPT TO ADMIN_A
 WITH GRANT OPTION;
```

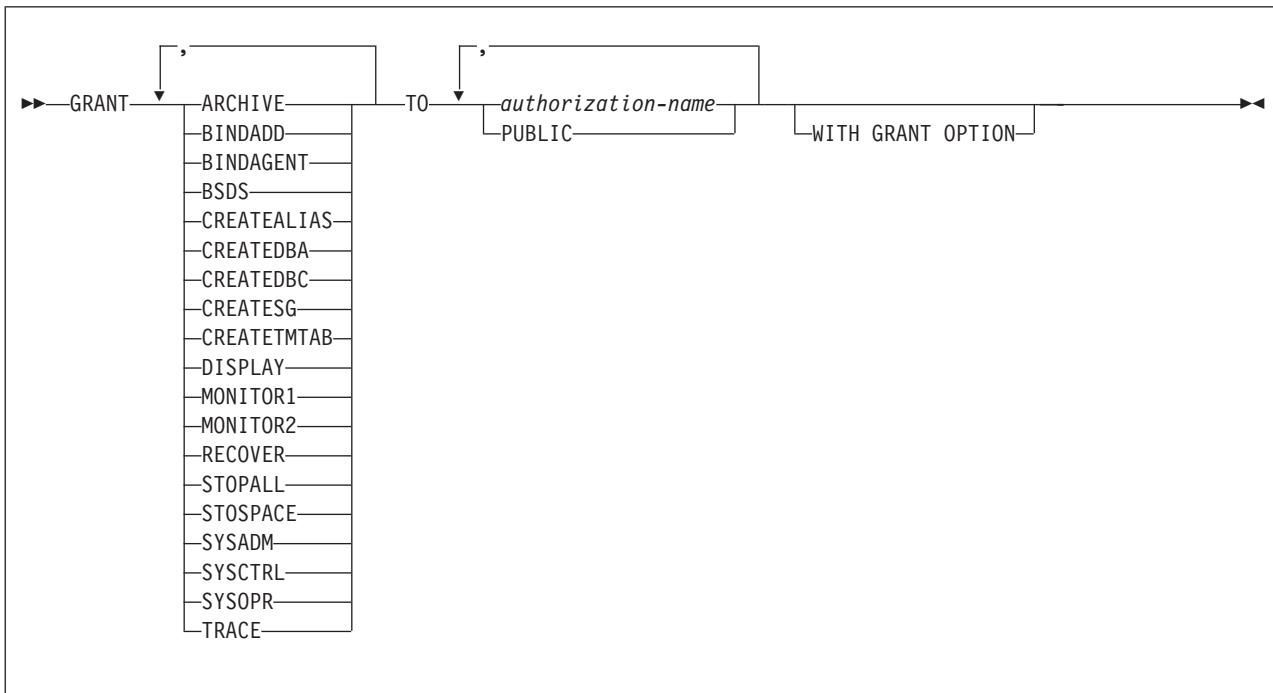
*Example 4:* Grant the CREATEIN, ALTERIN, and DROPIN privileges on schemas NEW\_HIRE, PROMO, and RESIGN to HR (Human Resources).

```
GRANT CREATEIN, ALTERIN, DROPIN ON SCHEMA NEW_HIRE, PROMO, RESIGN TO HR;
```

## GRANT (system privileges)

This form of the GRANT statement grants system privileges.

### Syntax



### Description

#### **ARCHIVE**

Grants the privilege to use the ARCHIVE LOG and SET LOG commands.

#### **BINDADD**

Grants the privilege to create plans and packages by using the BIND subcommand with the ADD option.

#### **BINDAGENT**

Grants the privilege to issue the BIND, FREE PACKAGE, or REBIND subcommands for plans and packages and the DROP PACKAGE statement on behalf of the grantor. The privilege also allows the holder to copy and replace plans and packages on behalf of the grantor.

A warning is issued if WITH GRANT OPTION is specified when granting this privilege.

#### **BSDS**

Grants the privilege to issue the RECOVER BSDS command.

#### **CREATEALIAS**

Grants the privilege to use the CREATE ALIAS statement.

#### **CREATEDBA**

Grants the privilege to issue the CREATE DATABASE statement and acquire DBADM authority over those databases.

## **GRANT (system privileges)**

### **CREATEDBC**

Grants the privilege to issue the CREATE DATABASE statement and acquire DBCTRL authority over those databases.

### **CREATESG**

Grants the privilege to create new storage groups.

### **CREATETM TAB**

Grants the privilege to use the CREATE GLOBAL TEMPORARY TABLE statement.

### **DISPLAY**

Grants the privilege to use the following commands:

- The DISPLAY ARCHIVE command for archive log information
- The DISPLAY BUFFERPOOL command for the status of buffer pools
- The DISPLAY DATABASE command for the status of all databases
- The DISPLAY LOCATION command for statistics about threads with a distributed relationship
- The DISPLAY LOG command for log information, including the status of the offload task
- The DISPLAY THREAD command for information on active threads within DB2
- The DISPLAY TRACE command for a list of active traces

### **MONITOR1**

Grants the privilege to obtain IFC data classified as serviceability data, statistics, accounting, and other performance data that does not contain potentially secure data.

### **MONITOR2**

Grants the privilege to obtain IFC data classified as containing potentially sensitive data such as SQL statement text and audit data. Users with MONITOR2 privileges have MONITOR1 privileges.

### **RECOVER**

Grants the privilege to issue the RECOVER INDOUBT command.

### **STOPALL**

Grants the privilege to issue the STOP DB2 command.

### **STOSPACE**

Grants the privilege to use the STOSPACE utility.

### **SYSADM**

Grants all DB2 privileges except for a few reserved for installation SYSADM authority. The privileges the user possesses are all grantable, including the SYSADM authority itself. The privileges the user lacks restrict what the user can do with the directory and the catalog. Using WITH GRANT OPTION when granting SYSADM is redundant but valid. For more on SYSADM and install SYSADM authority, see Part 3 (Volume 1) of *DB2 Administration Guide*.

### **SYSCTRL**

Grants the system control authority, which allows the user to have most of the privileges of a system administrator but excludes the privileges to read or change user data. Using WITH GRANT OPTION when granting SYSCTRL is redundant but valid. For more information on SYSCTRL authority, see Part 3 (Volume 1) of *DB2 Administration Guide*.

### **SYSOPR**

Grants the privilege to have system operator authority.

**TRACE**

Grants the privilege to issue the MODIFY TRACE, START TRACE, and STOP TRACE commands.

**TO**

Refer to “GRANT” on page 803 for a description of the TO clause.

**WITH GRANT OPTION**

If you grant the SYSADM or SYSCTRL system privilege, WITH GRANT OPTION is valid but unnecessary. It is unnecessary because whoever is granted SYSADM or SYSCTRL has that authority and all the privileges it implies, with the GRANT option.

## Examples

*Example 1:* Grant DISPLAY privileges to user LUTZ.

```
GRANT DISPLAY
 TO LUTZ;
```

*Example 2:* Grant BSDS and RECOVER privileges to users PARKER and SETRIGHT, with the WITH GRANT OPTION.

```
GRANT BSDS,RECOVER
 TO PARKER,SETRIGHT
 WITH GRANT OPTION;
```

*Example 3:* Grant TRACE privileges to all local users.

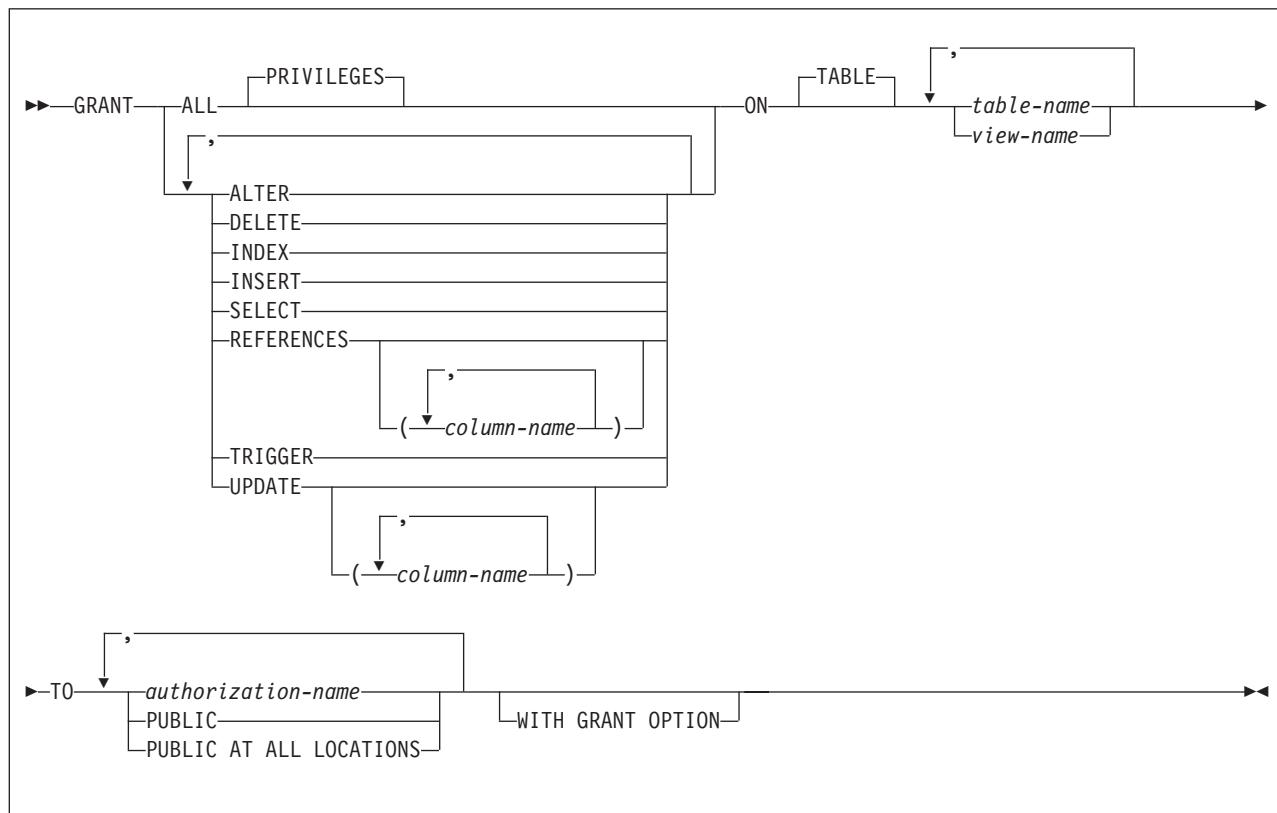
```
GRANT TRACE
 TO PUBLIC;
```

## GRANT (table or view privileges)

# GRANT (table or view privileges)

This form of the GRANT statement grants privileges on tables and views.

## Syntax



## Description

### ALL or ALL PRIVILEGES

Grants all table or view privileges for which you have GRANT authority, for the tables and views named in the ON clause. Does not include ALTER, INDEX, REFERENCES, or TRIGGER for a grant to PUBLIC AT ALL LOCATIONS.

If you do not use ALL, you must use one or more of the keywords in the following list. For each keyword that you use, you must have GRANT authority for that privilege on every table or view identified in the ON clause.

### ALTER

Grants the privilege to use the ALTER TABLE statement. ALTER cannot be granted to PUBLIC AT ALL LOCATIONS. Nor can it be used if the statement identifies an auxiliary table or a view.

### DELETE

Grants the privilege to use the DELETE statement. DELETE cannot be granted on an auxiliary table.

### INDEX

Grants the privilege to use the CREATE INDEX statement. INDEX cannot be granted to PUBLIC AT ALL LOCATIONS. Nor can it be used if the statement identifies a view.

**INSERT**

Grants the privilege to use the INSERT statement. INSERT cannot be granted on an auxiliary table.

**REFERENCES(*column-name*,...)**

Grants the privilege to define and drop a referential constraint in which the table is a parent. Grantees can create referential constraints by using all the named columns in the parent key. If a list of columns is not specified, REFERENCES applies to all the columns of every table identified in the ON clause.

REFERENCES cannot be granted to PUBLIC AT ALL LOCATIONS. Nor can it be used if the statement identifies an auxiliary table or a view.

If you specify a list of columns, each *column-name* must be the unqualified name of a column in a table identified in the ON clause.

**SELECT**

Grants the privilege to use the SELECT statement. SELECT cannot be granted on an auxiliary table.

**TRIGGER**

Grants the privilege to use the CREATE TRIGGER statement. TRIGGER cannot be granted to PUBLIC AT ALL LOCATIONS. Nor can it be used if the statement identifies an auxiliary table or a view.

**UPDATE**

Grants the privilege to use the UPDATE statement. UPDATE cannot be granted on an auxiliary table.

**UPDATE(*column-name*,...)**

Grants the privilege to use the UPDATE statement to update only the columns named. Each *column-name* must be the unqualified name of a column of every table or view identified in the ON clause. Each *column-name* must not identify a column of an auxiliary table.

**ON or ON TABLE**

Specifies the tables or views on which you are granting the privileges. The list can be a list of table names or view names, or a combination of the two. A declared temporary table must not be identified.

If you use GRANT ALL, then for each named table or view, the privilege set (described in “Authorization” in “GRANT” on page 803) must include at least one privilege with the GRANT option.

**TO**

Refer to “GRANT” on page 803 for a description of the TO clause.

**WITH GRANT OPTION**

Refer to “GRANT” on page 803 for a description of the WITH GRANT OPTION clause.

**Notes**

The REFERENCES privilege does not replace the ALTER privilege. It was added to conform to the SQL standard. To define a foreign key that references a parent table, you must have either the REFERENCES or the ALTER privilege, or both.

For a created temporary table or a view of a created temporary table, only ALL or ALL PRIVILEGES can be granted. Specific table or view privileges cannot be granted. In addition, only the ALTER, DELETE, INSERT, and SELECT privileges apply to a created temporary table.

## GRANT (table or view privileges)

For a declared temporary table, no privileges can be granted. When a declared temporary table is defined, PUBLIC implicitly receives all table privileges (without GRANT authority) for the table. These privileges are not recorded in the DB2 catalog, and they cannot be revoked.

For an auxiliary table, only the INDEX privilege can be granted. DELETE, INSERT, SELECT, and UPDATE privileges on the base table that is associated with the auxiliary table extend to the auxiliary table.

**PUBLIC AT ALL LOCATIONS:** PUBLIC AT ALL LOCATIONS can continue to be specified as an alternative to PUBLIC as in prior releases. PUBLIC AT ALL LOCATIONS was introduced and was intended for use only with DB2 private protocol access.

## Examples

*Example 1:* Grant SELECT privileges on table DSN8710.EMP to user PULASKI.

```
GRANT SELECT ON DSN8710.EMP TO PULASKI;
```

*Example 2:* Grant UPDATE privileges on columns EMPNO and WORKDEPT in table DSN8710.EMP to all users at the current server.

```
GRANT UPDATE (EMPNO,WORKDEPT) ON TABLE DSN8710.EMP TO PUBLIC;
```

*Example 3:* Grant all privileges on table DSN8710.EMP to users KWAN and THOMPSON, with the WITH GRANT OPTION.

```
GRANT ALL ON TABLE DSN8710.EMP TO KWAN,THOMPSON WITH GRANT OPTION;
```

*Example 4:* Grant the SELECT and UPDATE privileges on the table DSN8710.DEPT to every user in the network.

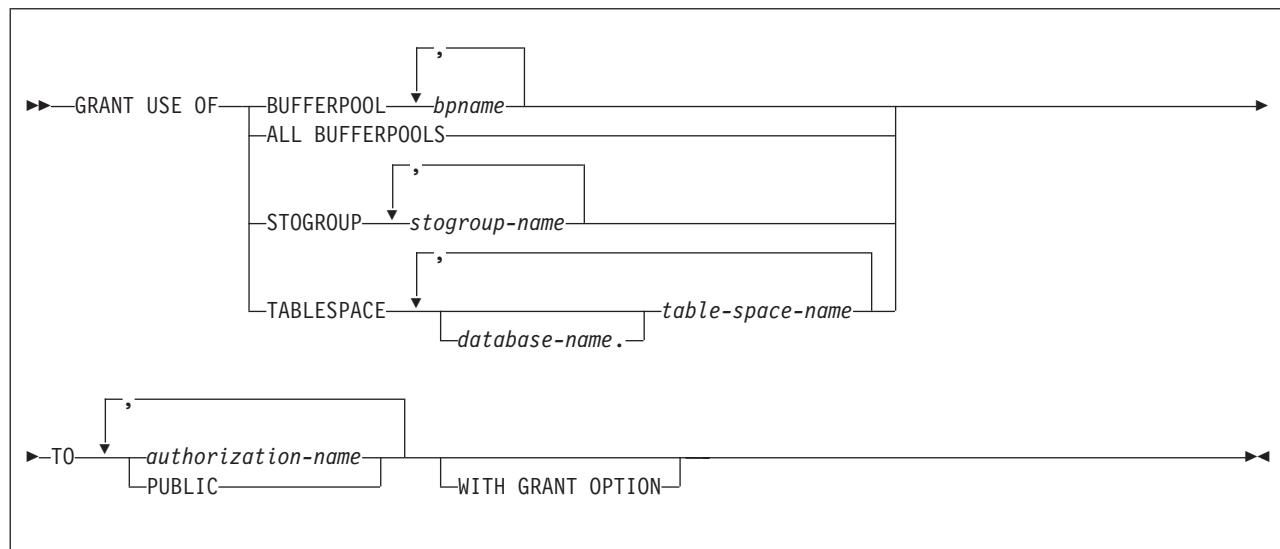
```
GRANT SELECT, UPDATE ON TABLE DSN8710.DEPT
TO PUBLIC AT ALL LOCATIONS;
```

Even with this grant, it is possible that some network users do not have access to the table at all, or to any other object at the table's subsystem. Controlling access to the subsystem involves the communications databases at the subsystems in the network. The tables for the communication databases are described in Appendix D, "DB2 catalog tables," on page 1005. Controlling access is described in Part 3 (Volume 1) of *DB2 Administration Guide*.

## GRANT (use privileges)

This form of the GRANT statement grants authority to use particular buffer pools, storage groups, or table spaces.

### Syntax



### Description

#### **BUFFERPOOL *bpname*,...**

Grants the privilege to refer to any of the identified buffer pools in a CREATE INDEX, CREATE TABLESPACE, ALTER INDEX, or ALTER TABLESPACE statement. See “Naming conventions” on page 34 for more details about *bpname*.

#### **ALL BUFFERPOOLS**

Grants the privilege to refer to any buffer pool in a CREATE INDEX, CREATE TABLESPACE, ALTER INDEX, or ALTER TABLESPACE statement.

#### **STOGROUP *stogroup-name*,...**

Grants the privilege to refer to any of the identified storage groups in a CREATE INDEX, CREATE TABLESPACE, ALTER INDEX, or ALTER TABLESPACE statement.

#### **TABLESPACE *database-name.table-space-name*,...**

Grants the privilege to refer to any of the identified table spaces in a CREATE TABLE statement. The default for *database-name* is DSNDB04.

You cannot grant the privilege for table spaces that are for declared temporary tables (table spaces in the TEMP database). For these table spaces, PUBLIC implicitly has the TABLESPACE privilege (without GRANT authority); this privilege is not recorded in the DB2 catalog, and it cannot be revoked.

#### **TO**

Refer to “GRANT” on page 803 for a description of the TO clause.

#### **WITH GRANT OPTION**

Refer to “GRANT” on page 803 for a description of the WITH GRANT OPTION clause.

## **GRANT (use privileges)**

### **Notes**

You can grant privileges for only one type of object with each statement. Thus, you can grant the use of several table spaces with one statement, but not the use of a table space and a storage group. For each object you identify, you must have the USE privilege with GRANT authority.

### **Examples**

*Example 1:* Grant authority to use buffer pools BP1 and BP2 to user MARINO.

```
GRANT USE OF BUFFERPOOL BP1,BP2
TO MARINO;
```

*Example 2:* Grant to all local users the authority to use table space DSN8S71D in database DSN8D71A.

```
GRANT USE OF TABLESPACE
DSN8D71A.DSN8S71D
TO PUBLIC;
```

## HOLD LOCATOR

The HOLD LOCATOR statement allows a LOB locator variable to retain its association with a value beyond a unit of work.

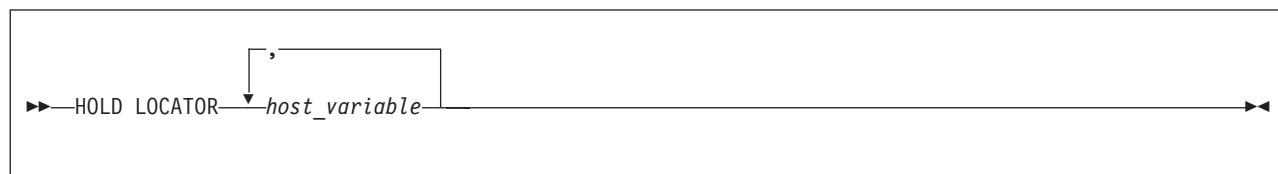
### Invocation

This statement can only be embedded in an application program. It cannot be issued interactively. It is an executable statement that can be dynamically prepared. However, the EXECUTE statement with the USING clause must be used to execute the prepared statement. HOLD LOCATOR cannot be used with the EXECUTE IMMEDIATE statement.

### Authorization

None required.

### Syntax



### Description

*host\_variable*,...

Identifies a *host-variable* locator variable that must have been previously declared according to the rules for declaring host variables. The locator variable type must be a binary large object locator, a character large object locator, or a double-byte character large object locator.

After the HOLD LOCATOR statement is executed, each locator variable in the host-variable list has the *hold* property.

If a locator variable is not an established locator within the current unit of work, an invalid locator error occurs. When this error occurs and more than one host variable was specified in the HOLD LOCATOR statement, only the locators up to the first invalid locator are held. Locators listed after the first invalid locator are not held.

### Notes

A host-variable LOB locator variable that has the hold property is freed (has its association between it and its value removed) when:

- The SQL FREE LOCATOR statement is executed for the locator variable.
- The SQL ROLLBACK statement is executed.
- The SQL session is terminated.

### Example

Assume that the employee table contains columns RESUME, HISTORY, and PICTURE and that locators have been established in a program to represent the values represented by the columns. Give the CLOB locator variables LOCRES and LOCHIST, and the BLOB locator variable LOCPIC the hold property.

```
EXEC SQL HOLD LOCATOR :LOCRES, :LOCHIST, :LOCPIC
```

## INCLUDE

## INCLUDE

The INCLUDE statement inserts declarations or code into a source program.

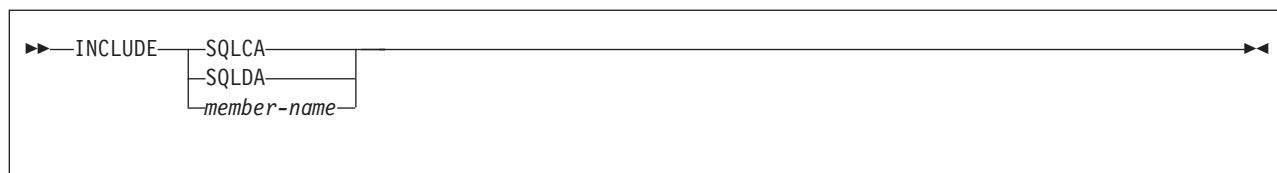
### Invocation

This statement can only be embedded in an application program. It is not an executable statement.

### Authorization

None required.

### Syntax



### Description

#### SQLCA

Indicates that the description of an SQL communication area (SQLCA) is to be included. INCLUDE SQLCA must not be specified more than once in the same application program. In COBOL, INCLUDE SQLCA must be specified in the Working-Storage Section or the Linkage Section. INCLUDE SQLCA must not be specified if the program is precompiled with the STDSQL(YES) option. Do not use the INCLUDE statement with REXX.

For a description of the SQLCA, see “SQL communication area (SQLCA)” on page 979.

#### SQLDA

Indicates that the description of an SQL descriptor area (SQLDA) is to be included. It must not be specified in a Fortran. For a description of the SQLDA, see “SQL descriptor area (SQLDA)” on page 986.

#### *member-name*

Names a member of the partitioned data set to be the library input when your application program is precompiled. It must be a short, ordinary identifier.

The member can contain any host language source statements and any SQL statements other than an INCLUDE statement. In COBOL, INCLUDE *member-name* must not be specified in other than the Data Division or the Procedure Division.

### Notes

When your application program is precompiled, the INCLUDE statement is replaced by source statements. Thus, the INCLUDE statement must be specified at a point in your application program where the resulting source statements are acceptable to the compiler.

The INCLUDE statement cannot refer to source statements that themselves contain INCLUDE statements.

The declarations that are generated by DCLGEN can be used in an application program by specifying the same member in the INCLUDE statement as in the DCLGEN LIBRARY parameter.

## **Example**

Include an SQL communications area in a PL/I program.

```
EXEC SQL INCLUDE SQLCA;
```

---

## INSERT

The INSERT statement inserts rows into a table or view. The table or view can be at the current server or any DB2 subsystem with which the current server can establish a connection. Inserting a row into a view also inserts the row into the table on which the view is based.

There are two forms of this statement:

- The INSERT via VALUES is used to insert a single row into the table or view using the values provided or referenced.
- The INSERT via SELECT is used to insert one or more rows into the table or view using values from other tables, or views, or both.

### Invocation

This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared.

### Authorization

Authority requirements depend on whether the object identified in the statement is a user-defined table, a catalog table for which inserts are allowed, or a view:

***When a user-defined table is identified:*** The privilege set must include at least one of the following:

- The INSERT privilege on the table
- Ownership of the table
- DBADM authority on the database that contains the table
- SYSADM authority

***When a catalog table is identified:*** The privilege set must include at least one of the following:

- DBADM authority on the catalog database
- SYSCTRL authority
- SYSADM authority

***When a view is identified:*** The privilege set must include at least one of the following:

- The INSERT privilege on the view
- SYSADM authority

The owner of a view, unlike the owner of a table, might not have INSERT authority on the view (or can have INSERT authority without being able to grant it to others). The nature of the view itself can preclude its use for INSERT. For more information, see the discussion of authority in “CREATE VIEW” on page 711.

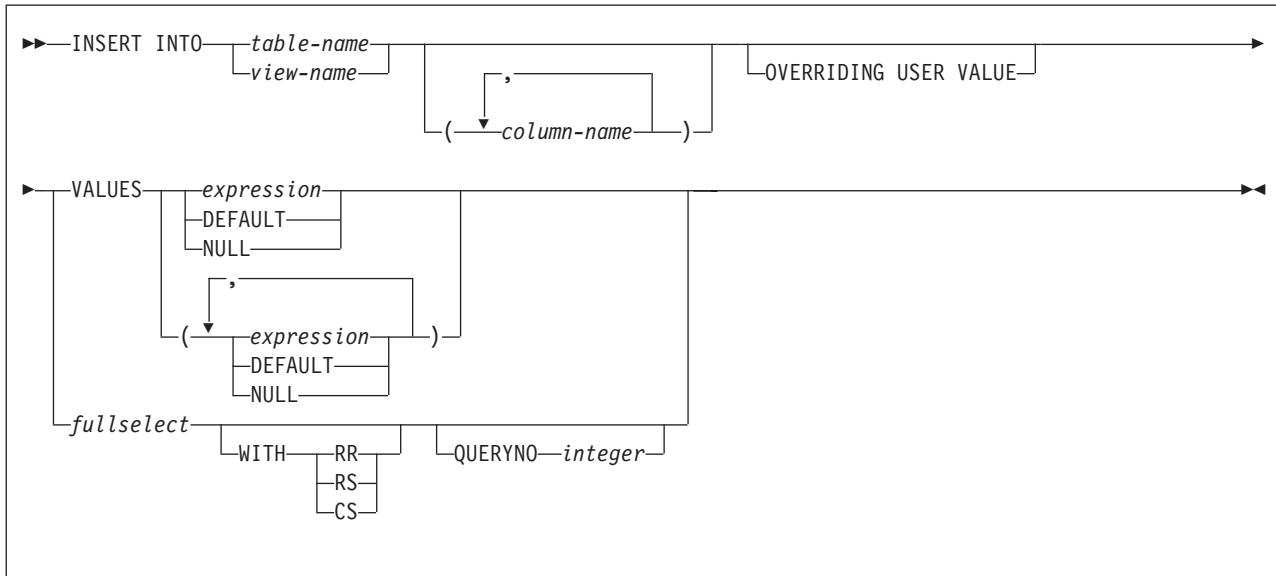
If an expression that refers to a function is specified, the privilege set must include any authority that is necessary to execute the function.

If a fullselect is specified, the privilege set must include authority to execute the fullselect. For more information about the fullselect authorization rules, see “Authorization” on page 348.

If the statement is embedded in an application program, the privilege set is the privileges that are held by the authorization ID of the owner of the plan or package. If the statement is dynamically prepared, the privilege set is determined by the DYNAMICRULES behavior in effect (run, bind, define, or invoke) and is

summarized in Table 40 on page 382. (For more information on these behaviors, including a list of the DYNAMICRULES bind option values that determine them, see “Authorization IDs and dynamic SQL” on page 43.)

## Syntax



## Description

### **INTO** *table-name* or *view-name*

Identifies the object of the INSERT statement. The name must identify a table or view that exists at the DB2 subsystem identified by the implicitly or explicitly specified location name. The name must not identify:

- An auxiliary table
- A catalog table for which inserts are not allowed
- A view of such a catalog table
- A read-only view. (For a description of a read-only view, see “CREATE VIEW” on page 711.)

A value cannot be inserted into a view column that is derived from:

- A constant, expression, or scalar function
- The same base table column as some other column of the view

If the object of the INSERT statement is a view with such columns, a list of column names must be specified, and the list must not identify these columns. In an IMS or CICS application, the DB2 subsystem that contains the identified table or view must not be a remote DB2 Version 2 Release 3 subsystem.

### *column-name*,...

Specifies the columns for which insert values are provided. Each name must be an unqualified name that identifies a column of the table or view. The columns can be identified in any order, but the same column must not be identified more than once. A view column that cannot accept insert values must not be identified.

Omission of the column list is an implicit specification of a list in which every column of the table or view is identified in left-to-right order. This list is

## INSERT

established when the statement is bound and therefore does not include columns that were added to the table after the statement was bound.

The effect of a rebind on INSERT statements that do not include a column list is that the implicit list of names is re-established. Therefore, the number of columns into which data is inserted can change and cause an error.

### OVERRIDING USER VALUE

Specifies that the value specified in the VALUES clause or produced by a fullselect for a column that is defined as GENERATED ALWAYS is ignored. Instead, a system-generated value is inserted, overriding the user-specified value.

Specify OVERRIDING USER VALUE only if the insert involves a column defined as GENERATED ALWAYS, such as a ROWID column or an identity column.

### VALUES

Specifies one new row in the form of a list of values. The number of values in the VALUES clause must equal the number of names in the column list. The first value is inserted in the first column in the list, the second value in the second column, and so on. If more than one value is specified, the list of values must be enclosed in parentheses.

#### *expression*

Any expression of the type described in “Expressions” on page 111. The expression must not include a column name. If *expression* is a single host variable, the host variable can identify a structure. Any host variable or structure that is specified must be described in the application program according to the rules for declaring host structures and variables.

### DEFAULT

The default value assigned to the column. If the column is a ROWID column or an identity column, DB2 will generate a unique value for the column. You can specify DEFAULT only for columns that have an assigned default value, ROWID columns, and identity columns.

For information on default values of data types, see the description of the DEFAULT clause for “CREATE TABLE” on page 653.

### NULL

The null value.

For a ROWID or an identity column that was defined as GENERATED ALWAYS, you must specify DEFAULT unless you specify the OVERRIDING USER VALUE clause to indicate that any user-specified value will be ignored and a unique system-generated value will be inserted.

For a ROWID or identity column that is defined as GENERATED BY DEFAULT, you can specify a value. However, a value can be inserted into ROWID column defined BY DEFAULT only if a single-column unique index is defined on the ROWID column and the specified value is a valid row ID value that was previously generated by DB2. When a value is inserted into an identity column defined BY DEFAULT, DB2 does not verify that the specified value is a unique value for the column unless the identity column has a single-column unique index. Without a unique index, DB2 can guarantee unique values only among the set of system-generated values as long as NO CYCLE is in effect.

|

**fullselect**

Specifies a set of new rows in the form of the result table of a fullselect. If the result table is empty, SQLCODE is set to +100, and SQLSTATE is set to '02000'.

(For an explanation of fullselect, see "fullselect" on page 365.)

The number of columns in the result table must equal the number of names in the column list. The value of the first column of the result is inserted in the first column in the list, the second value in the second column, and so on. Any values that are produced for a ROWID or identity column must conform to the rules that are described for those columns under the VALUES clause.

**WITH**

Specifies the isolation level at which the fullselect is executed.

- RR** Repeatable read
- RS** Read stability
- CS** Cursor stability

The **default** isolation level of the statement is the isolation level of the package or plan in which the statement is bound, with the package isolation taking precedence over the plan isolation. When a package isolation is not specified, the plan isolation is the default.

**QUERYNO integer**

Specifies the number to be used for this SQL statement in EXPLAIN output and trace records. The number is used for the QUERYNO column of the plan table for the rows that contain information about this SQL statement. This number is also used in the QUERYNO column of the SYSIBM.SYSSTMT and SYSIBM.SYSPACKSTMT catalog tables.

If the clause is omitted, the number associated with the SQL statement is the statement number assigned during precompilation. Thus, if the application program is changed and then precompiled, that statement number might change.

Using the QUERYNO clause to assign unique numbers to the SQL statements in a program is helpful:

- For simplifying the use of optimization hints for access path selection
- For correlating SQL statement text with EXPLAIN output in the plan table

For information on using optimization hints, such as enabling the system for optimization hints and setting valid hint values, and for information on accessing the plan table, see Part 5 (Volume 2) of *DB2 Administration Guide*.

## Notes

**Insert rules:** Insert values must satisfy the following rules. If they do not, or if any other errors occur during the execution of the INSERT statement, no rows are inserted and the position of the cursors are not changed.

- *Default values.* The value inserted in any column that is not in the column list is the default value of the column. Columns without a default value must be included in the column list. Similarly, if you insert into a view, the default value is inserted into any column of the base table that is not included in the view. Hence, all columns of the base table that are not in the view must have a default value.
- *Length.* If the insert value of a column is a number, the column must be a numeric column with the capacity to represent the integral part of the number. If the insert value of a column is a string, the column must be either a string

## INSERT

column with a length attribute at least as great as the length of the string, or a datetime column if the string represents a date, time, or timestamp.

- *Assignment.* Insert values are assigned to columns in accordance with the assignment rules described in Chapter 2, “Language elements,” on page 27.
- *Uniqueness constraints.* If the identified table or the base table of the identified view has one or more unique indexes, each row inserted into the table must conform to the constraints imposed by those indexes.
- *Referential constraints.* Each nonnull insert value of a foreign key must be equal to some value of the parent key of the parent table in the relationship.
- *Check constraints.* The identified table or the base table of the identified view might have one or more check constraints. Each row inserted must conform to the conditions imposed by those constraints. Thus, each check condition must be true or unknown.
- *Field and validation procedures.* If the identified table or the base table of the identified view has a field or validation procedure, each row inserted must conform to the constraints imposed by that procedure.
- *Views and the WITH CHECK OPTION.* For views defined with WITH CHECK OPTION, each row you insert into the view must conform to the definition of the view. If the view you name is dependent on other views whose definitions include WITH CHECK OPTION, the inserted rows must also conform to the definitions of those views. For an explanation of the rules governing this situation, see “CREATE VIEW” on page 711.

For views that are not defined with WITH CHECK OPTION, you can insert rows that do not conform to the definition of the view. Those rows cannot appear in the view but are inserted into the base table of the view.

- *Omitting the column list.* When you omit the column list, you must specify a value for every column that was present in the table when the INSERT statement was bound or (for dynamic execution) prepared.
- *Triggers.* An INSERT statement might cause triggers to be activated. A trigger might cause other statements to be executed or raise error conditions based on the insert values.

**Number of rows inserted:** Normally, after an INSERT statement completes execution, the value of SQLERRD(3) in the SQLCA is the number of rows inserted. (For a complete description of the SQLCA, including exceptions to the above statement, see “SQL communication area (SQLCA)” on page 979.)

**Nesting user-defined functions or stored procedures:** An INSERT statement can implicitly or explicitly refer to user-defined functions or stored procedures. This is known as *nesting* of SQL statements. A user-defined function or stored procedure that is nested within the INSERT must not access the table into which you are inserting values.

**Locking:** Unless appropriate locks already exist, one or more exclusive locks are acquired at the execution of a successful INSERT statement. Until a commit or rollback operation releases the locks, only the application process that performed the insert can access the inserted row. If LOBs are not inserted into the row, application processes that are running with uncommitted read can also access the inserted row. The locks can also prevent other application processes from performing operations on the table. However, application processes that are running with uncommitted read can access locked pages and rows.

Locks are not acquired on declared temporary tables.

**Inserting rows into catalog table SYSIBM.SYSSTRINGS:** If the object table is SYSIBM.SYSSTRINGS, only certain values can be specified, as described in Appendix B (Volume 2) of *DB2 Administration Guide*.

**Datetime representation when using datetime registers:** As explained under “Datetime special registers” on page 84, when two or more datetime registers are implicitly or explicitly specified in a single SQL statement, they represent the same point in time. This is also true when multiple rows are inserted.

## Examples

*Example 1:* Insert values into sample table DSN8710.EMP.

```
INSERT INTO DSN8710.EMP
VALUES ('000205','MARY','T','SMITH','D11','2866',
 '1981-08-10','ANALYST',16,'F','1956-05-22',
 16345,500,2300);
```

*Example 2:* Assume that SMITH.TEMPEMPL is a created temporary table. Populate the table with data from sample table DSN8710.EMP.

```
INSERT INTO SMITH.TEMPEMPL
SELECT *
FROM DSN8710.EMP;
```

*Example 3:* Assume that SESSION.TEMPEMPL is a declared temporary table. Populate the table with data from department D11 in sample table DSN8710.EMP.

```
INSERT INTO SESSION.TEMPEMPL
SELECT *
FROM DSN8710.EMP
WHERE WORKDEPT='D11';
```

*Example 4:* Insert a row into sample table DSN8710.EMP\_PHOTO\_RESUME. Set the value for column EMPNO to the value in host variable HV\_ENUM. Let the value for column EMP\_ROWID be generated because it was defined with a row ID data type and with clause GENERATED ALWAYS.

```
INSERT INTO DSN8710.EMP_PHOTO_RESUME(EMPNO, EMP_ROWID)
VALUES (:HV_ENUM, DEFAULT);
```

You can only insert user-specified values into ROWID columns that are defined as GENERATED BY DEFAULT and not as GENERATED ALWAYS.. Therefore, in the above example, if you were to try to insert a value into EMP\_ROWID instead of specifying DEFAULT, the statement would fail unless you also specify OVERRIDING USER VALUE. For columns that are defined as GENERATED ALWAYS, the OVERRIDING USER VALUE clause causes DB2 to ignore any user-specified value and generate a value instead.

For example, assume that you want to copy the rows in DSN8710.EMP\_PHOTO\_RESUME to another table that has a similar definition (both tables have a ROWID columns defined as GENERATED ALWAYS). For the following INSERT statement, the OVERRIDING USER VALUE clause causes DB2 to ignore the EMP\_ROWID column values from DSN8710.EMP\_PHOTO\_RESUME and generate values for the corresponding ROWID column in B.EMP\_PHOTO\_RESUME.

```
INSERT INTO B.EMP_PHOTO_RESUME
OVERRIDING USER VALUE
SELECT * FROM DSN8710.EMP_PHOTO_RESUME;
```

## LABEL ON

### LABEL ON

The LABEL ON statement adds or replaces labels in the descriptions of tables, views, aliases, or columns in the catalog at the current server.

### Invocation

This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared.

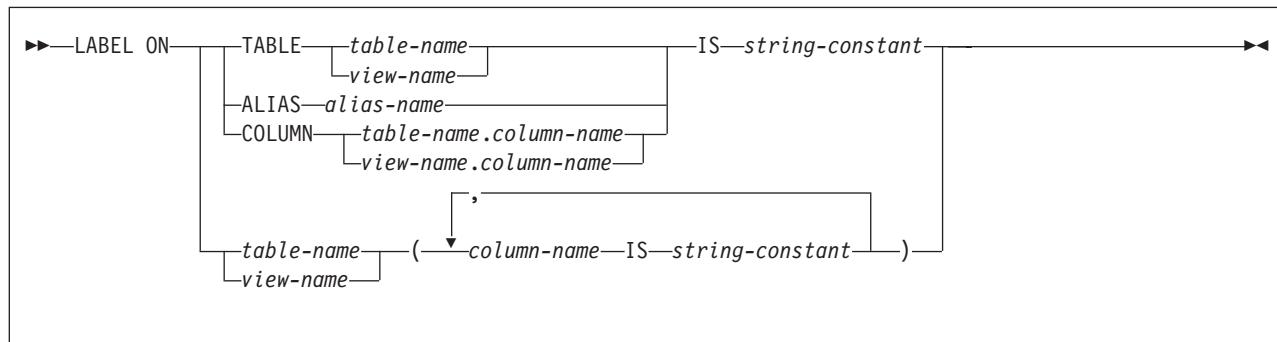
### Authorization

The privilege set that is defined below must include at least one of the following:

- Ownership of the table, view, or alias
- DBADM authority for its database (tables only)
- SYSADM or SYSCtrl authority

**Privilege set:** If the statement is embedded in an application program, the privilege set is the privileges that are held by the authorization ID of the owner of the plan or package. If the statement is dynamically prepared, the privilege set is determined by the DYNAMICRULES behavior in effect (run, bind, define, or invoke) and is summarized in Table 40 on page 382. (For more details on these behaviors, including a list of the DYNAMICRULES bind option values that determine them, see “Authorization IDs and dynamic SQL” on page 43.)

### Syntax



### Description

#### TABLE

Indicates that the label is for a table or a view.

#### table-name or view-name

Identifies the table or view to which the label applies. The name must identify a table or view that exists at the current server. *table-name* must not identify a declared temporary table. The label is placed into the LABEL column of the SYSIBM.SYSTABLES catalog table for the row that describes the table or view.

#### ALIAS

Identifies the alias to which the comment applies.

#### alias-name

The name must identify an alias that exists at the current server. The label is placed in the LABEL column of the SYSIBM.SYSTABLES catalog table for the row that describes the alias.

**COLUMN**

Indicates that the label is for a column.

*table-name.column-name or view-name.column-name*

Identifies the column to which the label applies. The name must identify a column of a table or view that exists at the current server. The name must not identify a column of a declared temporary table. The label is placed in the LABEL column of the SYSIBM.SYSCOLUMNS catalog table in the row that describes the column.

**Do not use TABLE or COLUMN to define a label for more than one column in a table or view.** Give the table or view name and then, in parentheses, a list in the form:

*column-name IS string-constant,  
column-name IS string-constant,...*

See Example 2 below.

The column names must not be qualified, each name must identify a column of the specified table or view, and that table or view must exist at the current server.

**IS** Introduces the label you want to provide.

*string-constant*

Can be any SQL character string constant of up to 30 bytes in length.

## Examples

*Example 1:* Enter a label on the DEPTNO column of table DSN8710.DEPT.

```
LABEL ON COLUMN DSN8710.DEPT.DEPTNO
IS 'DEPARTMENT NUMBER';
```

*Example 2:* Enter labels on two columns in table DSN8710.DEPT.

```
LABEL ON DSN8710.DEPT
(MGRNO IS 'MANAGER'S EMPLOYEE NUMBER',
ADMRDEPT IS 'ADMINISTERING DEPARTMENT');
```

## LOCK TABLE

### LOCK TABLE

The LOCK TABLE statement requests a lock on a table or table space at the current server. The lock is not acquired if the process already holds an appropriate lock.

#### Invocation

This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared.

#### Authorization

The privilege set that is defined below must include at least one of the following:

- The SELECT privilege on the identified table
- Ownership of the table
- DBADM authority for the database
- SYSADM or SYSCTRL authority

**Privilege set:** If the statement is embedded in an application program, the privilege set is the privileges that are held by the authorization ID of the owner of the plan or package. If the statement is dynamically prepared, the privilege set is determined by the DYNAMICRULES behavior in effect (run, bind, define, or invoke) and is summarized in Table 40 on page 382. (For more details on these behaviors, including a list of the DYNAMICRULES bind option values that determine them, see “Authorization IDs and dynamic SQL” on page 43.)

#### Syntax

```
►►LOCK TABLE—table-name [PART—integer] IN {SHARE | EXCLUSIVE} MODE►►
```

#### Description

##### *table-name*

Identifies the table to be locked. The name must identify a table that exists at the current server. It must not identify a view, a temporary table (created or declared), or a catalog table. The lock might or might not apply exclusively to the table. The effect of locking an auxiliary table is to lock the LOB table space that contains the auxiliary table.

##### PART *integer*

Identifies the partition of a partitioned table space to lock. The table identified by *table-name* must belong to a partitioned table space that is defined with LOCKPART YES. The value specified for *integer* must be an integer that is no greater than the number of partitions in the table space.

##### IN SHARE MODE

For a lock on a table that is not an auxiliary table, requests the acquisition of a lock that prevents other processes from executing anything but read-only operations on the table. For a lock on a LOB table space, IN SHARE mode requests a lock that prevents storage from being reallocated. When a LOB table space is locked, other processes can delete LOBs or update them to a null

value, but they cannot insert LOBs with a non-null value. The type of lock that the process holds after execution of the statement depends on what lock, if any, the process already holds.

#### IN EXCLUSIVE MODE

Requests the acquisition of an exclusive lock for the application process. Until the lock is released, it prevents concurrent processes from executing any operations on the table. However, unless the lock is on a LOB table space, concurrent processes that are running at an isolation level of uncommitted read (UR) can execute read-only operations on the table.

### Notes

**Releasing locks:** If LOCK TABLE is a static SQL statement, the RELEASE option of bind determines when DB2 releases a lock. For RELEASE(COMMIT), DB2 releases the lock at the next commit point. For RELEASE(DEALLOCATE), DB2 releases the lock when the plan is deallocated (the application ends).

If LOCK TABLE is a dynamic SQL statement, DB2 uses RELEASE(COMMIT) and releases the lock at the next commit point, *unless* the table or table space is referenced by cached dynamic statements. Caching allows DB2 to keep prepared statements in memory past commit points. In this case, DB2 holds the lock until deallocation or until the commit after the prepared statements are freed from memory. Under some conditions, if a lock is held past a commit point, DB2 demotes the lock state of a segmented table or a nonsegmented table space to an intent lock at the commit point.

For more information on using LOCK TABLE (such as the size and duration of locks), and on locking in general, see Part 4 of *DB2 Application Programming and SQL Guide* or Part 5 (Volume 2) of *DB2 Administration Guide*.

### Example

Obtain a lock on the sample table named DSN8710.EMP, which resides in a partitioned table space. The lock obtained applies to every partition and prevents other application programs from either reading or updating the table.

```
LOCK TABLE DSN8710.EMP IN EXCLUSIVE MODE;
```

## OPEN

## OPEN

The OPEN statement opens a cursor.

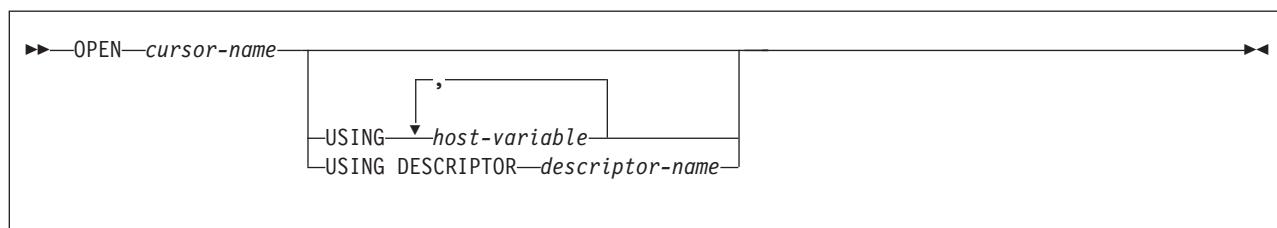
### Invocation

This statement can only be embedded in an application program. It is an executable statement that cannot be dynamically prepared.

### Authorization

See “DECLARE CURSOR” on page 718 for the authorization required to use a cursor.

### Syntax



### Description

#### *cursor-name*

Identifies the cursor to be opened. The *cursor-name* must identify a declared cursor as explained in “Notes” on page 721 in the description of the DECLARE CURSOR statement. When the OPEN statement is executed, the cursor must be in the closed state.

The SELECT statement of the cursor is either:

- The *select-statement* specified in the DECLARE CURSOR statement, or
- The prepared *select-statement* identified by the *statement-name* specified in the DECLARE CURSOR statement. If the statement has not been successfully prepared, or is not a *select-statement*, the cursor cannot be successfully opened.

The result table of the cursor is derived by evaluating the SELECT statement. The evaluation uses the current values of any special registers specified in the SELECT statement and the current values of any host variables specified in the SELECT statement or the USING clause of the OPEN statement. The rows of the result table can be derived during the execution of the OPEN statement and a temporary copy of a result table can be created to hold them. They can be derived during the execution of later FETCH statements. In either case, the cursor is placed in the open state and positioned before the first row of its result table. If the table is empty the position of the cursor is effectively “after the last row.” DB2 does not indicate an empty table when the OPEN statement is executed. But it does indicate that condition, on the first execution of FETCH, by returning values of +100 for SQLCODE and '02000' for SQLSTATE.

#### USING

Introduces a list of host variables whose values are substituted for the parameter markers (question marks) of a prepared statement. (For an explanation of parameter markers, see “PREPARE” on page 846.) If the DECLARE CURSOR statement names a prepared statement that includes

parameter markers, you must use USING. If the prepared statement does not include parameter markers, USING is ignored.

*host-variable*...

Identifies host structures or variables that must be described in the application program in accordance with the rules for declaring host structures and variables. When the statement is executed, a reference to a structure is replaced by a reference to each of its variables. The number of variables must be the same as the number of parameter markers in the prepared statement. The *n*th variable corresponds to the *n*th parameter marker in the prepared statement. Where appropriate, locator variables can be provided as the source of values for parameter markers.

**USING DESCRIPTOR** *descriptor-name*

Identifies an SQLDA that contains a valid description of the input host variables.

Before the OPEN statement is processed, the user must set the following fields in the SQLDA:

- SQLN to indicate the number of SQLVAR occurrences provided in the SQLDA  
A REXX SQLDA does not contain this field.
- SQLD to indicate the number of variables used in the SQLDA when processing the statement
- SQLVAR occurrences to indicate the attributes of the variables

The SQLDA must have enough storage to contain all SQLVAR occurrences. If LOBs or distinct types are present in the results, there must be additional SQLVAR entries for each input host variable. For more information on the SQLDA, which includes a description of the SQLVAR and an explanation on how to determine the number of SQLVAR occurrences, see “SQL descriptor area (SQLDA)” on page 986.

SQLD must be set to a value greater than or equal to zero and less than or equal to SQLN. It must be the same as the number of parameter markers in the prepared statement.

See “Identifying an SQLDA in C or C++” on page 1003 for how to represent *descriptor-name* in C.

When the SELECT statement of the cursor is evaluated, each parameter marker in the statement is effectively replaced by the value of its corresponding host variable. For more on the process of replacement, see “Parameter marker replacement” on page 844.

The USING clause is intended for a prepared SELECT statement that contains parameter markers. However, it can also be used when the SELECT statement of the cursor is part of the DECLARE CURSOR statement and contains a host variable. In this case, the OPEN statement is executed as if each host variable in the SELECT statement were a parameter marker except that the attributes of the target variable are the same as the attributes of the host variables in the SELECT statement. The effect is to override the values of the host variables in the SELECT statement of the cursor with the values of the host variables specified in the USING clause. If a predicate of the SELECT refers to a host variable that does not have an indicator variable, the overriding value is always the value of the main variable because the indicator variable is ignored without a warning.

## Notes

**Errors occurring on OPEN:** In local and remote processing, the DEFER(PREPARE) and REOPT(VARS) bind options can cause some SQL statements to receive “delayed” errors. For example, an OPEN statement might receive an SQLCODE that normally occurs during PREPARE processing. Or a FETCH statement might receive an SQLCODE that normally occurs at OPEN time.

**Closed state of cursors:** All cursors in an application process are in the closed state when:

- The application process is started.
- A new unit of work is started for the application process unless the WITH HOLD option has been used in the DECLARE CURSOR statement.
- A CONNECT has been performed. (CONNECT implicitly closes any open cursors.)

A cursor can also be in the closed state because:

- A CLOSE statement was executed.
- An error was detected that made the position of the cursor unpredictable.

To retrieve rows from the result table of a cursor, you must execute a FETCH statement when the cursor is open. The only way to change the state of a cursor from closed to open is to execute an OPEN statement.

**Effect of a temporary copy of a result table:** DB2 can process a cursor in two different ways:

- It can create a temporary copy of the result table during the execution of the OPEN statement.
- It can derive the result table rows as they are needed during the execution of later FETCH statements.

If the result table is not read-only, DB2 uses the latter method. If the result table is read-only, either method could be used. The results produced by these two methods could differ in the following respects:

*When a temporary copy of the result table is used:* An error can occur during OPEN that would otherwise not occur until some later FETCH statement. Moreover, INSERT, UPDATE, and DELETE statements executed while the cursor is open cannot affect the result table.

*When a temporary copy of the result table is not used:* INSERT, UPDATE, and DELETE statements executed while the cursor is open can affect the result table if they are issued from the same application process. The effect of such operations is not always predictable. For example, if cursor C is positioned on a row of its result table defined as SELECT \* FROM T, and you insert a row into T, the effect of that insert on the result table is not predictable because its rows are not ordered. A later FETCH C might or might not retrieve the new row of T.

**Parameter marker replacement:** Before the OPEN statement is executed, each parameter marker in the query is effectively replaced by its corresponding host variable. The replacement is an assignment operation in which the source is the value of the host variable and the target is a variable within DB2. The assignment rules are those described for assignment to a column in “Assignment and comparison” on page 64. For a typed parameter marker, the attributes of the target variable are those specified by the CAST specification. For an untyped parameter

marker, the attributes of the target variable are determined according to the context of the parameter marker. For the rules that affect parameter markers, see “Parameter markers” on page 852.

Let V denote a host variable that corresponds to parameter marker P. The value of V is assigned to the target variable for P in accordance with the rules for assigning a value to a column:

- V must be compatible with the target.
- If V is a string, its length (excluding trailing blanks) must not be greater than the length attribute of the target.
- If V is a number, the absolute value of its integral part must not be greater than the maximum absolute value of the integral part of the target.
- If the attributes of V are not identical to the attributes of the target, the value is converted to conform to the attributes of the target.

When the prepared statement is executed, the value used in place of P is the value of the target variable for P. For example, if V is CHAR(6) and the target is CHAR(8), the value used in place of P is the value of V padded on the right with two blanks.

**Considerations for scrollables cursors:** For a scrollable cursor, the OPEN cursor statement returns the following information in the SQLCA:

- Information on the scrollability of the cursor is returned in SQLWARN1 with the following values:
  - N = non-scrollable
  - S = scrollable
- Information on the effective sensitivity of the cursor is returned in SQLWARN4 with the following values:
  - I = insensitive
  - S = sensitive static

The information can be used by applications (such as an ODBC driver) to determine what type of FETCH (INSENSITIVE or SENSITIVE) to issue for a cursor defined as ASENSITIVE.

- Information on the effective capability of the cursor as updatable, deleteable, or read-only is returned in SQLWARN5 with the following values:
  - 1 = Read-Only. (The result table of the query is read-only either because the content of the SELECT statement was implicitly read-only or FOR READ/FETCH ONLY was explicitly specified.)
  - 2 = Read and DELETE allowed. (The result table of the query is deleteable, but not updatable.)
  - 4 = Read, DELETE, and UPDATE allowed. (The result table of the query is deleteable and updatable.)

## Example

The OPEN statement in the following example places the cursor at the beginning of the rows to be fetched.

```

EXEC SQL DECLARE C1 CURSOR FOR
 SELECT DEPTNO, DEPTNAME, MGRNO FROM DSN8710.DEPT
 WHERE ADMRDEPT = 'A00';
EXEC SQL OPEN C1;
DO WHILE (SQLCODE = 0);
 EXEC SQL FETCH C1 INTO :DNUM, :DNAME, :MNUM;
END;
EXEC SQL CLOSE C1;
```

## PREPARE

## PREPARE

The PREPARE statement creates an executable SQL statement from a string form of the statement. The executable form is called a prepared statement. The string form is called a statement string.

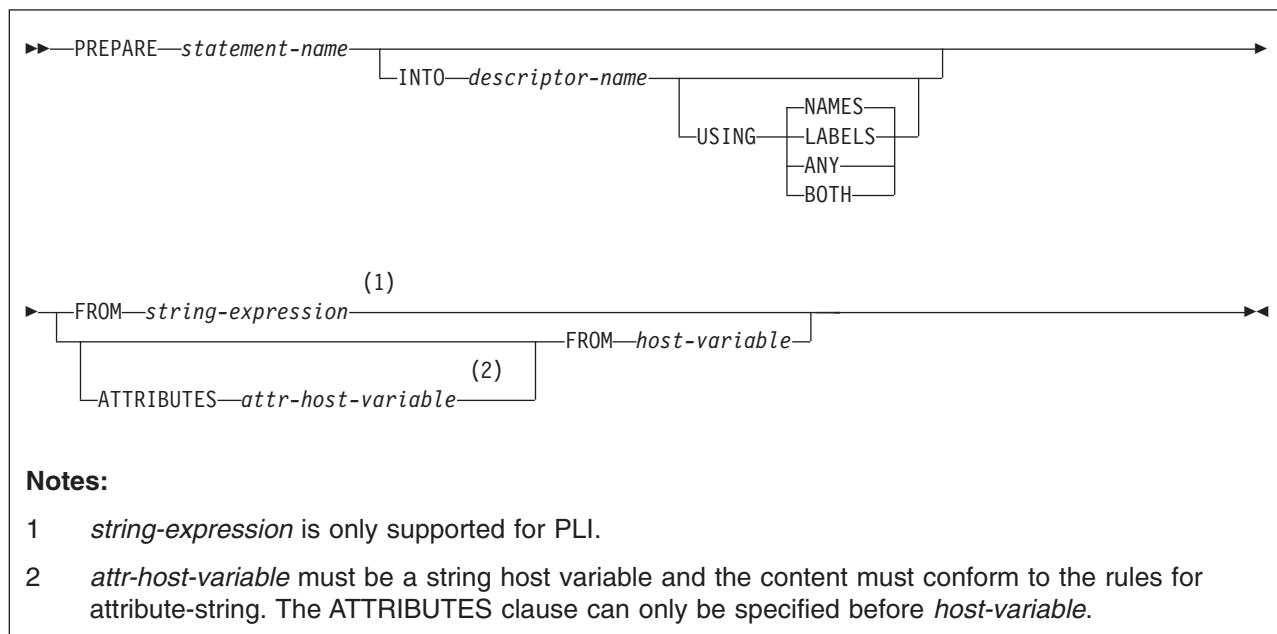
### Invocation

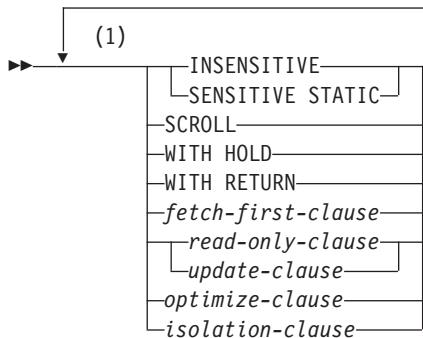
This statement can only be embedded in an application program. It is an executable statement that cannot be dynamically prepared.

### Authorization

The authorization rules are those defined for the dynamic preparation of the SQL statement specified by the PREPARE statement. For example, see Chapter 4, “Queries,” on page 347 for the authorization rules that apply when a SELECT statement is prepared.

### Syntax



**attribute-string****Notes:**

- 1 The same clause must not be specified more than once. If the options are not specified, their defaults are whatever was specified for the corresponding option in an associated DECLARE CURSOR statement.

## Description

*statement-name*

Names the prepared statement. If the name identifies an existing prepared statement, that prepared statement is destroyed. The name must not identify a prepared statement that is the SELECT statement of an open cursor.

**INTO**

If you use INTO, and the PREPARE statement is successfully executed, information about the prepared statement is placed in the SQLDA specified by the descriptor name. Thus, the PREPARE statement:

```
EXEC SQL PREPARE S1 INTO :SQLDA FROM :V1;
```

is equivalent to:

```
EXEC SQL PREPARE S1 FROM :V1;
EXEC SQL DESCRIBE S1 INTO :SQLDA;
```

*descriptor-name*

Identifies the SQLDA. For languages other than REXX, SQLN must be set to indicate the number of SQLVAR occurrences. See “DESCRIBE (prepared statement or table)” on page 749 and “SQL descriptor area (SQLDA)” on page 986 for information about how to determine the number of SQLVAR occurrences to use and for an explanation of the information that is placed in the SQLDA.

See “Identifying an SQLDA in C or C++” on page 1003 for how to represent *descriptor-name* in C.

**USING**

Indicates what value to assign to each SQLNAME variable in the SQLDA when INTO is used. If the requested value does not exist, SQLNAME is set to length 0.

## PREPARE

### NAMES

Assigns the name of the column. This is the default.

### LABELS

Assigns the label of the column. (Column labels are defined by the LABEL ON statement.)

### ANY

Assigns the column label, and, if the column has no label, the column name.

### BOTH

Assigns both the label and name of the column. In this case, two or three occurrences of SQLVAR per column, depending on whether the result set contains distinct types, are needed to accommodate the additional information. To specify this expansion of the SQLVAR array, set SQLN to  $2 \times n$  or  $3 \times n$ , where  $n$  is the number of columns in the object being described. For each of the columns, the first  $n$  occurrences of SQLVAR, which are the base SQLVAR entries, contain the column names. Either the second or third  $n$  occurrences of SQLVAR, which are the extended SQLVAR entries, contain the column labels. If there are no distinct types, the labels are returned in the second set of SQLVAR entries. Otherwise, the labels are returned in the third set of SQLVAR entries.

A REXX SQLDA does not include the SQLN field, so you do not need to set SQLN for REXX programs.

### ATTRIBUTES *attr-host-variable*

Specifies the attributes for this cursor that are in effect if a corresponding attribute has not been specified as part of the associated SELECT statement. If attributes are specified in the SELECT statement, they are used instead of the corresponding attributes specified on the PREPARE statement. In turn, if attributes are specified in the PREPARE statement, they are used instead of the corresponding attributes specified on a DECLARE CURSOR statement.

*attr-host-variable* must identify a host variable that is described in the program in accordance with the rules for declaring string variables. *attr-host-variable* must be a string variable (either fixed-length or varying-length) that has a length attribute that does not exceed the maximum length of a VARCHAR. Leading and trailing blanks are removed from the value of the host variable. The host variable must contain a valid *attribute-string*.

An indicator variable can be used to indicate whether or not attributes are actually provided on the PREPARE statement. Thus, applications can use the same PREPARE statement regardless of whether attributes need to be specified or not.

The options that can be specified as part of the *attribute-string* are as follows:

#### INSENSITIVE, or SENSITIVE STATIC<sup>37</sup>

If *INSENSITIVE* is specified, changes to the database after the result table is created are not visible to the cursor. INSENSITIVE defines the sensitivity of the cursor to updates and deletes. A cursor defined as INSENSITIVE cannot be used for updates and deletes. If the SELECT

37. The scrollability and sensitivity of the cursor are independent and do not have to be specified together. Thus, the cursor might be defined as SCROLL INSENSITIVE, but the PREPARE statement might specify SENSITIVE STATIC as an override for the sensitivity.

statement contains a FOR UPDATE clause, an error is returned. An update or delete attempt using an INSENSITIVE scrollable cursor also results in an error.

If *SENSITIVE* is specified, changes made to the database after the result table is created are visible to the cursor. SENSITIVE defines the sensitivity of the cursor to updates and deletes. The extent of changes visible to the cursor depends on the sensitivity specified in the FETCH statement.

Using a non-deterministic function (built-in or user-defined) in the WHERE clause of *select-statement* or *statement-name* of a SENSITIVE STATIC cursor can cause misleading results. This occurs because DB2 constructs a temporary result table and retrieves rows from this table for INSENSITIVE FETCH statements. When DB2 processes a SENSITIVE FETCH statement, rows are fetched from the underlying table and predicates are re-evaluated if they contain non-correlated subqueries. Using a non-deterministic function can yield a different result for the re-evaluated query causing the row to no longer be considered a match.

*STATIC* specifies that the size of the result table does not change after the cursor is opened and positioned updates and positioned deletes are allowed if the result table is updatable.

Static cursors have visibility to changes made by a cursor using UPDATE WHERE CURRENT OF or DELETE WHERE CURRENT OF. Visibility of changes made outside the cursor is possible with the new SENSITIVE option of the FETCH statement. A FETCH SENSITIVE can result in *holes* in the result table. If the row no longer qualifies the predicate, it results in an *update hole*. If the row was deleted, it results in a *delete hole*. When a FETCH SENSITIVE detects a delete hole, no data is returned and the cursor is left positioned on the delete hole. When a FETCH SENSITIVE detects an update hole, no data is returned and the cursor is left positioned on the update hole.

Updates through the cursor result in automatic refresh of the row. This refresh means that the updates can create a hole themselves. This refresh also means that the refreshed row reflects changes as a result of update triggers. It is important to reflect these changes to maintain the consistency of data in the row.

If INSENSITIVE or SENSITIVE STATIC is specified as part of the ATTRIBUTES clause, SCROLL must be specified either as part of the ATTRIBUTES clause or as part of the definition of the cursor (DECLARE CURSOR).

### SCROLL<sup>37</sup>

SCROLL specifies that the cursor is scrollable.

If SCROLL is specified as part of the ATTRIBUTES clause, a cursor sensitivity option (INSENSITIVE or SENSITIVE STATIC) must be specified either as part of the ATTRIBUTES clause or as part of the definition of the cursor (DECLARE CURSOR).

### WITH HOLD

Prevents the cursor from being closed as a consequence of a commit operation. A cursor declared with WITH HOLD is closed at commit time if one of the following is true:

## PREPARE

- The connection associated with the cursor is in the release pending status.
- The bind option DISCONNECT(AUTOMATIC) is in effect.
- The environment is one in which the option WITH HOLD is ignored.

When WITH HOLD is specified, a commit operation commits all the changes in the current unit of work, but releases only locks that are not required to maintain the cursor. Afterwards, an initial FETCH statement is required before a positioned update or delete statement can be executed. After the initial FETCH, the cursor is positioned on the row following the one it was positioned on before the commit operation.

All cursors are implicitly closed by a connect (Type 1) or rollback operation. A cursor is also implicitly closed by a commit operation if WITH HOLD is ignored or not specified.

Cursors that are declared with WITH HOLD in CICS or in IMS non-message-driven programs will not be closed by a rollback operation if the cursor was opened in a previous unit of work and no changes have been made to the database in the current unit of work. The cursor cannot be closed because CICS and IMS do not broadcast the rollback request to DB2 for a null unit of work.

If a cursor is closed before the commit operation, the effect is the same as if the cursor was declared without the option WITH HOLD.

WITH HOLD is ignored in IMS message driven programs (MPP, IFP, and message-driven BMP). WITH HOLD maintains the cursor position in a CICS pseudo-conversational program until the end-of-task (EOT).

For details on restrictions that apply to declaring cursors with WITH HOLD, see Part 2 of *DB2 Application Programming and SQL Guide*.

### WITH RETURN

Specifies that the cursor, if it is declared in a stored procedure, can return a result set to the caller.

#### *fetch-first-clause*

Limits the number of rows that can be fetched. It improves the performance of queries with potentially large result sets when only a limited number of rows are needed. If the clause is specified, the number of rows retrieved will not exceed n, where n is the value of the integer. An attempt to fetch n+1 rows is handled the same way as normal end of date. The value of integer must be positive and non-zero. The default is 1.

If the OPTIMIZE FOR clause is not specified, a default of "OPTIMIZE FOR integer ROWS" is assumed. If both the FETCH FIRST and OPTIMIZE FOR clauses are specified, the lower of the integer values from these clauses is used to influence optimization and the communications buffer size.

#### *read-only-clause*

Declares that the result table is read-only and therefore the cursor cannot be referred to in positioned UPDATE and DELETE statements.

#### *update-clause*

Identifies the columns that can be updated in a later positioned UPDATE statement. Each column must be unqualified and must identify a column of the table or view identified in the first FROM clause of the fullselect. The clause must not be specified if the result table of the fullselect is

read-only. The clause must also not be specified if a created temporary table is referenced in the first FROM clause of the select-statement.

If the clause is specified without a list of columns, the columns that can be updated include all the updatable columns of the table or view that is identified in the first FROM clause of the fullselect.

*optimize-clause*

Requests special optimization of the *select-statement*. If the clause is omitted, optimization is based on the assumption that all rows of the result table will be retrieved. If the clause is specified, optimization is based on the assumption that the number of rows retrieved will not exceed  $n$ , where  $n$  is the value of the integer. The clause does not limit the number of rows that can be fetched or affect the result in any way other than performance.

### *isolation-clause*

Specifies the isolation level at which the statement is executed. See “with-clause” on page 373.

**FROM**

Specifies the statement string. The statement string is the value of the specified *string-expression* or the identified *host-variable*.

*string-expression*

*string-expression* is any PL/I expression that yields a string. If the source program does not include any DECLARE VARIABLE statements, an optional colon can precede the *string-expression*. The colon introduces PL/I syntax. Therefore, host variables within a *string-expression* that includes operators or functions should not be preceded with a colon. However, if the source program includes at least one DECLARE VARIABLE statement, a *string-expression* cannot be preceded by a colon and an expression that consists of just a variable name preceded by a colon is interpreted as a *host-variable*, not as a *string-expression*.

The ATTRIBUTES clause cannot be specified following *string-expression*.

The precompiler-generated structures for a *string-expression* use an EBCDIC CCSID.

### *host-variable*

Must identify a host variable that is described in the application program in accordance with the rules for declaring string variables. The host variable must not have a CLOB data type, and an indicator variable must not be specified. In COBOL and Assembler language, the host variable must be a varying-length string variable. In C, the host variable must not be a NUL-terminated string.

In PL/I, if the source program includes at least one DECLARE VARIABLE statement or if the PREPARE statement contains the ATTRIBUTES clause, a host variable (preceded by a colon) is considered a *host-variable* and must be a varying-length string variable. The host variable may be either a fixed-length or varying-length string variable if the source program does not include any DECLARE VARIABLE statements and the PREPARE statement does not include an ATTRIBUTES clause. It is then considered a *string-expression*. When a *string-expression* is used, the precompiler-generated structures for it use an EBCDIC CCSID and an informational message is issued.

## Notes

**Rules for statement strings:** The statement string must be one of the following SQL statements:

|                                |                               |
|--------------------------------|-------------------------------|
| ALLOCATE CURSOR                | LOCK TABLE                    |
| ALTER                          | RENAME                        |
| ASSOCIATE LOCATOR              | REVOKE                        |
| COMMENT ON                     | ROLLBACK                      |
| COMMIT                         | SET CURRENT DEGREE            |
| CREATE                         | SET CURRENT LOCALE LC_CTYPE   |
| DECLARE GLOBAL TEMPORARY TABLE | SET CURRENT OPTIMIZATION HINT |
| DELETE                         | SET CURRENT PRECISION         |
| DROP                           | SET CURRENT RULES             |
| EXPLAIN                        | SET CURRENT SQLID             |
| FREE LOCATOR                   | SET PTH                       |
| GRANT                          | SIGNAL SQLSTATE               |
| HOLD LOCATOR                   | UPDATE                        |
| INSERT                         | select-statement              |
| LABEL ON                       |                               |

The statement string must not:

- Begin with EXEC SQL and end with a statement terminator
- Include references to host variables
- Include comments

**Parameter markers:** Although a statement string cannot include references to host variables, it can include *parameter markers*. The parameter markers are replaced by the values of host variables when the prepared statement is executed. A parameter marker is a question mark (?) that appears where a host variable could appear if the statement string were a static SQL statement. For an explanation of how parameter markers are replaced by values, see “EXECUTE” on page 776, “OPEN” on page 842, and Part 6 of *DB2 Application Programming and SQL Guide*.

The two types of parameter markers are typed and untyped:

### Typed parameter marker

A parameter marker that is specified with its target data type. A typed parameter marker has the general form:

CAST(?) AS data-type)

This notation is not a function call, but rather is a “promise” that the data type of the host variable at run time will be the same as, or can be converted to, the data type that was specified.

In the following example, the value of the argument that is provided for the TRANSLATE function at run time must be VARCHAR(12) or a data type that can be converted to VARCHAR(12).

```
UPDATE EMPLOYEE
 SET LASTNAME = TRANSLATE(CAST(?) AS VARCHAR(12))
 WHERE EMPNO = ?
```

### Untyped parameter marker

A parameter marker that is specified without its target data type. An untyped parameter marker has the form of a single question mark. The context in which the parameter marker appears determines its data type. For example, in the above UPDATE statement, the data type of the untyped parameter marker in the predicate is the same as the data type of the EMPNO column.

Typed parameter markers can be used in dynamic SQL statements wherever a host variable is supported and the data type is based on the promise made in the CAST function.

Untyped parameters markers can be used in dynamic SQL statements in selected locations where host variables are supported. Table 64 shows these locations and the resulting data type of the parameter. The table groups the locations into expressions, predicates, and functions to help show where untyped parameter markers are allowed.

*Table 64. Untyped parameter marker usage*

| Location of untyped parameter marker                                                                                                                                      | Data type (or error if not supported)                                                                                                                                                                      |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Expressions (including select list, CASE, and VALUES)</b>                                                                                                              |                                                                                                                                                                                                            |
| Alone in a select list. For example:<br><code>SELECT ?</code>                                                                                                             | Error                                                                                                                                                                                                      |
| Both operands of a single arithmetic operator, after considering operator precedence and the order of operation rules. Includes cases such as:<br><code>? + ? + 10</code> | Error                                                                                                                                                                                                      |
| One operand of a single operator in an arithmetic expression (except datetime arithmetic expressions). Includes cases such as:<br><code>? + ? * 10</code>                 | The data type of the other operand                                                                                                                                                                         |
| Any operand of a datetime expression. For example:<br><code>'timecol + ?'</code> or<br><code>'? - datecol'</code>                                                         | Error                                                                                                                                                                                                      |
| Labeled duration in a datetime expression                                                                                                                                 | Error                                                                                                                                                                                                      |
| Both operands of a CONCAT operator                                                                                                                                        | Error                                                                                                                                                                                                      |
| One operand of a CONCAT operator when the other operand is any character data type except CLOB                                                                            | If the other operand is CHAR( <i>n</i> ) or VARCHAR( <i>n</i> ), where <i>n</i> is less than 128, the data type is VARCHAR(255 - <i>n</i> ). In all other cases, the data type is VARCHAR(255).            |
| One operand of a CONCAT operator when the other operand is any graphic data type except DBCLOB                                                                            | If the other operand is GRAPHIC( <i>n</i> ) or VARGRAPHIC( <i>n</i> ), where <i>n</i> is less than 64, the data type is VARGRAPHIC(127 - <i>n</i> ). In all other cases, the data type is VARGRAPHIC(127). |
| One operand of a CONCAT operator when the other operand is a LOB string                                                                                                   | The data type of the other operand (the LOB string)                                                                                                                                                        |
| The value on the right-hand side of a SET clause in an UPDATE statement                                                                                                   | The data type of the column or, if the column is defined as a distinct type, the source data type of the distinct type                                                                                     |

## PREPARE

Table 64. Untyped parameter marker usage (continued)

| Location of untyped parameter marker                                                                                                                             | Data type (or error if not supported)                                                                                                                                                                       |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| The expression following the CASE keyword in a simple CASE expression                                                                                            | Error                                                                                                                                                                                                       |
| Any or all expressions following the WHEN keyword in a simple CASE expression                                                                                    | The result of applying the “Rules for result data types” on page 77 to the expression following CASE and the expressions following WHEN that are not untyped parameter markers                              |
| A <i>result-expression</i> in any CASE expression when all the other <i>result-expressions</i> are either NULL or untyped parameter markers.                     | Error                                                                                                                                                                                                       |
| A <i>result-expression</i> in any CASE expression when at least one other <i>result-expression</i> is neither NULL nor an untyped parameter marker.              | The result of applying the “Rules for result data types” on page 77 to all the <i>result-expressions</i> that are not NULL or untyped parameter markers                                                     |
| Alone as a column-expression in a single-row VALUES clause that is not within an INSERT statement                                                                | Error                                                                                                                                                                                                       |
| Alone as a column-expression in a single-row VALUES clause within an INSERT statement                                                                            | The data type of the column or, if the column is defined as a distinct type, the source data type of the distinct type                                                                                      |
| Predicates                                                                                                                                                       |                                                                                                                                                                                                             |
| Both operands of a comparison operator                                                                                                                           | Error                                                                                                                                                                                                       |
| One operand of a comparison operator when the other operand is not an untyped parameter marker                                                                   | The data type of the other operand. If the operand has a datetime data type, the result of DESCRIBE INPUT will show the data type as CHAR(255) although DB2 uses the datetime data type in any comparisons. |
| All the operands of a BETWEEN predicate                                                                                                                          | Error                                                                                                                                                                                                       |
| Two operands of a BETWEEN predicate (either the first and second, or the first and third)                                                                        | The data type of the operand that is not a parameter marker                                                                                                                                                 |
| Only one operand of a BETWEEN predicate                                                                                                                          | The result of applying the “Rules for result data types” on page 77 on the other operands that are not parameter markers                                                                                    |
| All the operands of an IN predicate, for example, ? IN (?, ?, ?)                                                                                                 | Error                                                                                                                                                                                                       |
| The first and second operands of an IN predicate, for example, ? IN (?, A, B)                                                                                    | The result of applying the “Rules for result data types” on page 77 on the operands in the IN list that are not parameter markers                                                                           |
| The first operand of an IN predicate and zero or more operands of the IN list except for the first operand of the IN list, for example, ? IN (A, ?, B, ?)        | The result of applying the “Rules for result data types” on page 77 on the operands in the IN list that are not parameter markers                                                                           |
| The first operand of an IN predicate when the right-hand side is a fullselect of fullselect, for example, ? IN (fullselect)                                      | The data type of the selected column                                                                                                                                                                        |
| Any or all operands of the IN list of the IN predicate and the first operand of the IN predicate is not an untyped parameter marker, for example, A IN (?, A, ?) | The data type of the first operand (the operand on the left-hand side of the IN list)                                                                                                                       |

Table 64. Untyped parameter marker usage (continued)

| Location of untyped parameter marker                                                                                                                                                                                                                                                                                                                                                                                                                                                                     | Data type (or error if not supported)                                                                                                                                    |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| All the operands of a LIKE predicate                                                                                                                                                                                                                                                                                                                                                                                                                                                                     | The first and second operands ( <i>match-expression</i> and <i>pattern-expression</i> ) are VARCHAR(4000). The third operand ( <i>escape-expression</i> ) is VARCHAR(1). |
| The first operand (the <i>match-expression</i> ) when at least one other operand (the <i>pattern-expression</i> or <i>escape-expression</i> ) is not an untyped parameter marker.                                                                                                                                                                                                                                                                                                                        | VARCHAR(4000), VARGRAPHIC(2000), or BLOB(4000), depending on the data type of the first operand that is not an untyped parameter marker                                  |
| The second operand (the <i>pattern-expression</i> ) when at least one other operand (the <i>match-expression</i> or <i>escape-expression</i> ) is not an untyped parameter marker. When the pattern specified in a LIKE predicate is a parameter marker and a fixed-length character host variable is used to replace the parameter marker, specify a value for the host variable that is the correct length. If you do not specify the correct length, the select does not return the intended results. | VARCHAR(4000), VARGRAPHIC(2000), or BLOB(4000), depending on the data type of the first operand that is not an untyped parameter marker.                                 |
| The third operand (the <i>escape-expression</i> ) when at least one other operand (the <i>match-expression</i> or <i>pattern-expression</i> ) is not an untyped parameter marker                                                                                                                                                                                                                                                                                                                         | CHAR(1), GRAPHIC(1), or BLOB(1), depending on the data type of the first operand that is not an untyped parameter marker                                                 |
| Operand of a NULL predicate                                                                                                                                                                                                                                                                                                                                                                                                                                                                              | Error                                                                                                                                                                    |
| Functions                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |                                                                                                                                                                          |
| All operands of COALESCE (also called VALUE) or NULLIF                                                                                                                                                                                                                                                                                                                                                                                                                                                   | Error                                                                                                                                                                    |
| Any operand of COALESCE or NULLIF when at least one other operand is not an untyped parameter marker                                                                                                                                                                                                                                                                                                                                                                                                     | The result of applying the “Rules for result data types” on page 77 on the operands that are not untyped parameter markers, the data type of the other operand           |
| Both operands of POSSTR                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  | VARCHAR(4000) for both operands                                                                                                                                          |
| One operand of POSSTR when the other operand is a character data type                                                                                                                                                                                                                                                                                                                                                                                                                                    | VARCHAR(4000)                                                                                                                                                            |
| One operand of POSSTR when the other operand is a graphic data type                                                                                                                                                                                                                                                                                                                                                                                                                                      | VARGRAPHIC(2000)                                                                                                                                                         |
| One operand of POSSTR when the other operand is a BLOB                                                                                                                                                                                                                                                                                                                                                                                                                                                   | BLOB(4000)                                                                                                                                                               |
| First operand of SUBSTR                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  | VARCHAR(4000)                                                                                                                                                            |
| Second or third operand of SUBSTR                                                                                                                                                                                                                                                                                                                                                                                                                                                                        | INTEGER                                                                                                                                                                  |
| One operand of TIMESTAMP                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 | TIME                                                                                                                                                                     |
| First operand of TIMESTAMP_FORMAT                                                                                                                                                                                                                                                                                                                                                                                                                                                                        | VARCHAR(255)                                                                                                                                                             |
| First operand of TRANSLATE                                                                                                                                                                                                                                                                                                                                                                                                                                                                               | Error                                                                                                                                                                    |
| Second or third operand of TRANSLATE                                                                                                                                                                                                                                                                                                                                                                                                                                                                     | VARCHAR(4000) or VARGRAPHIC(2000), depending on whether the data type of the first operand is character or graphic                                                       |
| Fourth operand of TRANSLATE                                                                                                                                                                                                                                                                                                                                                                                                                                                                              | VARCHAR(1) or VARGRAPHIC(1), depending on whether the data type of the first operand is character or graphic                                                             |

## PREPARE

Table 64. Untyped parameter marker usage (continued)

| Location of untyped parameter marker                                                                                                                                                                                 | Data type (or error if not supported)                                    |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------|
| First operand of VARCHAR_FORMAT                                                                                                                                                                                      | TIMESTAMP                                                                |
| Unary minus                                                                                                                                                                                                          | DOUBLE PRECISION                                                         |
| Unary plus                                                                                                                                                                                                           | DOUBLE PRECISION                                                         |
| Operand of any built-in scalar function<br>(except those that are described above in this<br>table for COALESCE, NULLIF, POSSTR,<br>SUBSTR, TIMESTAMP,<br>TIMESTAMP_FORMAT, TRANSLATE,<br>VALUE, and VARCHAR_FORMAT) | Error                                                                    |
| Operand of a built-in column function                                                                                                                                                                                | Error                                                                    |
| Operand of a user-defined scalar function,<br>user-defined column function, or user-defined<br>table function                                                                                                        | The data type of the corresponding parameter<br>in the function instance |

**Error checking:** When a PREPARE statement is executed, the statement string is parsed and checked for errors. If the statement string is invalid, a prepared statement is not created and the error condition that prevents its creation is reported in the SQLCA.

In local and remote processing, the DEFER(PREPARE) and REOPT(VARS) bind options can cause some SQL statements to receive “delayed” errors. For example, DESCRIBE, EXECUTE, and OPEN might receive an SQLCODE that normally occurs during PREPARE processing.

**Reference and execution rules:** Prepared statements can be referred to in the following kinds of statements, with the following restrictions shown:

**In... The prepared statement...**

**DESCRIBE**

has no restrictions

**DECLARE CURSOR**

must be SELECT when the cursor is opened

**EXECUTE**

must *not* be SELECT

A prepared statement can be executed many times. Indeed, if a prepared statement is not executed more than once and does not contain parameter markers, it is more efficient to use the EXECUTE IMMEDIATE statement rather than the PREPARE and EXECUTE statements.

**Prepared statement persistence:** All prepared statements created by a unit of work are destroyed when the unit of work is terminated, with the following exceptions:

- A SELECT statement whose cursor is declared with the option WITH HOLD persists over the execution of a commit operation if the cursor is open when the commit operation is executed.
- SELECT, INSERT, UPDATE, and DELETE statements that are bound with KEEPDYNAMIC(YES) are kept past the commit operation if your system is enabled for dynamic statement caching, and none of the following are true:
  - SQL RELEASE has been issued for the site
  - Bind option DISCONNECT(AUTOMATIC) was used

- Bind option DISCONNECT(CONDITIONAL) was used and there are no hold cursors for the site

**Scope of a statement name:** The scope of a *statement-name* is the same as the scope of a *cursor-name*. See “Notes” on page 721 for more information about the scope of a *cursor-name*.

**Preparation with PREPARE INTO and REOPTVAR:** If bind option REOPT(VARS) is in effect, PREPARE INTO is equivalent to a PREPARE and a DESCRIBE being performed. If a statement has input variables, the DESCRIBE causes the statement to be prepared with default values, and the statement must be prepared again when it is opened or executed. To avoid having a statement prepared twice, avoid using PREPARE INTO when REOPT(VARS) is in effect.

**Relationship of cursor attributes on PREPARE statements and SELECT or DECLARE CURSOR statements:** Cursor attributes that are specified as part of the *select-statement* are used instead of any corresponding options that specified with the ATTRIBUTES clause on PREPARE. Attributes that are specified as part of the ATTRIBUTES clause of PREPARE take precedence over any corresponding option that is specified with the DECLARE CURSOR statement. The order for using cursor attributes is as follows:

- SELECT (highest priority)
- PREPARE statement ATTRIBUTES clause
- DECLARE CURSOR (lowest priority)

For example, assume that host variable MYQ has been set to the following SELECT statement:

```
SELECT WORKDEPT, EMPNO, SALARY, BONUS, COMM
 FROM EMP
 WHERE WORKDEPT IN ('D11', 'D21')
 FOR UPDATE OF SALARY, BONUS, COMM
```

If the following PREPARE statement were issued, then the FOR UPDATE clause specified as part of the SELECT statement would be used instead of the FOR READ ONLY clause specified with the ATTRIBUTES clause as part of the PREPARE statement. Thus, the cursor would be updatable.

```
attrstring = 'FOR READ ONLY';
EXEC SQL PREPARE stmt1 ATTRIBUTES :attrstring FROM :MYQ;
```

## Examples

**Example 1:** In this PL/I example, an INSERT statement with parameter markers is prepared and executed. Before execution, values for the parameter markers are read into the host variables S1, S2, S3, S4, and S5.

```
EXEC SQL PREPARE DEPT_INSERT FROM
 'INSERT INTO DSN8710.DEPT VALUES(?, ?, ?, ?, ?)';
 (Check for successful execution and read values into host variables)
EXEC SQL EXECUTE DEPT_INSERT USING :S1, :S2, :S3, :S4, :S5;
```

**Example 2:** Prepare a dynamic SELECT statement specifying the attributes of the cursor with a host variable on the PREPARE statement. Assume that the text of the SELECT statement is in a variable named stmttxt, and that the desired attributes of the cursor are in a variable named attrvar.

## **PREPARE**

```
| EXEC SQL DECLARE mycursor CURSOR FOR mystmt;
| EXEC SQL PREPARE mystmt ATTRIBUTES :attrvar
| FROM :stmttxt;
| EXEC SQL DESCRIBE mystmt INTO :mysqllda;
| EXEC SQL OPEN mycursor;
| EXEC SQL FETCH FROM mycursor USING DESCRIPTOR :mysqllda ;
|
```

## RELEASE (connection)

The RELEASE (connection) statement places one or more connections in the release pending state.

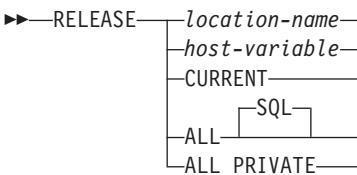
### Invocation

This statement can only be embedded in an application program, except in REXX programs. It is an executable statement that cannot be dynamically prepared.

### Authorization

None required.

### Syntax



### Description

#### *location-name* or *host-variable*

Identifies an SQL connection or a DB2 private connection by the specified location name or the location name contained in the host variable. If a host variable is specified:

- It must be a character string variable with a length attribute that is not greater than 16. (A C NUL-terminated character string can be up to 17 bytes.)
- It must not be followed by an indicator variable.
- The location name must be left-justified within the host variable and must conform to the rules for forming an ordinary location identifier.
- If the length of the location name is less than the length of the host variable, it must be padded on the right with blanks.

The specified location name or the location name contained in the host variable must identify an existing SQL connection or DB2 private connection of the application process.

If the RELEASE (connection) statement is successful, the identified connection is placed in the release pending status and will therefore be ended during the next commit operation. If the RELEASE (connection) statement is unsuccessful, the connection state of the application process and the states of its connections are unchanged.

#### CURRENT

Identifies the current SQL connection of the application process. The application process must be in the connected state.

If the RELEASE (connection) statement is successful, the identified connection is placed in the release pending state and will therefore be ended during the

## **RELEASE (connection)**

next commit operation. If the RELEASE (connection) statement is unsuccessful, the connection state of the application process and the states of its connections are unchanged.

### **ALL or ALL SQL**

Identifies all existing connections (including local, SQL, and DB2 private connections) of the application process and places these connections in the release pending status. These connections are ended during the next commit operation. An error or warning does not occur if no connections exist when the statement is executed.

### **ALL PRIVATE**

Identifies all existing DB2 private connections of the application process and places these connections in the release pending status. These DB2 private connections are ended during the next commit operation. An error or warning does not occur if no DB2 private connections exist when the statement is executed.

## **Notes**

Using CONNECT (Type 1) semantics does not prevent using RELEASE (connection).

RELEASE (connection) does not close cursors, does not release any resources, and does not prevent further use of the connection.

ROLLBACK does not reset the state of a connection from release pending to held.

Resources are required to create and maintain remote connections. Thus, a remote connection that is not going to be reused should be in the release pending status and one that is going to be reused should not be in the release pending status. Remote connections can also be ended during a commit operation as a result of the DISCONNECT(AUTOMATIC) or DISCONNECT(CONDITIONAL) bind option.

If the current SQL connection is in the release pending status when a commit operation is performed, the connection is ended and the application process is in the unconnected state. In this case, the next executed SQL statement should be CONNECT or SET CONNECTION.

A database server named CURRENT or ALL can only be identified by a host variable or a delimited identifier. A connection in the release pending state is ended during a commit operation even though it has an open cursor defined with WITH HOLD.

For further information, see “When a connection is ended” on page 20.

If the RELEASE statement contains host variables, the contents of the host variables are assumed to be in the encoding scheme that was specified in the ENCODING parameter when the package or plan that contains the statement was bound.

## **Examples**

*Example 1:* The SQL connection to TOROLAB1 is not needed in the next unit of work. The following statement causes it to be ended during the next commit operation:

```
EXEC SQL RELEASE TOROLAB1;
```

## RELEASE (connection)

*Example 2:* The current SQL connection is not needed in the next unit of work. The following statement causes it to be ended during the next commit operation:

```
EXEC SQL RELEASE CURRENT;
```

*Example 3:* The first phase of an application involves explicit CONNECTs to remote servers and the second phase involves the use of DB2 private protocol access with the local DB2 subsystem as the server. None of the existing connections are needed in the second phase and their existence could prevent the allocation of DB2 private connections. Accordingly, the following statement is executed before the commit operation that separates the two phases:

```
EXEC SQL RELEASE ALL SQL;
```

*Example 4:* The first phase of an application involves the use of DB2 private protocol access with the local DB2 subsystem as the server and the second phase involves explicit CONNECTs to remote servers. The existence of the DB2 private connections allocated during the first phase could cause a CONNECT operation to fail. Accordingly, the following statement is executed before the commit operation that separates the two phases:

```
EXEC SQL RELEASE ALL PRIVATE;
```

## RELEASE SAVEPOINT

### RELEASE SAVEPOINT

The RELEASE SAVEPOINT statement releases the identified savepoint and any subsequently established savepoints within a unit of recovery.

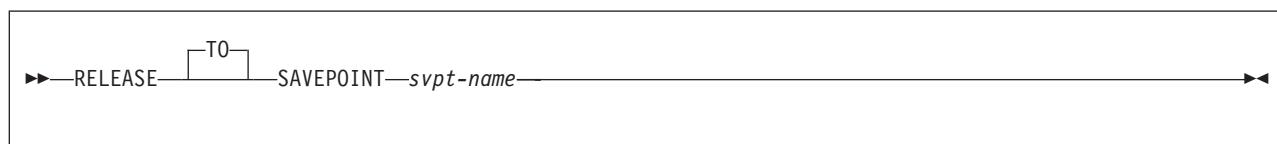
#### Invocation

This statement can be imbedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared. This statement cannot be issued from a global transaction.

#### Authorization

None required.

#### Syntax



#### Description

##### *svpt-name*

A savepoint identifier that identifies the savepoint to release. If the named savepoint does not exist, an error occurs. The named savepoint and all the savepoints that were subsequently established in the unit of recovery are released. After a savepoint is released, it is no longer maintained and rollback to the savepoint is no longer possible.

#### Example

Assume that a main routine sets savepoint A and then invokes a subroutine that sets savepoints B and C. When control returns to the main routine, release savepoint A and any subsequently set savepoints. Savepoints B and C, which were set by the subroutine, are released in addition to A.

```
:
RELEASE SAVEPOINT A;
```

## RENAME

The RENAME statement renames an existing table.

### Invocation

This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is implicitly or explicitly specified.

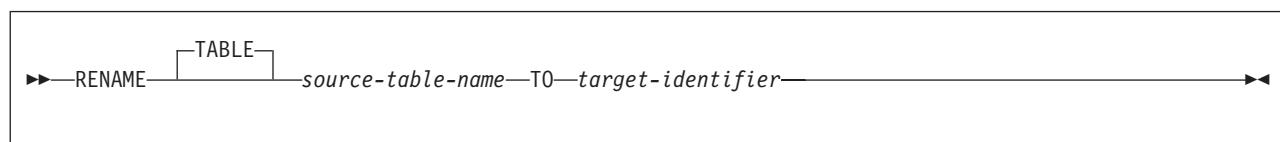
### Authorization

The privilege set that is defined below must include at least one of the following:

- Ownership of the table
- DBADM, DBCTRL, or DBMAINT authority for the database that contains the table
- SYSADM or SYSCTRL authority

**Privilege set:** If the statement is embedded in an application program, the privilege set is the privileges held by the authorization ID of the owner of the plan or package. If the statement is dynamically prepared, the privilege set is the union of the privilege sets that are held by each authorization ID of the process.

### Syntax



### Description

#### *source-table-name*

Identifies the existing table that is to be renamed. The name, including the implicit or explicit qualifier, must identify a table that exists at the current server. The name must not identify a table that has a trigger defined on it, a declared temporary table, a catalog table, an active RLST table, a view, a synonym, or a table with a synonym defined on it. If you specify a three-part name or alias for the source, the source table must exist at the current server.

There is no support for changing the name of an alias. An alias on the source table continues to refer to the source table after the rename.

If any view definitions currently refer to the source table, an error is issued.

The specified table is renamed to the new name. All privileges and indexes on the table are preserved.

#### *target-identifier*

Specifies the new name for the table without a qualifier. The qualifier of the *source-table-name* is used to qualify the new name for the table. The qualified name must *not* identify a table, view, alias, or synonym that already exists at the current server.

### Notes

**Catalog Table Updates:** Entries in the following catalog tables are updated to reflect the new table name:

- SYSAUXRELS

## RENAME

- SYSCHECKS
- SYSCHECKS2
- SYSCHECKDEP
- SYSCOLAUTH
- SYSCOLDIST
- SYSCOLDISTSTATS
- SYSCOLSTATS
- SYSCOLUMNS
- SYSFIELDS
- SYSFOREIGNKEYS
- SYSINDEXES
- SYSPLANDEP
- SYSPACKDEP
- SYSRELS
- SYSSYNONYMS
- SYSTABAUTH
- SYSTABLES
- SYSTABSTATS

Entries in SYSSTMT and SYSPACKSTMT are not updated.

**Invalidation of plans, packages, and dynamic statements:** When any table except an auxiliary table is renamed, plans and packages that refer to that table are invalidated. If any dynamic statements in the statement cache refer to the table, they are invalidated; DB2 must refresh those statements in the cache the next time they are executed.

When an auxiliary table is renamed, plans and packages that refer to the auxiliary table are not invalidated.

**Transfer of authorization, referential integrity constraints, and indexes:** All authorizations associated with the source table name are *transferred* to the new (target) table name. The authorization catalog tables are updated appropriately.

Referential integrity constraints involving the source table are updated to refer to the new table. The catalog tables are updated appropriately.

Indexes defined over the source table are *transferred* to the new table. The index catalog tables are updated appropriately.

**Object Identifier:** Renamed tables keep the same object identifier (OBID) as the original table.

**Renaming Registration Tables:** If an application registration table or object registration table is specified as the source table for RENAME, then once RENAME completes, it is as if that table had been dropped. There is no ART (application registration table) or ORT (object registration table) once the ART or ORT table has been renamed.

## Example

Change the name of the EMP table to EMPLOYEE:

```
RENAME TABLE EMP TO EMPLOYEE;
```

---

## REVOKE

The REVOKE statement revokes privileges from authorization IDs. There is a separate form of the statement for each of these classes of privilege:

- Collection
- Database
- Distinct type
- Function or stored procedure
- Package
- Plan
- Schema
- System
- Table or view
- Use

The applicable objects are always at the current server.

### Invocation

This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is implicitly or explicitly specified.

If the authorization mechanism was not activated when the DB2 subsystem was installed, an error condition occurs.

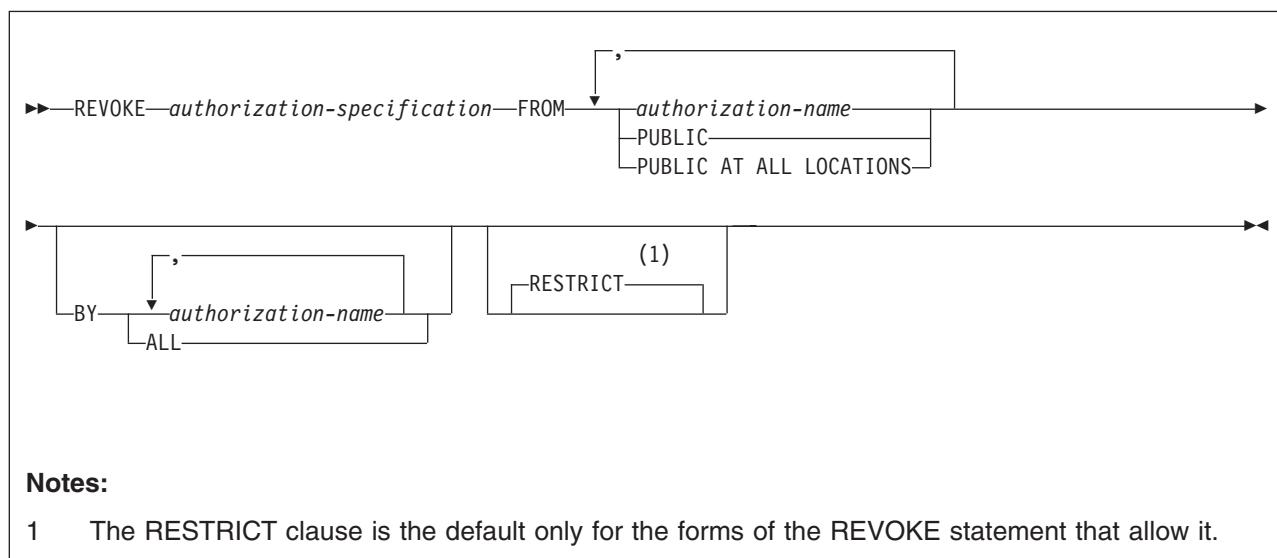
### Authorization

If the BY clause is not specified, the authorization ID of the statement must have granted at least one of the specified privileges to every *authorization-name* specified in the FROM clause (including PUBLIC, if specified). If the BY clause is specified, the authorization ID of the statement must have SYSADM or SYSCTRL authority.

If the statement is embedded in an application program, the authorization ID of the statement is the authorization ID of the owner of the plan or package. If the statement is dynamically prepared, the authorization ID of the statement is the SQL authorization ID of the process.

## REVOKE

### Syntax



#### Notes:

- 1 The RESTRICT clause is the default only for the forms of the REVOKE statement that allow it.

### Description

#### *authorization-specification*

Names one or more privileges, in one of the formats described below. The same privilege must not be specified more than once.

#### **FROM**

Specifies from what authorization IDs the privileges are revoked.

#### *authorization-name*,...

Lists one or more authorization IDs. Do not use the same authorization ID more than once.

The value of CURRENT RULES determines if you can use the ID of the REVOKE statement itself (to revoke privileges from yourself). When CURRENT RULES is:

#### **DB2**

You cannot use the ID of the REVOKE statement.

#### **STD**

You can use the ID of the REVOKE statement.

#### **PUBLIC**

Revokes a grant of privileges to PUBLIC.

#### **PUBLIC AT ALL LOCATIONS**

Revokes a grant of privileges to PUBLIC AT ALL LOCATIONS.

#### **BY**

Lists grantors who have granted privileges and revokes each named privilege that was explicitly granted to some named user by one of the named grantors. Only an authorization ID with SYSADM or SYSCTRL authority can use BY, even if the authorization ID names only itself in the BY clause.

#### *authorization-name*,...

Lists one or more authorization IDs of users who were the grantors of the privileges named. Do not use the same authorization ID more than once.

Each grantor listed must have explicitly granted some named privilege to all named users.

**ALL**

Revokes each named privilege from all named users who were explicitly granted the privilege, regardless of who granted it.

**RESTRICT**

Prevents the named privilege from being revoked when certain conditions apply. RESTRICT is the default only for the forms of the REVOKE statement that allow it. These forms are revoking the USAGE privilege on distinct types and the EXECUTE privilege on user-defined functions and stored procedures.

**Notes**

For more on DB2 privileges, read Part 3 (Volume 1) of *DB2 Administration Guide*. For information on access control authorization, see Appendix B (Volume 2) of *DB2 Administration Guide*.

**Revoked privileges:** The privileges revoked from an authorization ID are those that are identified in the statement and which were granted to the authorization ID by the authorization ID of the statement. Other privileges can be revoked as the result of a cascade revoke.

**Cascade revoke:** Revoking a privilege from a user can also cause that privilege to be revoked from other users. This is called a *cascade revoke*. The following rules must be true for privilege P' to be revoked from U3 when U1 revokes privilege P from U2:

- P and P' are the same privilege.
- U2 granted privilege P' to U3.
- No one granted privilege P to U2 prior to the grant by U1.
- U2 does not have installation SYSADM authority.

The rules also apply to the implicit grants that are made as a result of a CREATE VIEW statement.

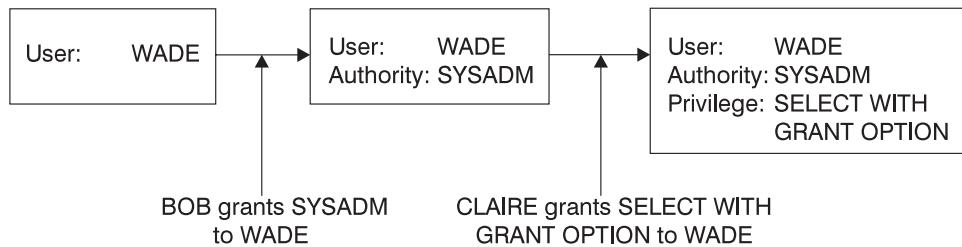
Cascade revoke does not occur under any of the following conditions:

- The privilege was granted by a current install SYSADM.
- The privilege is the USAGE privilege on a distinct type and the user owns any of these items:
  - A user-defined function or stored procedure that uses the distinct type
  - A table that has a column that uses the distinct type
- The privilege is the EXECUTE privilege on a user-defined function and the user owns any of these items:
  - A user-defined function that is sourced on the function
  - A view that uses the function
  - A trigger package that uses the function
  - A table that uses the function in a check constraint or a user-defined default type
- The privilege is the EXECUTE privilege on a stored procedure and the user owns any of these items:
  - A trigger package that refers to the stored procedure in a CALL statement.

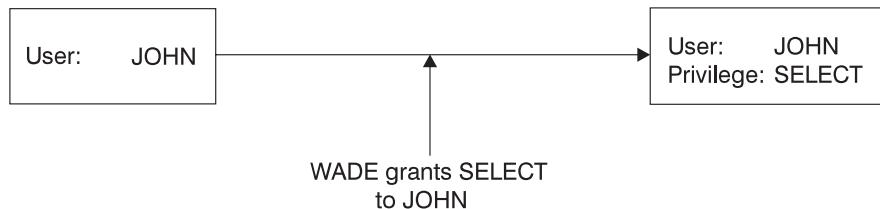
Refer to the diagrams for the following example:

1. Suppose BOB grants SYSADM authority to WADE. Later, CLAIRE grants the SELECT privilege on a table with the WITH GRANT OPTION to WADE.

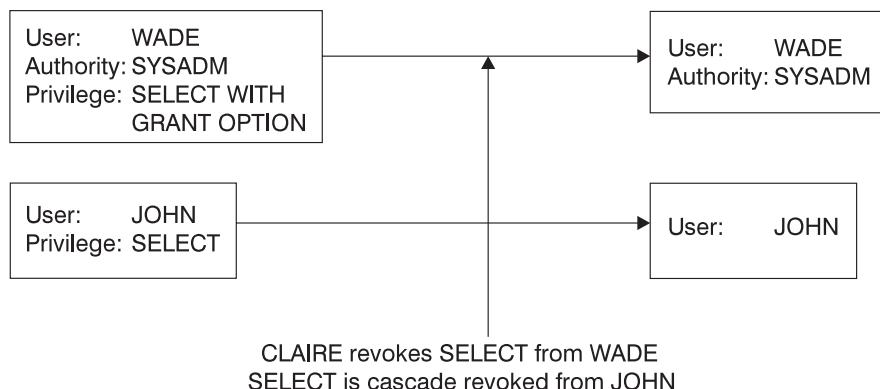
## REVOKE



2. WADE grants the SELECT privilege to JOHN on the same table.



3. When CLAIRE revokes the SELECT privilege on the table from WADE, the SELECT privilege on that table is also revoked from JOHN.



The grant from WADE to JOHN is removed because WADE had not been granted the SELECT privilege from any other source before CLAIRE made the grant. The SYSADM authority granted to WADE from BOB does not affect the cascade revoke. For more on SYSADM and install SYSADM authority, see Part 3 (Volume 1) of *DB2 Administration Guide*. For another example of cascading revokes, see Part 3 (Volume 1) of *DB2 Administration Guide*.

Revoking a SELECT privilege that was exercised to create a view causes the view to be dropped, unless the owner of the view was directly granted the SELECT privilege from another source before the view was created. Revoking a SYSADM privilege that was required to create a view causes the view to be dropped. For details on when SYSADM authority is required to create a view, see *Authorization in "CREATE VIEW"* on page 711.

**Invalidation of plans and packages:** A revoke or cascaded revoke of any privilege, excluding the EXECUTE privilege on a user-defined function, that was

38. Dependencies on stored procedures can be checked only if the procedure name is specified as a literal and not via a host variable in the CALL statement.

exercised to create a plan or package makes the plan or package invalid when the revokee no longer holds the privilege from any other source. Corresponding authorization caches are cleared even if the revokee has the privilege from any other source.<sup>38</sup>

**Inoperative plans and packages:** A revoke or cascaded revoke of the EXECUTE privilege on a user-defined function that was exercised to create a plan or package makes the plan or package inoperative and causes the corresponding authorization caches to be cleared when the revokee no longer holds the privilege from any other source.<sup>38</sup>

**Invalidation of dynamic statements in the cache:** If a dynamic SQL statement is cached and its authorization involved a DELETE, INSERT, SELECT, or UPDATE table privilege, or the EXECUTE function or stored procedure privilege, revoking the privilege makes the statement invalid. DB2 will have to prepare the statement the next time that it is executed.

**Multiple grants:** If you granted the same privilege to the same user more than once, revoking that privilege from that user nullifies all those grants. It does not nullify any grant of that privilege made by others.

When a REVOKE statement revokes multiple grants, the grants are revoked, one at a time, in an undefined order. If, for some reason, a revocation is in error, the execution of the statement is stopped, and all the revoked grants are restored. Such an error certainly occurs if a table or view is specified twice after the keyword ON. One also occurs when a table and a view based on the table are both specified after ON. The error would occur if revoking some grant for the table causes the view to be dropped before all grants have been revoked for the view.

**Privileges belonging to an authority:** You can revoke an administrative authority, but you cannot separately revoke the specific privileges inherent in that administrative authority.

Let P be a privilege inherent in authority X. A user with authority X can also have privilege P as a result of an explicit grant of P. In this case:

- If X is revoked, the user still has privilege P.
- If P is revoked, the user still has the privilege because it is inherent in X.

**Ownership privileges:** The privileges inherent in the ownership of an object cannot be revoked.

**PUBLIC AT ALL LOCATIONS:** PUBLIC AT ALL LOCATIONS can continue to be specified as an alternative to PUBLIC as in prior releases. PUBLIC AT ALL LOCATIONS is intended for use only with DB2 private protocol access.

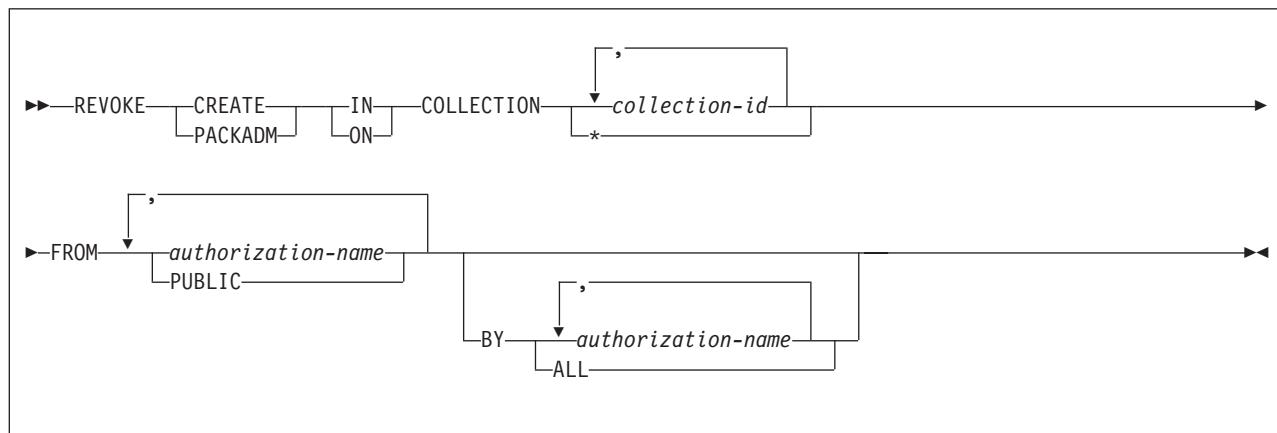
|  
|  
|

## REVOKE (collection privileges)

## REVOKE (collection privileges)

This form of the REVOKE statement revokes privileges on collections.

### Syntax



### Description

#### CREATE IN

Revokes the privilege to use the BIND subcommand to create packages in the designated collections.

The word ON can be used instead of IN.

#### PACKADM ON

Revokes the package administrator authority for the designated collections.

The word IN can be used instead of ON.

#### COLLECTION *collection-id*,...

Identifies the collections on which the specified privilege is revoked. For each identified collection, you (or the indicated grantors) must have granted the specified privilege on that collection to all identified users (including PUBLIC if specified). The same collection must not be identified more than once.

#### COLLECTION \*

Indicates that the specified privilege on COLLECTION \* is revoked. You (or the indicated grantors) must have granted the specified privilege on COLLECTION \* to all identified users (including PUBLIC if specified). Privileges granted on specific collections are not affected.

#### FROM

Refer to "REVOKE" on page 865 for a description of the FROM clause.

#### BY

Refer to "REVOKE" on page 865 for a description of the BY clause.

### Example

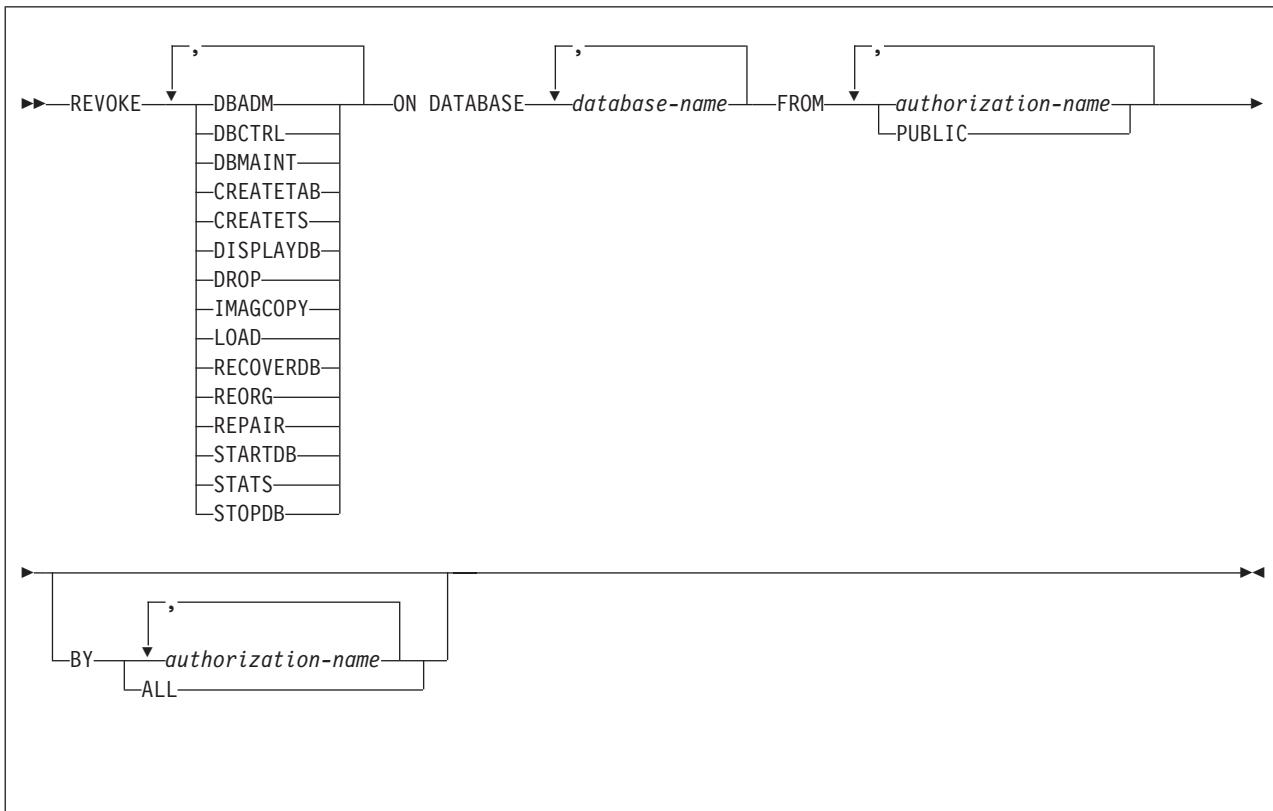
Revoke the privilege to create new packages in collections QAACLONE and DSN8CC61 from CLARK.

```
REVOKE CREATE IN COLLECTION QAACLONE, DSN8CC61 FROM CLARK;
```

## REVOKE (database privileges)

This form of the REVOKE statement revokes database privileges.

### Syntax



### Description

Each keyword listed revokes the privilege described, but only as it applies to or within the databases named in the statement.

#### **DBADM**

Revokes the database administrator authority.

#### **DBCTRL**

Revokes the database control authority.

#### **DBMAINT**

Revokes the database maintenance authority.

#### **CREATETAB**

Revokes the privilege to create new tables. For a TEMP database, you cannot revoke the privilege from PUBLIC. When a TEMP database is created, PUBLIC implicitly receives the CREATETAB privilege (without GRANT authority); this privilege is not recorded in the DB2 catalog, and it cannot be revoked.

#### **CREATESTS**

Revokes the privilege to create new table spaces.

#### **DISPLAYDB**

Revokes the privilege to issue the DISPLAY DATABASE command.

## REVOKE (database privileges)

### **DROP**

Revokes the privilege to issue the DROP or ALTER statements in the specified databases.

### **IMAGCOPY**

Revokes the privilege to run the COPY, MERGECOPY, and QUIESCE utilities against table spaces of the specified databases, and to run the MODIFY utility.

### **LOAD**

Revokes the privilege to use the LOAD utility to load tables.

### **RECOVERDB**

Revokes the privilege to use the RECOVER and REPORT utilities to recover table spaces and indexes.

### **REORG**

Revokes the privilege to use the REORG utility to reorganize table spaces and indexes.

### **REPAIR**

Revokes the privilege to use the REPAIR and DIAGNOSE utilities.

### **STARTDB**

Revokes the privilege to issue the START DATABASE command.

### **STATS**

Revokes the privilege to use the RUNSTATS utility to update statistics, and the CHECK utility to test whether indexes are consistent with the data they index.

### **STOPDB**

Revokes the privilege to issue the STOP DATABASE command.

### **ON DATABASE *database-name*,...**

Identifies databases on which you are revoking the privileges. For each database you identify, you (or the indicated grantors) must have granted at least one of the specified privileges on that database to all identified users (including PUBLIC, if specified). The same database must not be identified more than once.

### **FROM**

Refer to “REVOKE” on page 865 for a description of the FROM clause.

### **BY**

Refer to “REVOKE” on page 865 for a description of the BY clause.

## Examples

*Example 1:* Revoke drop privileges on database DSN8D71A from user PEREZ.

```
REVOKE DROP
 ON DATABASE DSN8D71A
 FROM PEREZ;
```

*Example 2:* Revoke repair privileges on database DSN8D71A from all local users.  
(Grants to specific users will not be affected.)

```
REVOKE REPAIR
 ON DATABASE DSN8D71A
 FROM PUBLIC;
```

*Example 3:* Revoke authority to create new tables and load tables in database DSN8D71A from users WALKER, PIANKA, and FUJIMOTO.

## **REVOKE (database privileges)**

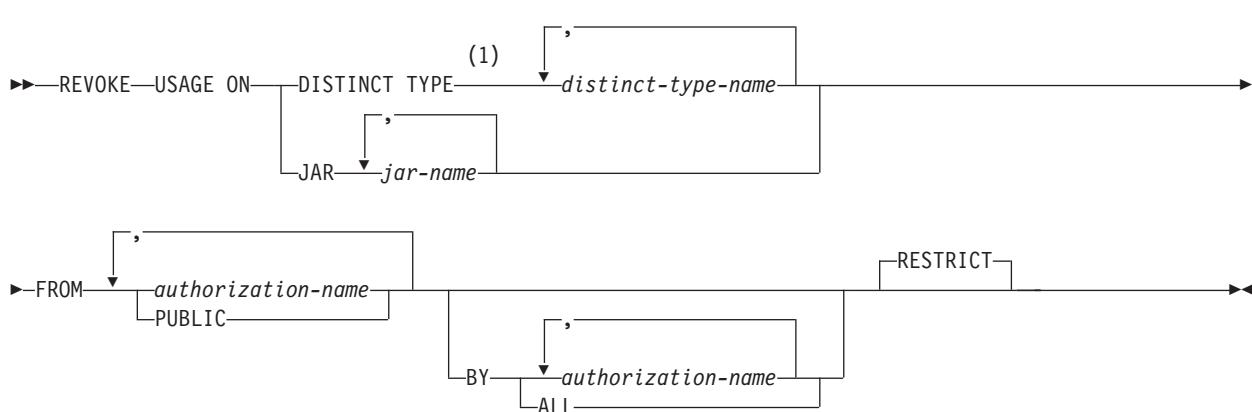
```
REVOKE CREATETAB,LOAD
ON DATABASE DSN8D71A
FROM WALKER,PIANKA,FUJIMOTO;
```

## REVOKE (distinct type or JAR privileges)

### REVOKE (distinct type or JAR privileges)

This form of the REVOKE statement revokes the privilege to use distinct types (user-defined data types) or JARs.

### Syntax



#### Notes:

- 1 DATA can be used as a synonym for DISTINCT. DISTINCT is the keyword that is used on CREATE.

## Description

### USAGE

Revokes the privilege to use the identified distinct types or revokes the privilege to use the identified JARs.

### DISTINCT TYPE *distinct-type-name*

Identifies a distinct type. The name, including the implicit or explicit schema qualifier, must identify a unique distinct type that exists at the current server. If you specify an unqualified name, the name is implicitly qualified with a schema name according to the following rules:

- If the statement is embedded in a program, the schema name is the authorization ID in the QUALIFIER option when the plan or package was created or last rebound. If QUALIFIER was not used, the schema name is the owner of the plan or package.
- If the statement is dynamically prepared, the schema name is the SQL authorization ID in the CURRENT\_SQLID special register.

### JAR *jar-name*

Identifies the JAR. The name, including the implicit or explicit schema name, must identify a unique JAR that exists at the current server. If you do not explicitly qualify the JAR name, it is implicitly qualified with a schema name according to the following rules:

- If the statement is embedded in a program, the schema name is the authorization ID in the QUALIFIER option when the plan or package was created or last rebound. If QUALIFIER was not used, the schema name is the owner of the plan or package.

## REVOKE (distinct type or JAR privileges)

- If the statement is dynamically prepared, the schema name is the SQL authorization ID in the CURRENT\_SQLID special register.

### FROM

Refer to “REVOKE” on page 865 for a description of the FROM clause.

### BY

Refer to “REVOKE” on page 865 for a description of the BY clause.

### RESTRICT

Prevents the USAGE privilege from being revoked on a distinct type or JAR if any of the following conditions exist:

- The revokee owns a function or stored procedure that uses the distinct type or references the JAR.
- The revokee owns a table that has a column that uses the distinct type.

## Examples

*Example 1:* Revoke the USAGE privilege on distinct type SHOESIZE from user JONES.

```
REVOKE USAGE ON DISTINCT TYPE SHOESIZE FROM JONES;
```

*Example 2:* Revoke the USAGE privilege on distinct type US\_DOLLAR from all users at the current server except for those who have been specifically granted USAGE and not through PUBLIC.

```
REVOKE USAGE ON DISTINCT TYPE US_DOLLAR FROM PUBLIC;
```

*Example 3:* Revoke the USAGE privilege on distinct type CANADIAN\_DOLLARS from the administrative assistant (ADMIN\_A).

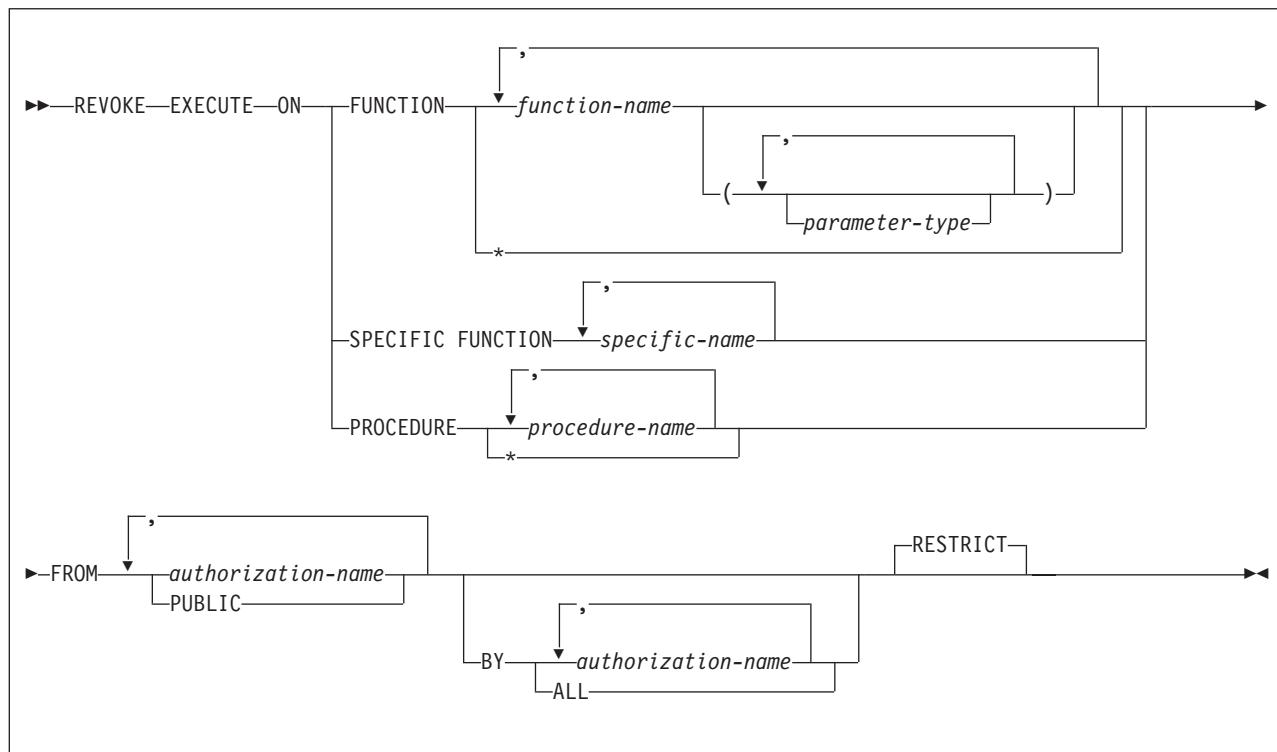
```
REVOKE USAGE ON DISTINCT TYPE CANADIAN_DOLLARS
FROM ADMIN_A;
```

### **REVOKE (function or procedure privileges)**

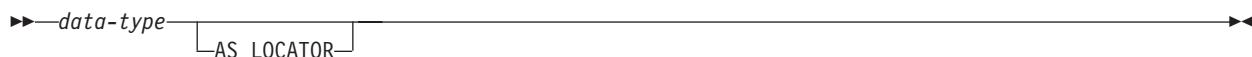
## **REVOKE (function or procedure privileges)**

This form of the REVOKE statement revokes privileges on user-defined functions, cast functions that were generated for distinct types, and stored procedures.

## Syntax

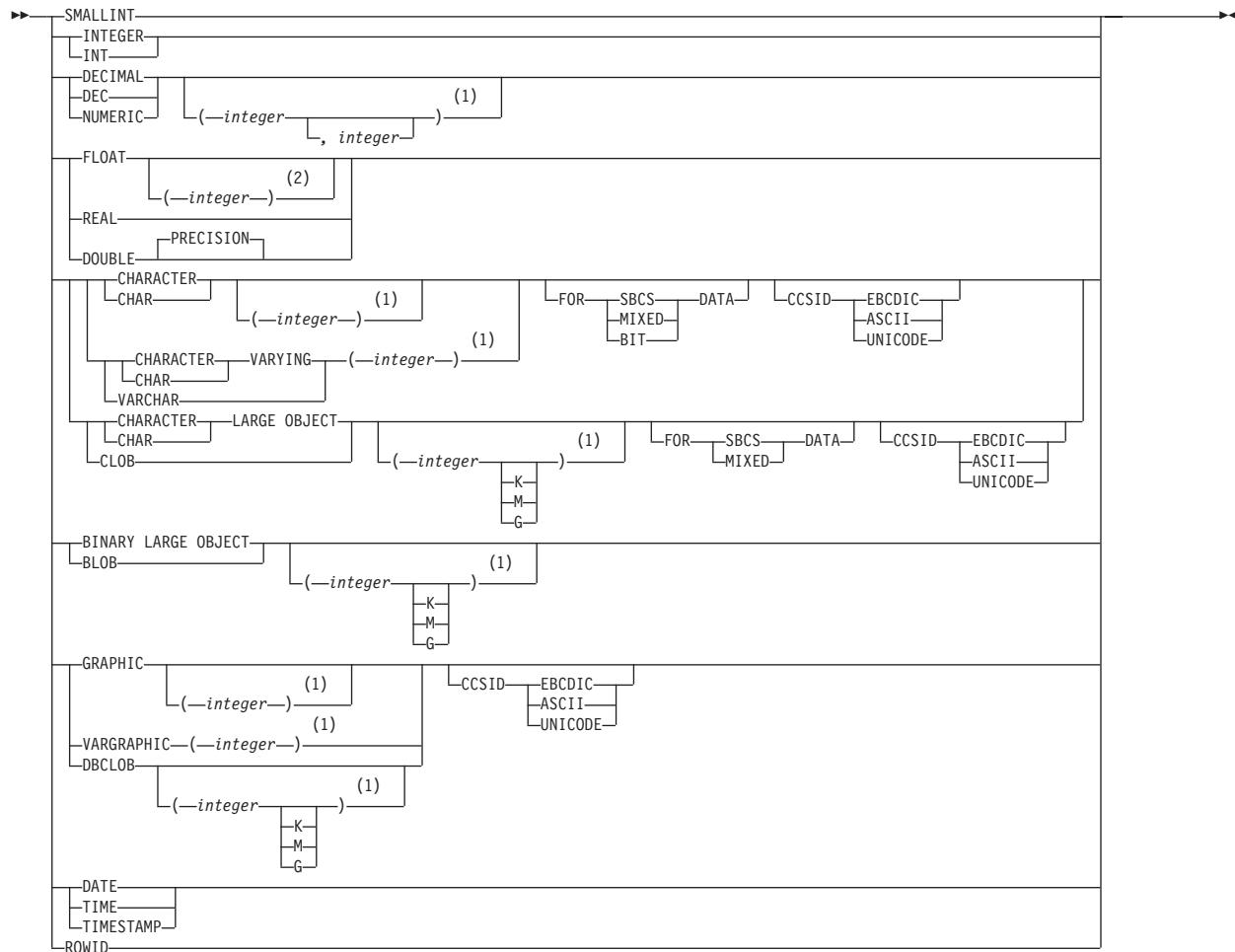


## parameter type:



### **data type:**



**built-in data type:****Notes:**

- 1 The values that are specified for length, precision, or scale attributes must match the values that were specified when the function was created. Coding specific values is optional. Empty parentheses, (), can be used instead to indicate that DB2 ignores the attributes when determining whether data types match.
- 2 The value that is specified does not have to match the value that was specified when the function was created because matching is based on data type (REAL or DOUBLE).  $1 \leq integer \leq 21$  indicates REAL and  $22 \leq integer \leq 53$  indicates DOUBLE. Coding a specific value is optional. Empty parentheses cannot be used.

**Description****EXECUTE**

Revokes the privilege to run the identified user-defined function, cast function that was generated for a distinct type, or stored procedure.

## REVOKE (function or procedure privileges)

### FUNCTION or SPECIFIC FUNCTION

The function that is identified must exist at the current server, and it must be a function that was defined with the CREATE FUNCTION statement or a cast function that was generated by a CREATE DISTINCT TYPE statement.

If the function was defined with a table parameter (the LIKE TABLE was specified in the CREATE FUNCTION statement to indicate that one of the input parameters is a transition table), the function signature cannot be used to identify the function. Instead, identify the function with its function name, if unique, or with its specific name.

#### FUNCTION *function-name*

Identifies the function by its name. You can identify a function by its name only if there is exactly one function with *function-name* in the schema. If you do not explicitly qualify the function name with a schema name, the function name is implicitly qualified with a schema name according to the following rules:

- If the statement is embedded in a program, the schema name is the authorization ID in the QUALIFIER option when the plan or package was created or last rebound. If QUALIFIER was not used, the schema name is the owner of the plan or package.
- If the statement is dynamically prepared, the schema name is the SQL authorization ID in the CURRENT SQLID special register.

An \* can be specified for a qualified or unqualified *function-name*. An \* (or *schema-name.\**) indicates that the privilege is revoked for all the functions in the schema. You (or the indicated grantors) must have granted the privilege on FUNCTION \* to all identified users (including PUBLIC if specified). Privileges granted on specific functions are not affected.

#### FUNCTION *function-name* (*parameter-type*,...)

Identifies the function by its function signature, which uniquely identifies the function. If *function-name()* is specified, the function that is identified must have zero parameters.

#### *function-name*

Specifies the name of the function. If you do not explicitly qualify the function name with a schema name, the function name is implicitly qualified with a schema name as described in the preceding description for FUNCTION *function-name*.

#### (*parameter-type*,...)

Identifies the number of input parameters of the function and their data types.

The data type of each parameter must match the data type that was specified in the CREATE FUNCTION statement for the parameter in the corresponding position. The number of data types and the logical concatenation of the data types is used to uniquely identify the function.

For data types that have a length, precision, or scale attribute, you can specify a value or use a set of empty parentheses:

- Empty parentheses indicate that DB2 ignores the attribute when determining whether the data types match.  
FLOAT cannot be specified with empty parentheses because its parameter value indicates different data types (REAL or DOUBLE).
- If you use a specific value for a length, precision, or scale attribute, the value must exactly match the value that was specified (implicitly or explicitly) in the CREATE FUNCTION statement.

## REVOKE (function or procedure privileges)

The specific value for FLOAT(*n*) does not have exactly match the defined value of the source function because 1<=n<= 21 indicates REAL and 22<=n<=53 indicates DOUBLE. Matching is based on whether the data type is REAL or DOUBLE.

- If length, precision, or scale is not explicitly specified, and empty parentheses are not specified, the default length of the data type is implied. For example:

|                |                      |
|----------------|----------------------|
| <b>CHAR</b>    | CHAR(1)              |
| <b>GRAPHIC</b> | GRAPHIC(1)           |
| <b>DECIMAL</b> | DECIMAL(5,0)         |
| <b>FLOAT</b>   | DOUBLE (length of 8) |

The implicit length must exactly match the value that was specified (implicitly or explicitly) in the CREATE FUNCTION statement. For a complete list of the default lengths of data types, see “CREATE TABLE” on page 653.

For data types with a subtype or encoding scheme attribute, specifying the FOR DATA clause or CCSID clause is optional. Omission of either clause indicates that DB2 ignores the attribute when determining whether the data types match. If you specify either clause, it must match the value that was implicitly or explicitly specified in the CREATE FUNCTION statement.

### SPECIFIC FUNCTION *specific-name*

Identifies the function by its specific name.

### PROCEDURE *procedure-name*

Identifies a stored procedure that is defined at the current server. If you do not explicitly qualify the procedure name with a schema name, the procedure name is implicitly qualified with a schema name according to the following rules:

- If the statement is embedded in a program, the schema name is the authorization ID in the QUALIFIER option when the plan or package was created or last rebound. If QUALIFIER was not used, the schema name is the owner of the plan or package.
- If the statement is dynamically prepared, the schema name is the SQL authorization ID in the CURRENT SQLID special register.

An \* can be specified for a qualified or unqualified *procedure-name*. An \* (or *schema-name.\**) indicates that the privilege is revoked for all the procedures in the schema. You (or the indicated grantors) must have granted the privilege on PROCEDURE \* to all identified users (including PUBLIC if specified). Privileges granted on specific procedures are not affected.

### FROM

Refer to “REVOKE” on page 865 for a description of the FROM clause.

### BY

Refer to “REVOKE” on page 865 for a description of the BY clause.

### RESTRICT

Prevents the EXECUTE privilege from being revoked on a user-defined function or stored procedure if the revokee owns any of the following objects:

- A function that is sourced on the function
- A view that uses the function
- A trigger package that uses the function or stored procedure

## REVOKE (function or procedure privileges)

- A table that uses the function in a check constraint or user-defined default clause

## Examples

*Example 1:* Revoke the EXECUTE privilege on function CALC\_SALARY for user JONES. Assume that there is only one function in the schema with function CALC\_SALARY.

```
REVOKE EXECUTE ON FUNCTION CALC_SALARY FROM JONES;
```

*Example 2:* Revoke the EXECUTE privilege on procedure VACATION\_ACCR from all users at the current server.

```
REVOKE EXECUTE ON PROCEDURE VACATION_ACCR FROM PUBLIC;
```

*Example 3:* Revoke the privilege of the administrative assistant to grant EXECUTE privileges on function DEPT\_TOTAL to other users. The administrative assistant will still have the EXECUTE privilege on function DEPT\_TOTALS.

```
REVOKE EXECUTE ON FUNCTION DEPT_TOTALS
FROM ADMIN_A;
```

*Example 4:* Revoke the EXECUTE privilege on function NEW\_DEPT\_HIRES for HR (Human Resources). The function has two input parameters with data types of INTEGER and CHAR(10), respectively. Assume that the schema has more than one function that is named NEW\_DEPT\_HIRES.

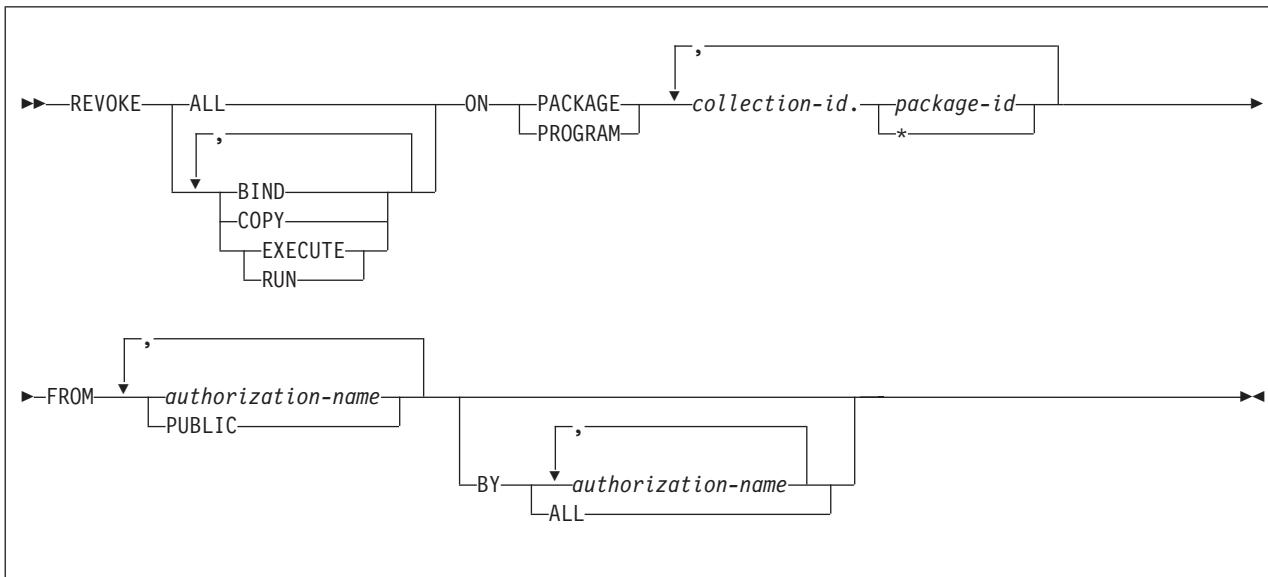
```
REVOKE EXECUTE ON FUNCTION NEW_DEPT_HIRES (INTEGER, CHAR(10))
FROM HR;
```

You can also code the CHAR(10) data type as CHAR().

## REVOKE (package privileges)

This form of the REVOKE statement revokes privileges on packages.

### Syntax



### Description

#### BIND

Revokes the privilege to use the BIND and REBIND subcommands for the designated packages. In addition, if the value of field BIND NEW PACKAGE on installation panel DSNTIPP is BIND, the additional BIND privilege of adding new versions of packages is revoked. (For details, see “Notes” on page 817 for “GRANT (package privileges)” on page 816.)

#### COPY

Revokes the privilege to use the COPY option of the BIND subcommand for the designated packages.

#### EXECUTE

Revokes the privilege to run application programs that use the designated packages and to specify the packages following PKLIST for the BIND PLAN and REBIND PLAN commands. RUN is an alternate name for the same privilege.

#### ALL

Revokes all package privileges for which you have authority for the packages named in the ON clause.

#### ON PACKAGE *collection-id.package-id...*

Identifies packages for which you are revoking privileges. The revoking of a package privilege applies to all versions of that package. For each package that you identify, you (or the indicated grantors) must have granted at least one of the specified privileges on that package to all identified users (including PUBLIC, if specified). An authorization ID with PACKADM authority over the collection or all collections, SYSADM, or SYSCTRL authority can specify all packages in the collection by using \* for *package-id*. The same package must not be specified more than once.

## **REVOKE (package privileges)**

The word PROGRAM can be used in place of PACKAGE.

### **FROM**

Refer to “REVOKE” on page 865 for a description of the FROM clause.

### **BY**

Refer to “REVOKE” on page 865 for a description of the BY clause.

## **Example**

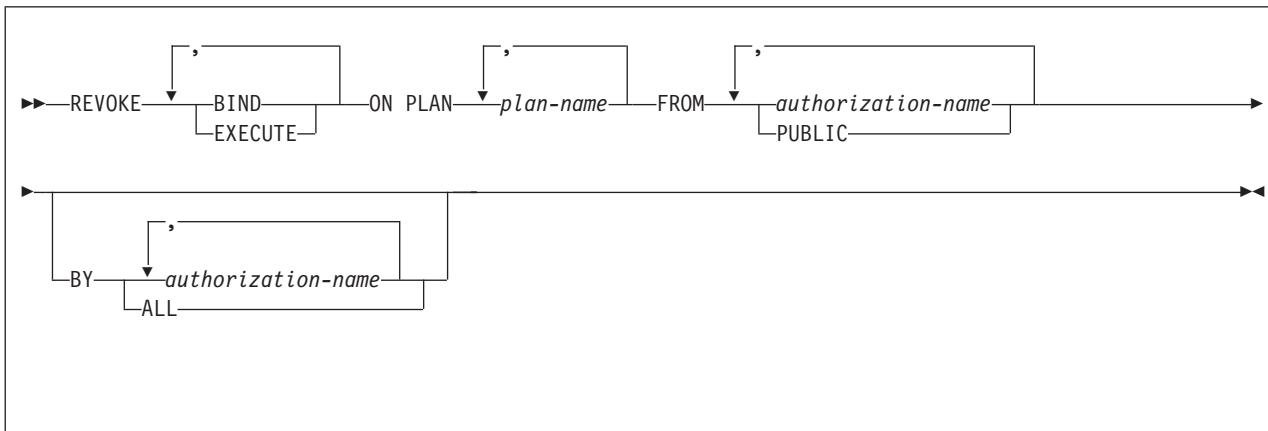
Revoke the privilege to copy all packages in collection DSN8CC61 from LEWIS.

```
REVOKE COPY ON PACKAGE DSN8CC61.* FROM LEWIS;
```

## REVOKE (plan privileges)

This form of the REVOKE statement revokes privileges on application plans.

### Syntax



### Description

#### BIND

Revokes the privilege to use the BIND, REBIND, and FREE subcommands for the identified plans.

#### EXECUTE

Revokes the privilege to run application programs that use the identified plans.

#### ON PLAN *plan-name*,...

Identifies application plans for which you are revoking privileges. For each plan that you identify, you (or the indicated grantors) must have granted at least one of the specified privileges on that plan to all identified users (including PUBLIC, if specified). The same plan must not be specified more than once.

#### FROM

Refer to “REVOKE” on page 865 for a description of the FROM clause.

#### BY

Refer to “REVOKE” on page 865 for a description of the BY clause.

### Examples

*Example 1:* Revoke authority to bind plan DSN8IP71 from user JONES.

```
REVOKE BIND ON PLAN DSN8IP71 FROM JONES;
```

*Example 2:* Revoke authority previously granted to all users at the current server to bind and execute plan DSN8CP71. (Grants to specific users will not be affected.)

```
REVOKE BIND,EXECUTE ON PLAN DSN8CP71 FROM PUBLIC;
```

*Example 3:* Revoke authority to execute plan DSN8CP71 from users ADAMSON and BROWN.

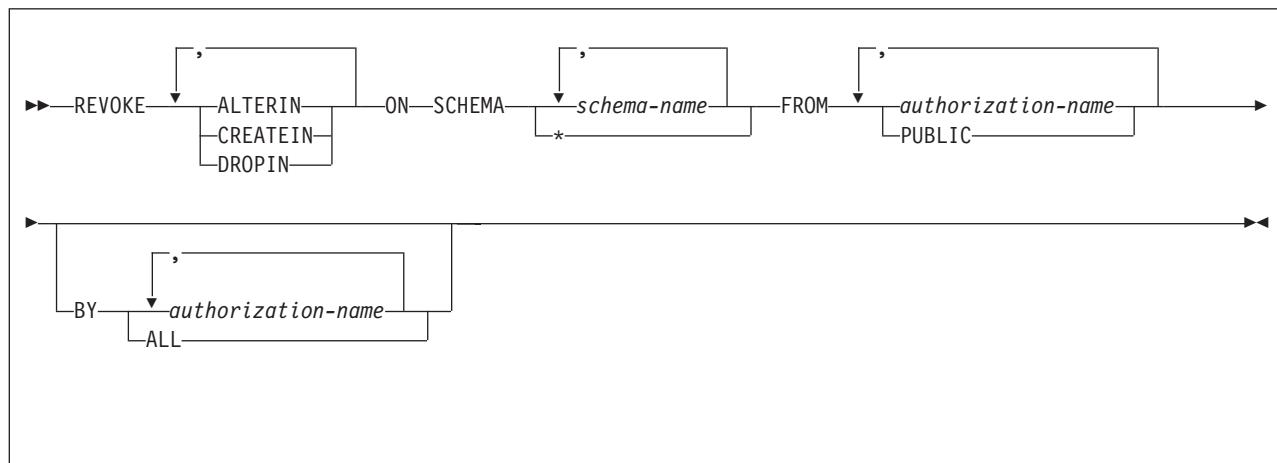
```
REVOKE EXECUTE ON PLAN DSN8CP71 FROM ADAMSON,BROWN;
```

## REVOKE (schema privileges)

## REVOKE (schema privileges)

This form of the REVOKE statement revokes privileges on schemas.

### Syntax



### Description

#### ALTERIN

Revokes the privilege to alter stored procedures and user-defined functions, or specify a comment for distinct types, cast functions that are generated for distinct types, stored procedures, triggers, and user-defined functions in the designated schemas.

#### CREATEIN

Revokes the privilege to create distinct types, stored procedures, triggers, and user-defined functions in the designated schemas.

#### DROPIN

Revokes the privilege to drop distinct types, stored procedures, triggers, and user-defined functions in the designated schemas.

#### SCHEMA *schema-name*

Identifies the schema on which the privilege is revoked.

#### SCHEMA \*

Indicates that the specified privilege on all schemas is revoked. You (or the indicated grantors) must have previously granted the specified privilege on SCHEMA \* to all identified users (including PUBLIC if specified). Privileges granted on specific schemas are not affected.

#### FROM

Refer to “REVOKE” on page 865 for a description of the FROM clause.

#### BY

Refer to “REVOKE” on page 865 for a description of the BY clause.

### Examples

*Example 1:* Revoke the CREATEIN privilege on schema T\_SCORES from user JONES.

```
REVOKE CREATEIN ON SCHEMA T_SCORES FROM JONES;
```

## REVOKE (schema privileges)

*Example 2:* Revoke the CREATEIN privilege on schema VAC from all users at the current server.

```
REVOKE CREATEIN ON SCHEMA VAC FROM PUBLIC;
```

*Example 3:* Revoke the ALTERIN privilege on schema DEPT from the administrative assistant.

```
REVOKE ALTERIN ON SCHEMA DEPT FROM ADMIN_A;
```

*Example 4:* Revoke the ALTERIN and DROPIN privileges on schemas NEW\_HIRE, PROMO, and RESIGN from HR (Human Resources).

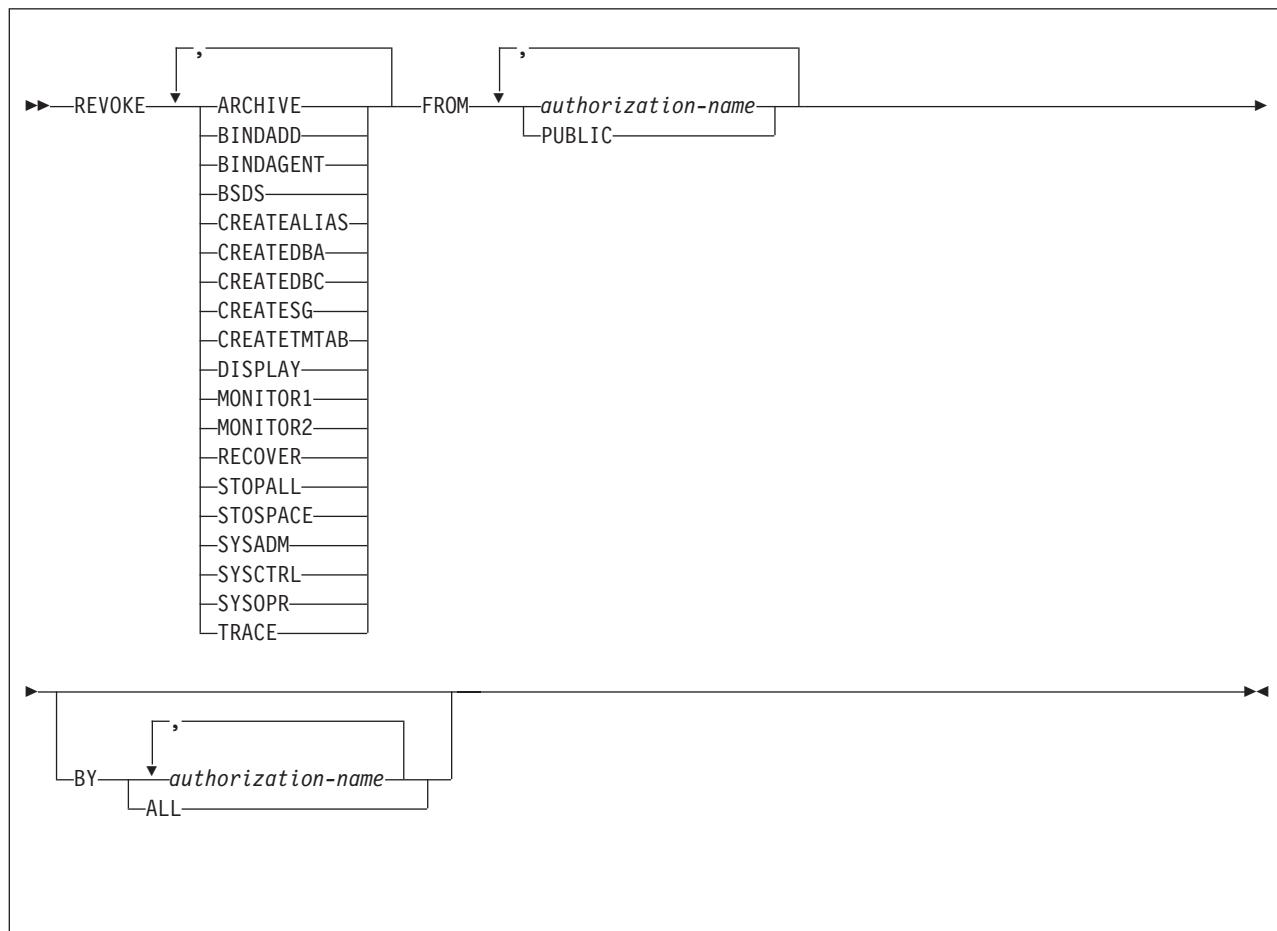
```
REVOKE ALTERIN, DROPIN ON SCHEMA NEW_HIRE, PROMO, RESIGN FROM HR;
```

## REVOKE (system privileges)

## REVOKE (system privileges)

This form of the REVOKE statement revokes system privileges.

### Syntax



### Description

#### ARCHIVE

Revokes the privilege to use the ARCHIVE LOG command.

#### BINDADD

Revokes the privilege to create plans and packages using the BIND subcommand with the ADD option.

#### BINDAGENT

Revokes the privilege to issue the BIND, FREE PACKAGE, or REBIND subcommands for plans and packages and the DROP PACKAGE statement on behalf of the grantor. The privilege also allows the holder to copy and replace plans and packages on behalf of the grantor.

A revoke of this privilege does not cascade.

#### BSDS

Revokes the privilege to issue the RECOVER BSDS command.

#### CREATEALIAS

Revokes the privilege to use the CREATE ALIAS statement.

**CREATEDBA**

Revokes the privilege to issue the CREATE DATABASE statement and acquire DBADM authority over those databases.

**CREATEDBC**

Revokes the privilege to issue the CREATE DATABASE statement and acquire DBCTRL authority over those databases.

**CREATESG**

Revokes the privilege to create new storage groups.

**CREATETMTAB**

Revokes the privilege to use the CREATE GLOBAL TEMPORARY TABLE statement.

**DISPLAY**

Revokes the privilege to use the following commands:

- The DISPLAY ARCHIVE command for archive log information
- The DISPLAY BUFFERPOOL command for the status of buffer pools
- The DISPLAY DATABASE command for the status of all databases
- The DISPLAY FUNCTION SPECIFIC command for statistics about accessed external user-defined functions
- The DISPLAY LOCATION command for statistics about threads with a distributed relationship
- The DISPLAY PROCEDURE command for statistics about accessed stored procedures
- The DISPLAY THREAD command for information on active threads with in DB2
- The DISPLAY TRACE command for a list of active traces

**MONITOR1**

Revokes the privilege to obtain IFC data classified as serviceability data, statistics, accounting, and other performance data that does not contain potentially secure data.

**MONITOR2**

Revokes the privilege to obtain IFC data classified as containing potentially sensitive data such as SQL statement text and audit data. (Having the MONITOR2 privilege also implies having MONITOR1 privileges, however, revoking the MONITOR2 privilege does not cause the revoke of an explicitly granted MONITOR1 privilege.)

**RECOVER**

Revokes the privilege to issue the RECOVER INDOUBT command.

**STOPALL**

Revokes the privilege to use the STOP DB2 command.

**STOSPACE**

Revokes the privilege to use the STOSPACE utility.

**SYSADM**

Revokes the system administrator authority.

**SYSCtrl**

Revokes the system control authority.

**SYSOPR**

Revokes the system operator authority.

## **REVOKE (system privileges)**

### **TRACE**

Revokes the privilege to use the MODIFY TRACE, START TRACE, and STOP TRACE commands.

### **FROM**

Refer to “REVOKE” on page 865 for a description of the FROM clause.

### **BY**

Refer to “REVOKE” on page 865 for a description of the BY clause.

## **Examples**

*Example 1:* Revoke DISPLAY privileges from user LUTZ.

```
REVOKE DISPLAY
 FROM LUTZ;
```

*Example 2:* Revoke BSDS and RECOVER privileges from users PARKER and SETRIGHT.

```
REVOKE BSDS,RECOVER
 FROM PARKER,SETRIGHT;
```

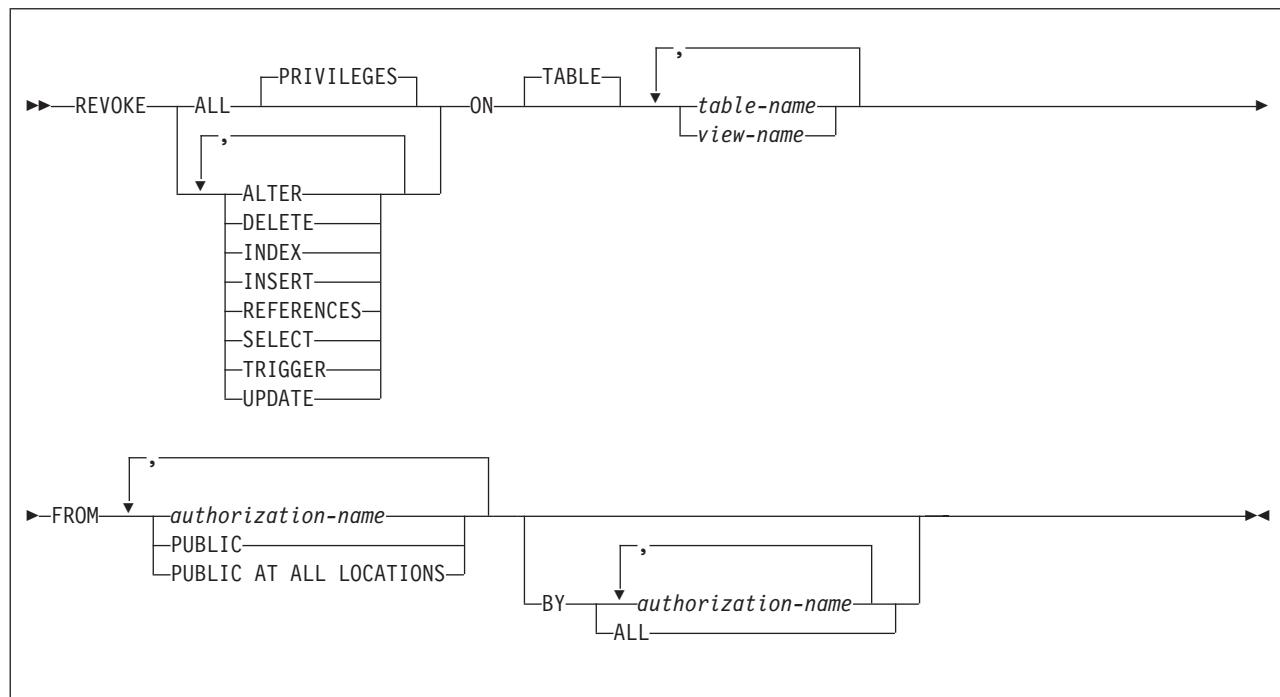
*Example 3:* Revoke TRACE privileges previously granted to all local users. (Grants to specific users will not be affected.)

```
REVOKE TRACE
 FROM PUBLIC;
```

## REVOKE (table or view privileges)

This form of the REVOKE statement revokes privileges on one or more tables or views.

### Syntax



### Description

#### ALL or ALL PRIVILEGES

If you specify ALL, the authorization ID of the statement must have granted at least one privilege on each identified table or view to each *authorization-name*. The privilege revoked from an authorization ID are those privileges on the table or view that the authorization ID of the statement granted to the authorization ID.

If you do not use ALL, you must use one or more of the keywords listed below. Each keyword revokes the privilege described, but only as it applies to the tables or views named in the ON clause.

#### ALTER

Revokes the privilege to use the ALTER statement.

#### DELETE

Revokes the privilege to use the DELETE statement.

#### INDEX

Revokes the privilege to use the CREATE INDEX statement.

#### INSERT

Revokes the privilege to use the INSERT statement.

#### REFERENCES

Revokes the privilege to define and drop referential constraints. Although you can use a list of column names with the GRANT statement, you cannot use a list of column names with REVOKE; the privilege is revoked for all columns.

## REVOKE (table or view privileges)

### SELECT

Revokes the privilege to use the SELECT statement. A view is dropped when the SELECT privilege that was used to create it is revoked, unless the owner of the view was directly granted the SELECT privilege from another source before the view was created.

### TRIGGER

Revokes the privilege to use the CREATE TRIGGER statement.

### UPDATE

Revokes the privilege to use the UPDATE statement. A list of column names can be used only with GRANT, not with REVOKE.

### ON or ON TABLE

Names one or more tables or views on which you are revoking the privileges. The list can consist of table names, view names, or a combination of the two. A table or view must not be identified more than once, and a declared temporary table must not be identified.

### FROM

Refer to “REVOKE” on page 865 for a description of the FROM clause.

### BY

If you omit BY, you must have granted each named privilege to each of the named users. More precisely, each privilege must have been granted to each user by a GRANT statement whose authorization ID is also the authorization ID of your REVOKE statement. Each of these grants is then revoked. (No single privilege need be granted on all tables and views.)

If BY is specified, each named grantor must satisfy the above requirement. In that case, the authorization ID of the statement need not satisfy the requirement unless it is one of the named grantors.

Refer to “REVOKE” on page 865 for a description of the BY clause.

## Notes

For a created temporary table or a view of a created temporary table, only ALL or ALL PRIVILEGES can be revoked. Specific table or view privileges cannot be revoked.

For a declared temporary table, no privileges can be revoked because none can be granted. When a declared temporary table is defined, PUBLIC implicitly receives all table privileges (without GRANT authority) for the table. These privileges are not recorded in the DB2 catalog.

**PUBLIC AT ALL LOCATIONS:** PUBLIC AT ALL LOCATIONS can continue to be specified as an alternative to PUBLIC as in prior releases. PUBLIC AT ALL LOCATIONS was introduced and was intended for use only with DB2 private protocol access.

## Examples

*Example 1:* Revoke SELECT privileges on table DSN8710.EMP from user PULASKI.

```
REVOKE SELECT ON TABLE DSN8710.EMP FROM PULASKI;
```

*Example 2:* Revoke update privileges on table DSN8710.EMP previously granted to all local DB2 users. (Grants to specific users are not affected.)

```
REVOKE UPDATE ON TABLE DSN8710.EMP FROM PUBLIC;
```

## REVOKE (table or view privileges)

*Example 3:* Revoke all privileges on table DSN8710.EMP from users KWAN and THOMPSON.

```
REVOKE ALL ON TABLE DSN8710.EMP FROM KWAN,THOMPSON;
```

*Example 4:* Revoke the grant of SELECT and UPDATE privileges on the table DSN8710.DEPT to every user in the network. Doing so does not affect users who obtained these privileges from some other grant.

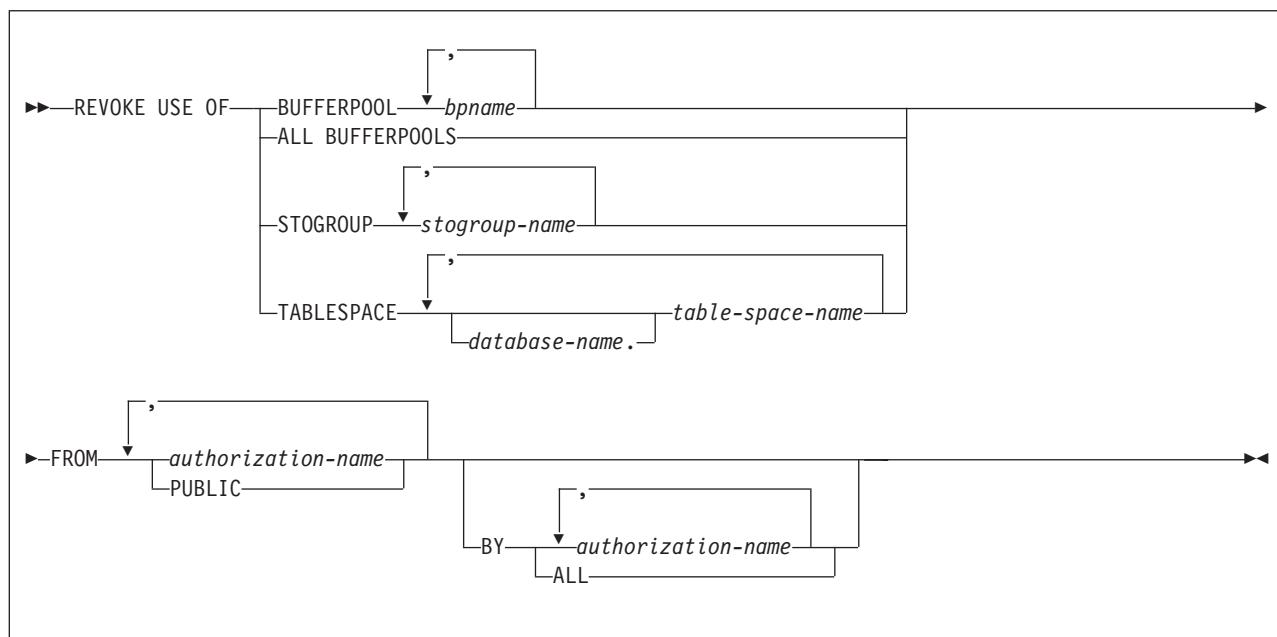
```
REVOKE SELECT, UPDATE ON TABLE DSN8710.DEPT
FROM PUBLIC AT ALL LOCATIONS;
```

## REVOKE (use privileges)

## REVOKE (use privileges)

This form of the REVOKE statement revokes authority to use particular buffer pools, storage groups, or table spaces.

## Syntax



## Description

### BUFFERPOOL *bpname*, ...

Revokes the privilege to refer to any of the identified buffer pools in a CREATE INDEX, CREATE TABLESPACE, ALTER INDEX, or ALTER TABLESPACE statement. See “Naming conventions” on page 34 for more details about *bpname*.

### ALL BUFFERPOOLS

Revokes the privilege to refer to any buffer pool in a CREATE INDEX, CREATE TABLESPACE, ALTER INDEX, or ALTER TABLESPACE statement.

### STOGROUP *stogroup-name*, ...

Revokes the privilege to refer to any of the identified storage groups in a CREATE INDEX, CREATE TABLESPACE, ALTER INDEX, or ALTER TABLESPACE statement.

### TABLESPACE *database-name*.*table-space-name*, ...

Revokes the privilege to refer to any of the specified table spaces in a CREATE TABLE statement. The default *database-name* is DSNDB04.

For table spaces in a TEMP database, which are for declared temporary tables, you cannot revoke the privilege from PUBLIC. When a table space is created in the TEMP database, PUBLIC implicitly receives the TABLESPACE privilege (without GRANT authority); this privilege is not recorded in the DB2 catalog, and it cannot be revoked.

### FROM

Refer to “REVOKE” on page 865 for a description of the FROM clause.

**BY**

Refer to “REVOKE” on page 865 for a description of the BY clause.

## Notes

You can revoke privileges for only one type of object with each statement. Thus you can revoke the use of several table spaces with one statement, but not the use of a table space and a storage group.

For each object you name, you (or the indicated grantors) must have granted the USE privilege on that object to all identified users (including PUBLIC, if specified). The same object must not be identified more than once.

Revoking the privilege USE OF ALL BUFFERPOOLS does not cascade to all other privileges that can be granted under that privilege. A user with the privilege USE OF ALL BUFFERPOOLS WITH GRANT OPTION can make two types of grants:

- GRANT USE OF ALL BUFFERPOOLS TO *userid*. This privilege is revoked when the original user's privilege is revoked.
- GRANT USE OF BUFFERPOOL BP*n* TO *userid*. This privilege is *not revoked* when the original user's privilege is revoked.

## Examples

*Example 1:* Revoke authority to use buffer pool BP2 from user MARINO.

```
REVOKE USE OF BUFFERPOOL BP2
 FROM MARINO;
```

*Example 2:* Revoke a grant of the USE privilege on the table space DSN8S71D in the database DSN8D71A. The grant is to PUBLIC, that is, to everyone at the local DB2 subsystem. (Grants to specific users are not affected.)

```
REVOKE USE OF TABLESPACE DSN8D71A.DSN8S71D
 FROM PUBLIC;
```

## ROLLBACK

## ROLLBACK

The ROLLBACK statement can be used to either:

- End a unit of recovery and back out all the relational database changes that were made by that unit of recovery. If relational databases are the only recoverable resources used by the application process, ROLLBACK also ends the unit of work.
- Back out only the changes made after a savepoint was set within the unit of recovery without ending the unit of recovery. Rolling back to a savepoint enables selected changes to be undone.

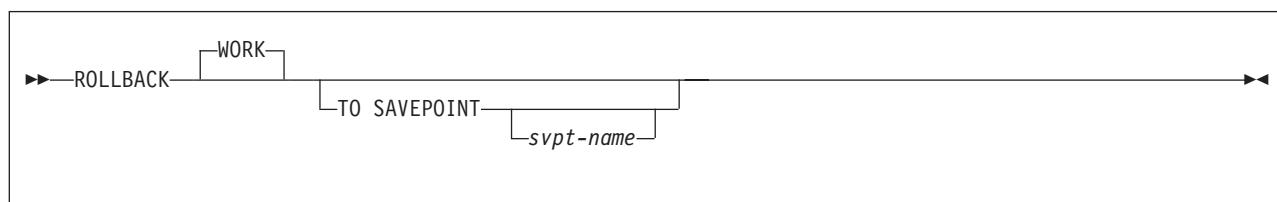
### Invocation

This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared. It can be used in the IMS environment, the CICS environment, or a global transaction only if the TO SAVEPOINT clause is specified.

### Authorization

None required.

### Syntax



### Description

When ROLLBACK is used without the SAVEPOINT clause, the unit of recovery in which the ROLLBACK statement is executed is ended and a new unit of recovery is effectively started. All changes made by ALTER, COMMENT ON, CREATE, DELETE, DROP, EXPLAIN, GRANT, INSERT, LABEL ON, RENAME, REVOKE, and UPDATE statements executed during the unit of recovery are backed out.

ROLLBACK without the TO SAVEPOINT clause also causes the following to occur:

- All locks implicitly acquired during the unit of recovery are released. See "LOCK TABLE" on page 840 for an explanation of the duration of explicitly acquired locks.
- All cursors are closed, all prepared statements are destroyed, and any cursors associated with the prepared statements are invalidated.
- All rows and all logical work files of every created temporary table of the application process are deleted. (All the rows of a declared temporary table are not implicitly deleted. As with base tables, any changes made to a declared temporary table during the unit of recovery are undone to restore the table to its state at the last commit.)
- All LOB locators, including those that are held, are freed.

#### TO SAVEPOINT

Specifies that the unit of recovery is not to be ended and that only a partial rollback (to a savepoint) is to be performed. If a savepoint name is not specified, rollback is to the last active savepoint. For example, if in a unit of

recovery, savepoints A, B, and C are set in that order and then C is released, ROLLBACK TO SAVEPOINT causes a rollback to savepoint B.

*svpt-name*

A savepoint-identifier that identifies the savepoint to which to roll back. If the named savepoint does not exist, an error occurs.

All database changes (including changes made to a declared temporary tables but excluding changes made to created temporary tables) that were made after the savepoint was set are backed out. Changes that are made to created temporary tables are not logged and are not backed out; a warning is issued instead. (A warning is also issued when a created temporary table is changed and there is an active savepoint.)

In addition, none of the following items are backed out:

- The opening or closing of cursors
- Changes in cursor positioning
- The acquisition and release of locks
- The caching of the rolled back statements

Any savepoints that are set after the one to which rollback is performed are released. The savepoint to which rollback is performed is not released.

ROLLBACK with or without the TO SAVEPOINT clause has no effect on connections.

## Notes

The following information applies only to rolling back all changes in the unit of recovery (the ROLLBACK statement without the TO SAVEPOINT clause):

- *Stored procedures.* The ROLLBACK statement cannot be used if the procedure is in the calling chain of a user-defined function or a trigger or if DB2 is not the commit coordinator.
- *IMS or CICS.* Using a ROLLBACK TO SAVEPOINT statement in an IMS or CICS environment only rolls back DB2 resources. Any other recoverable resources updated in the environment are not rolled back. To do a rollback operation for other recoverable resources in these environments, SQL programs must use the call prescribed by their transaction manager. The effect of these rollback operations on DB2 data is the same as that of the SQL ROLLBACK statement. A rollback operation in an IMS or CICS environment might handle the closing of cursors that were declared with the WITH hold option differently than the SQL ROLLBACK statement does. If an application requests a rollback operation from CICS or IMS, but no work has been performed in DB2 since the last commit point, the rollback request will not be broadcast to DB2. If the application had opened cursors using the WITH HOLD option in a previous unit of work, the cursors will not be closed, and any prepared statements associated with those cursors will not be destroyed.
- *Implicit rollback operations:* In all DB2 environments, the abend of a process is an implicit rollback operation.

## Examples

*Example 1:* Roll back all DB2 database changes made since the unit of recovery was started.

ROLLBACK WORK;

## ROLLBACK

*Example 2:* After a unit of recovery started, assume that three savepoints A, B, and C were set and that C was released:

```
...
SAVEPOINT A ON ROLLBACK RETAIN CURSORS;
...
SAVEPOINT B ON ROLLBACK RETAIN CURSORS;
...
SAVEPOINT C ON ROLLBACK RETAIN CURSORS;
...
RELEASE SAVEPOINT C;
...
```

Roll back all DB2 database changes only to savepoint A:

```
ROLLBACK WORK TO SAVEPOINT A;
```

If a savepoint name was not specified (that is, ROLLBACK WORK TO SAVEPOINT), the rollback would be to the last active savepoint that was set, which is B.

## SAVEPOINT

The SAVEPOINT statement sets a savepoint within a unit of recovery to identify a point in time within the unit of recovery to which relational database changes can be rolled back.

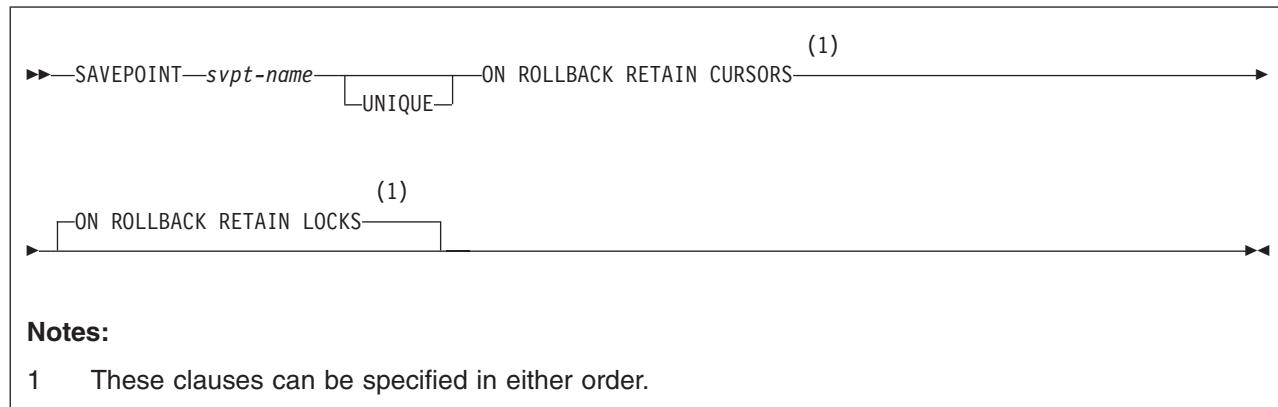
### Invocation

This statement can be imbedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared. This statement cannot be issued from a global transaction.

### Authorization

None required.

### Syntax



### Description

#### *svpt-name*

A savepoint identifier that names the savepoint. (A savepoint identifier is like an SQL identifier except that it has maximum length of 128 bytes.)

#### UNIQUE

Specifies that the application program cannot reuse the savepoint name within the unit of recovery. An error occurs if a savepoint with the same name as *svpt-name* already exists within the unit of recovery.

Omitting UNIQUE indicates that the application can reuse the savepoint name within the unit of recovery. If *svpt-name* identifies a savepoint that already exists within the unit of recovery and the savepoint was not created with the UNIQUE option, the existing savepoint is destroyed and a new savepoint is created. Destroying a savepoint to reuse its name for another savepoint is not the same as releasing the savepoint. Reusing a savepoint name destroys only one savepoint. Releasing a savepoint with the RELEASE SAVEPOINT statement releases the savepoint and all savepoints that have been subsequently set.

#### ON ROLLBACK RETAIN CURSORS

Specifies that any cursors that are opened after the savepoint is set are not tracked, and thus, are not closed upon rollback to the savepoint. Although these cursors remain open after rollback to the savepoint, they might not be usable.

## SAVEPOINT

For example, if rolling back to the savepoint causes the insertion of a row on which the cursor is positioned to be rolled back, using the cursor to update or delete the row results in an error.

### ON ROLLBACK RETAIN LOCKS

Specifies that any locks that are acquired after the savepoint is set are not tracked, and thus, are not released on rollback to the savepoint.

## Example

Assume that you want to set three savepoints at various points in a unit of recovery. Name the first savepoint A and allow the savepoint name to be reused. Name the second savepoint B and do not allow the name to be reused. Because you no longer need savepoint A when you are ready to set the third savepoint, reuse A as the name of the savepoint.

```
SAVEPOINT A ON ROLLBACK RETAIN CURSORS;
:
SAVEPOINT B UNIQUE ON ROLLBACK RETAIN CURSORS;
:
SAVEPOINT A ON ROLLBACK RETAIN CURSORS;
```

---

**SELECT**

For a description of the SELECT statement, see “select-statement” on page 369.

## SELECT INTO

### SELECT INTO

The SELECT INTO statement produces a result table that contains at most one row, and assigns the values in that row to host variables. If the table is empty, the statement assigns +100 to SQLCODE, '02000' to SQLSTATE, and does not assign values to the host variables. The tables or views identified in the statement can exist at the current server or at any DB2 subsystem with which the current server can establish a connection.

### Invocation

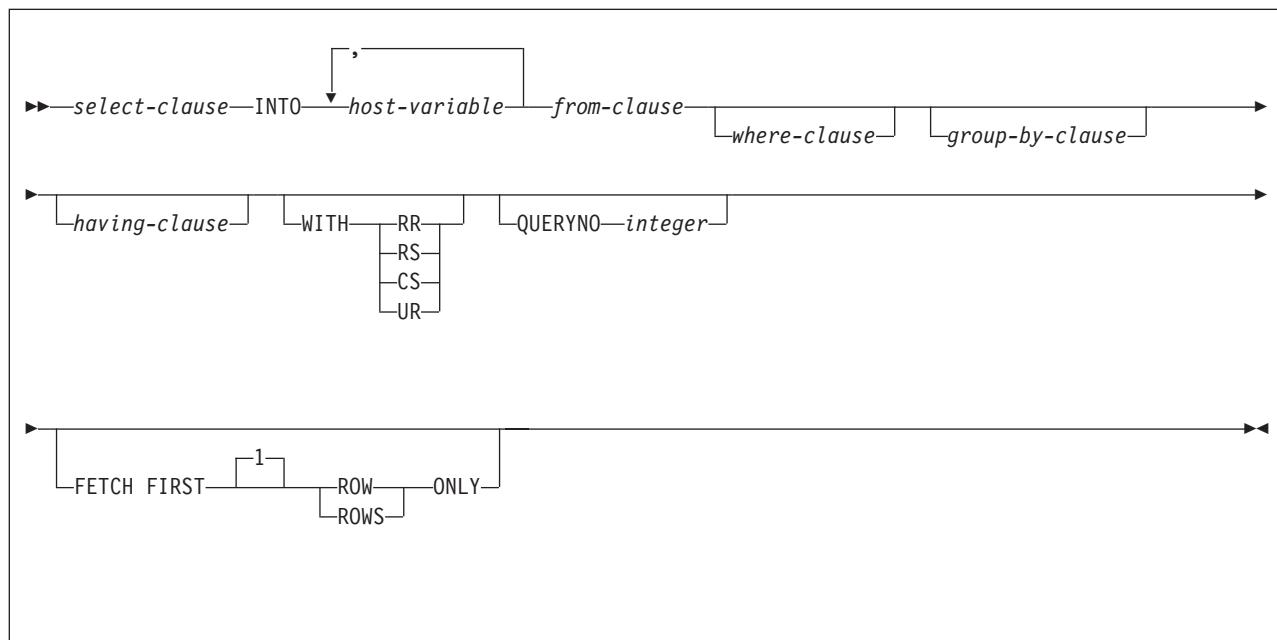
This statement can only be embedded in an application program. It is an executable statement that cannot be dynamically prepared.

### Authorization

The privileges that are held by the authorization ID of the owner of the plan or package must include at least one of the following for every table and view identified in the statement:

- The SELECT privilege on the table or view
- The EXECUTE privilege on any user-defined function
- Ownership of the table or view
- DBADM authority for the database (tables only)
- SYSADM authority
- SYSCTRL authority (catalog tables only)

### Syntax



### Description

The table is derived by evaluating the `from-clause`, `where-clause`, `group-by-clause`, `having-clause`, and `select-clause`, in this order. See Chapter 4, “Queries,” on page 347 for a description of these clauses.

#### **INTO host-variable,...**

Identifies one or more host structures or variables that must be declared in the

program in accordance with the rules for declaring host structures and variables. In the operational form of the INTO clause, a reference to a host structure is replaced by a reference to each of its host variables.

The first value in the result row is assigned to the first variable in the list, the second value to the second variable, and so on. If the number of host variables is less than the number of column values, the value W is assigned to the SQLWARN3 field of the SQLCA. (See “SQL communication area (SQLCA)” on page 979.)

The data type of a variable must be compatible with the value assigned to it. If the value is numeric, the variable must have the capacity to represent the integral part of the value. For a date or time value, the variable must be a character string variable of a minimum length as defined in Chapter 2, “Language elements,” on page 27. If the value is null, an indicator variable must be specified.

Each assignment to a variable is made according to the rules described in Chapter 2, “Language elements,” on page 27. Assignments are made in sequence through the list.

If an error occurs as the result of an arithmetic expression in the SELECT list of a SELECT INTO statement (division by zero or overflow) or a numeric conversion error occurs, the result is the null value. As in any other case of a null value, an indicator variable must be provided and the main variable is unchanged. In this case, however, the indicator variable is set to -2. Processing of the statement continues as if the error had not occurred. (However, this error causes a positive SQLCODE.) If you do not provide an indicator variable, a negative value is returned in the SQLCODE field of the SQLCA. Processing of the statement terminates when the error is encountered.

If an error occurs, no value is assigned to the host variable or to later variables, though any values that have already been assigned to variables remain assigned.

If an error occurs because the result table has more than one row, values may or may not be assigned to the host variables. If values are assigned to the host variables, the row that is the source of the values is undefined and not predictable.

#### WITH

Specifies the isolation level at which the statement is executed. (Isolation level does not apply to declared temporary tables because no locks are acquired.)

**RR** Repeatable read

**RS** Read stability

**CS** Cursor stability

**UR** Uncommitted read

WITH UR can be specified only if the result table is read-only.

The **default** isolation level of the statement depends on:

- The isolation of the package or plan that the statement is bound in
- Whether the result table is read-only

| If package isolation is: | And plan isolation is: | And the result table is: | Then the default isolation is: |
|--------------------------|------------------------|--------------------------|--------------------------------|
| RR                       | Any                    | Any                      | RR                             |
| RS                       | Any                    | Any                      | RS                             |
| CS                       | Any                    | Any                      | CS                             |

## SELECT INTO

| If package isolation is: | And plan isolation is: | And the result table is: | Then the default isolation is: |
|--------------------------|------------------------|--------------------------|--------------------------------|
| UR                       | Any                    | Read-only                | UR                             |
|                          |                        | Not read-only            | CS                             |
| Not specified            | Not specified          | Any                      | RR                             |
|                          | RR                     | Any                      | RR                             |
|                          | RS                     | Any                      | RS                             |
|                          | CS                     | Any                      | CS                             |
|                          | UR                     | Read-only                | UR                             |
|                          |                        | Not read-only            | CS                             |

### QUERYNO *integer*

Specifies the number to be used for this SQL statement in EXPLAIN output and trace records. The number is used for the QUERYNO columns of the plan tables for the rows that contain information about this SQL statement. This number is also used in the QUERYNO column of the SYSIBM.SYSSTMT and SYSIBM.SYSPACKSTMT catalog tables.

If the clause is omitted, the number associated with the SQL statement is the statement number assigned during precompilation. Thus, if the application program is changed and then precompiled, that statement number might change.

Using the QUERYNO clause to assign unique numbers to the SQL statements in a program is helpful:

- For simplifying the use of optimization hints for access path selection
- For correlating SQL statement text with EXPLAIN output in the plan table

For information on using optimization hints, such as enabling the system for optimization hints and setting valid hint values, and for information on accessing the plan table, see Part 5 (Volume 2) of *DB2 Administration Guide*.

### FETCH FIRST ROW ONLY *integer*

The FETCH FIRST ROW ONLY clause can be used in the SELECT INTO statement when the query can result in more than a single row. The clause indicates that only one row should be retrieved regardless of how many rows might be in the result table. When a number is explicitly specified, it must be 1.

Using the FETCH FIRST ROW ONLY clause to explicitly limit the result table to a single row provides a way for the SELECT INTO statement to be used with a query that returns more than a single row. Using the clause helps you to avoid using a cursor when you know that you want to retrieve only one row. If the FETCH FIRST ROW ONLY clause is not specified and the result table contains more than a single row, an error occurs.

## Examples

*Example 1:* Put the maximum salary in DSN8710.EMP into the host variable MAXSALRY.

```
EXEC SQL SELECT MAX(SALARY)
 INTO :MAXSALRY
 FROM DSN8710.EMP;
```

*Example 2:* Put the row for employee 528671, from DSN8710.EMP, into the host structure EMPREC.

## **SELECT INTO**

```
EXEC SQL SELECT * INTO :EMPREC
 FROM DSN8710.EMP
 WHERE EMPNO = '528671'
END-EXEC.
```

## SET CONNECTION

## SET CONNECTION

The SET CONNECTION statement establishes the database server of the process by identifying one of its existing connections.

### Invocation

This statement can only be embedded in an application program, except in REXX programs. It is an executable statement that cannot be dynamically prepared.

### Authorization

None required.

### Syntax

```
►►SET CONNECTION [location-name]
 [host-variable]►►
```

### Description

*location-name* or *host-variable*

Identifies the SQL connection by the specified location name or the location name contained in the host variable. If a host variable is specified:

- It must be a character string variable with a length attribute that is not greater than 16. (A C NUL-terminated character string can be up to 17 bytes.)
- It must not be followed by an indicator variable.
- The location name must be left-justified within the host variable and must conform to the rules for forming an ordinary location identifier.
- If the length of the location name is less than the length of the host variable, it must be padded on the right with blanks.

Let S denote the specified location name or the location name contained in the host variable. S must identify an existing SQL connection of the application process. If S identifies the current SQL connection, the state of S and all other connections of the application process are unchanged. The following rules apply when S identifies a dormant SQL connection.

If the SET CONNECTION statement is successful:

- SQL connection S is placed in the current state.
- S is placed in the CURRENT SERVER special register.
- Information about server S is placed in the SQLERRP field of the SQLCA. If the server is an IBM product, the information has the form *pppvrrm*, where:
  - *ppp* is:
    - ARI for DB2 Server for VSE & VM
    - DSN for DB2 for OS/390 and z/OS
    - QSQ for OS/400
    - SQL for all other DB2 products
  - *vv* is a two-digit version identifier such as '07'.
  - *rr* is a two-digit release identifier such as '01'.
  - *m* is a one-digit modification level such as '0'.

For example, if the server is Version 7 of DB2 for OS/390 and z/OS with the latest maintenance, the value of SQLERRP is 'DSN07011'.

- Any previously current SQL connection is placed in the dormant state.

If the SET CONNECTION statement is unsuccessful, the connection state of the application process and the states of its SQL connections are unchanged.

## Notes

The use of CONNECT (Type 1) statements does not prevent the use of SET CONNECTION, but the statement either fails or does nothing because dormant SQL connections do not exist. The SQLRULES(DB2) bind option does not prevent the use of SET CONNECTION, but the statement is unnecessary because CONNECT (Type 2) statements can be used instead. Use the SET CONNECTION statement to conform to the SQL standard.

When an SQL connection is used, made dormant, and then restored to the current state in the same unit of work, the status of locks, cursors, and prepared statements for that SQL connection reflects its last use by the application process.

If the SET CONNECTION statement contains host variables, the contents of the host variables are assumed to be in the encoding scheme that was specified in the ENCODING parameter when the package or plan that contains the statement was bound.

## Example

Execute SQL statements at TOROLAB1, execute SQL statements at TOROLAB2, and then execute more SQL statements at TOROLAB1.

```
EXEC SQL CONNECT TO TOROLAB1;
 (execute statements referencing objects at TOROLAB1)
EXEC SQL CONNECT TO TOROLAB2;
 (execute statements referencing objects at TOROLAB2)
EXEC SQL SET CONNECTION TOROLAB1;
 (execute statements referencing objects at TOROLAB1)
```

The first CONNECT statement creates the TOROLAB1 connection, the second CONNECT statement places it in the dormant state, and the SET CONNECTION statement returns it to the current state.

## SET CURRENT APPLICATION ENCODING SCHEME

### SET CURRENT APPLICATION ENCODING SCHEME

The SET CURRENT APPLICATION ENCODING SCHEME statement assigns a value to the CURRENT APPLICATION ENCODING SCHEME special register. This special register allows users to control which encoding scheme will be used for dynamic SQL statements after the SET statement has been executed.

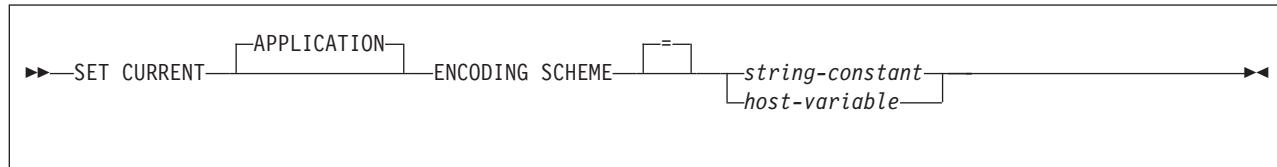
#### Invocation

This statement can be embedded only in an application program. It is an executable statement that cannot be dynamically prepared.

#### Authorization

None required.

#### Syntax



#### Description

##### *string-constant*

A character string constant that represents a valid encoding scheme (ASCII, EBCDIC, UNICODE, or a character representation of a number between 1 and 65533).

##### *host variable*

A variable with a data type of CHAR or VARCHAR. The value of *host-variable* must not be null and must represent a valid encoding scheme or a character representation of a number between 1 and 65533). An associated indicator variable must not be provided.

The value must:

- Be left justified within the host variable
- Be padded on the right with blanks if its length is less than that of the host variable

#### Example

*Examples:* The following examples set the CURRENT APPLICATION ENCODING SCHEME special register to 'EBCDIC' (in the second example, Host variable HV1 = 'EBCDIC').

```
EXEC SQL SET CURRENT APPLICATION ENCODING SCHEME = 'EBCDIC';
EXEC SQL SET CURRENT ENCODING SCHEME = :HV1;
```

## SET CURRENT DEGREE

The SET CURRENT DEGREE statement assigns a value to the CURRENT DEGREE special register.

### Invocation

This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared.

### Authorization

None required.

### Syntax

```
►►—SET CURRENT DEGREE = [string-constant]
 [host-variable]►►
```

### Description

The value of CURRENT DEGREE is replaced by the value of the string constant or host variable. The value must be a character string that is not longer than 3 bytes and the value must be 'ANY', '1', or '1  '.

### Notes

If the value of CURRENT DEGREE is '1' when a query is dynamically prepared, the execution of that query will not use parallel operations. If the value of CURRENT DEGREE is 'ANY' when a query is dynamically prepared, the execution of that query can involve parallel operations.

The initial value of CURRENT DEGREE is determined by the value of field CURRENT DEGREE on installation panel DSNTIP4. The default for the initial value is 1 unless your installation has changed it to be ANY by modifying the value in that field.

For distributed applications, the default value at the server is used unless the requesting application issues the SQL statement SET CURRENT DEGREE. For requests using DRDA, the SET CURRENT DEGREE statement must be within the scope of the CONNECT statement.

The value specified in the SET CURRENT DEGREE statement remains in effect until it is changed by the execution of another SET CURRENT DEGREE statement or until deallocation of the application process. For applications that connect to DB2 using the call attachment facility, the value of register CURRENT DEGREE can be requested to remain in effect for a longer duration. For more information, see the description of the call attachment facility CONNECT statement in Part 6 of *DB2 Application Programming and SQL Guide*.

### Examples

*Example 1:* The following statement inhibits parallel operations:

```
SET CURRENT DEGREE = '1';
```

## **SET CURRENT DEGREE**

*Example 2:* The following statement allows parallel operations:

```
SET CURRENT DEGREE = 'ANY';
```

## SET CURRENT LOCALE LC\_CTYPE

The SET CURRENT LOCALE LC\_CTYPE statement assigns a value to the CURRENT LOCALE LC\_CTYPE special register. The special register allows control over the LC\_CTYPE locale for statements that use a built-in function that refers to a locale, such as LCASE, UCASE, and TRANSLATE (with a single argument).

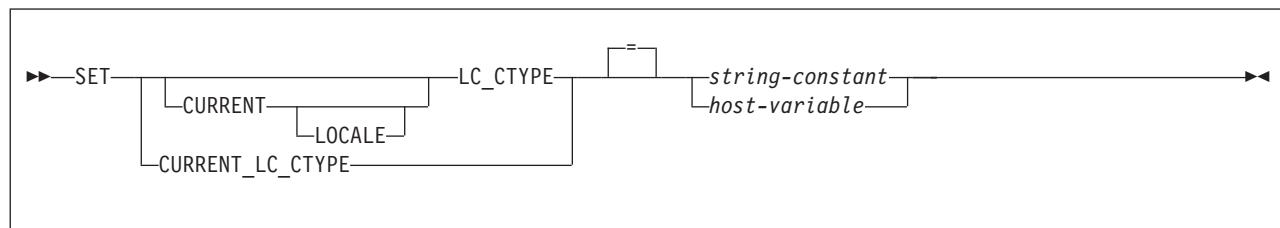
### Invocation

This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared.

### Authorization

None required.

### Syntax



### Description

The value of CURRENT LOCALE LC\_CTYPE is replaced by the string constant or host variable specified. The value must be a CHAR or VARCHAR character string that is no longer than 50 bytes.

If a host variable is specified, its value must not be null. If it has an associated indicator, the value of the indicator value must not indicate a null value. The locale must:

- Be left justified within the host variable
- Be padded on the right with blanks if its length is less than that of the host variable

The value of CURRENT LOCALE LC\_CTYPE is replaced by the value specified. The value must not be longer than 50 bytes and must be a valid locale.

#### *string-constant*

A character string constant that must not be longer than 50 bytes and must represent a valid locale.

#### *host-variable*

A variable with a data type of CHAR or VARCHAR and a length that is not longer than 50 bytes. The value of *host-variable* must not be null and must represent a valid locale. If the host variable has an associated indicator variable, the value of the indicator variable must not indicate a null value.

The locale must:

- Be left justified within the host variable
- Be padded on the right with blanks if its length is less than that of the host variable

## SET CURRENT LOCALE LC\_CTYPE

A locale can be specified in uppercase characters, lowercase characters, or a combination of the two. For information on locales and their naming conventions, see *OS/390 C/C++ Programming Guide*. Some examples of locales include:

Fr\_BE  
Fr\_FR@EURO  
En\_US  
Ja\_JP

## Examples

*Example 1:* Set the CURRENT LOCALE LC\_CTYPE special register to the locale 'En\_US'.

```
EXEC SQL SET CURRENT LOCALE LC_CTYPE = 'En_US';
```

*Example 2:* Set the CURRENT LOCALE LC\_CTYPE special register to the value of host variable HV1, which contains 'Fr\_FR@EURO'.

```
EXEC SQL SET CURRENT LOCALE LC_CTYPE = :HV1;
```

## SET CURRENT OPTIMIZATION HINT

The SET CURRENT OPTIMIZATION HINT statement assigns a value to the CURRENT OPTIMIZATION HINT special register.

### Invocation

This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared.

### Authorization

None required.

### Syntax

```
►►—SET CURRENT OPTIMIZATION HINT =—string-constant
 |
 host-variable—►►
```

### Description

The value of special register CURRENT OPTIMIZATION HINT is replaced by the value of the string constant or host variable. The value must be a character string that is not longer than 8 bytes.

### Notes

The initial value of the CURRENT OPTIMIZATION HINT special register is set to the value that was used for the OPTHINT bind option. The OPTHINT bind option specifies whether optimization hints are used in determining the access path of static statements and identifies which user-defined hint (rows in the authid.PLAN\_TABLE) is used. Therefore, if the SET CURRENT OPTIMIZATION HINT statement is not executed to change the value of the special register, DB2 uses the same optimization hint for dynamic statements that it uses for static statements. The default of OPTHINT for BIND PLAN and BIND PACKAGE is all blanks. All blanks indicate that DB2 uses normal optimization techniques and ignores optimization hints. DB2 does not use optimization hints for dynamic SQL if dynamic statement caching is enabled.

#  
#

### Example

Assume that delimited identifier 'NOHYB' identifies a user-defined optimization hint in authid.PLAN\_TABLE. Set the CURRENT OPTIMIZATION HINT special register so that DB2 uses this optimization hint to generate the access path for dynamic statements.

```
SET CURRENT OPTIMIZATION HINT = 'NOHYB'
```

#  
#  
#

If you set the register this way, DB2 validates and considers information in the rows in authid.PLAN\_TABLE where the value in the OPTHINT column matches 'NOHYB' for dynamic SQL statements.

## SET CURRENT PACKAGESET

### SET CURRENT PACKAGESET

The SET CURRENT PACKAGESET statement assigns a value to the CURRENT PACKAGESET special register.

#### Invocation

This statement can only be embedded in an application program. It is an executable statement that cannot be dynamically prepared.

#### Authorization

None required.

#### Syntax

```
►►SET CURRENT PACKAGESET = [USER
 [string-constant]
 [host-variable]]►►
```

#### Description

The value of CURRENT PACKAGESET is replaced by the value of the USER special register, *string-constant*, or *host-variable*. The value specified by *string-constant* or *host-variable* must be a character string that is not longer than 18 bytes. If the length of the replacement is less than 18 bytes, it is padded on the right with blanks so that its length is 18 bytes.

#### Notes

**Selection of plan elements:** A *plan element* is a DBRM that has been bound into the plan or a package that is implicitly or explicitly identified in the package list of the plan. Plan elements contain the control structures used to execute certain SQL statements.

Since a plan can have many elements, one of the first steps involved in the execution of an SQL statement that requires a control structure is the selection of the plan element that contains its control structure. The information used by DB2 to select plan elements includes the value of CURRENT PACKAGESET.

SET CURRENT PACKAGESET is used to specify the collection ID of a package that exists at the current server. SET CURRENT PACKAGESET is optional and should not be used without an understanding of the following rules for selecting a plan element.

If the CURRENT PACKAGESET special register is blank, DB2 searches for a DBRM or a package in one of these sequences:

**At the local location** (if CURRENT SERVER is blank or explicitly names that location), the order is:

1. All DBRMs bound directly to the plan
2. All packages that have already been allocated for the application process

3. All unallocated packages explicitly named in, and all collections completely included in, the package list of the plan. The order of search is the order those packages are named in the package list.

**At a remote location**, the order is:

1. All packages that have already been allocated for the application process at that location
2. All unallocated packages explicitly named in, and all collections completely included in, the package list of the plan, whose locations match the value of CURRENT SERVER. The order of search is the order those packages are named in the package list.

If the special register CURRENT PACKAGESET is set, DB2 skips the check for programs that are part of the plan and uses the value of CURRENT PACKAGESET as the collection. For example, if CURRENT PACKAGESET contains COL5, then DB2 uses COL5.PROG1.timestamp for the search. For additional information, see Part 4 of *DB2 Application Programming and SQL Guide*.

SET CURRENT PACKAGESET is executed by the requester and is therefore classified as a local SET statement in DRDA.

**CURRENT PACKAGESET special register with stored procedures and user-defined functions:** The initial value of the CURRENT PACKAGESET special register in a stored procedure or user-defined function is the value of the COLLID parameter with which the stored procedure or user-defined function was defined. If the routine was defined without a value for the COLLID parameter, the value of the special register is inherited from the calling program. A stored procedure or user-defined function can use the SET CURRENT PACKAGESET statement to change the value of the special register. This allows the routine to select the version of the DB2 package that is used to process the SQL statements in a called routine that is not defined with a COLLID value.

When control returns from the stored procedure to the calling program, the special register CURRENT PACKAGESET is restored to the value it contained before the stored procedure was called.

## Examples

*Example 1:* Limit the plan element selection to packages in the PERSONNEL collection at the current server.

```
EXEC SQL SET CURRENT PACKAGESET = 'PERSONNEL';
```

*Example 2:* Eliminate collections as a factor in plan element selection.

```
EXEC SQL SET CURRENT PACKAGESET = '';
```

## SET CURRENT PRECISION

### SET CURRENT PRECISION

The SET CURRENT PRECISION statement assigns a value to the CURRENT PRECISION special register.

#### Invocation

This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared.

#### Authorization

None required.

#### Syntax

```
►--SET CURRENT PRECISION = string-constant
 ┌─────────┐
 │host-variable│
 └─────────┘
```

#### Description

# This statement replaces the value of the CURRENT PRECISION special register  
# with the value of the string constant or host variable. The value must be a character  
# string 5 bytes in length. The value must be 'DEC15,' 'DEC31,' or 'Dpp.s', where 'pp'  
# is either 15 or 31 and 's' is a number between 1 and 9. If the form 'Dpp.s' is used,  
# 'pp' represents the precision that will be used with the rules that are used for  
# DEC15 or DEC31, and 's' represents the minimum divide scale to use for division  
# operations. The separator used in the form 'Dpp.s' may be either the '.' or the ','  
# character, regardless of the setting of the default decimal point. An error occurs if  
# any other values are specified.

#### Example

Set the CURRENT PRECISION special register so that subsequent statements that are prepared use DEC15 rules for decimal arithmetic.

```
EXEC SQL SET CURRENT PRECISION = 'DEC15';
```

## SET CURRENT RULES

The SET CURRENT RULES statement assigns a value to the CURRENT RULES special register.

### Invocation

This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared.

### Authorization

None required.

### Syntax

```
►►—SET CURRENT RULES =—string-constant
 |
 host-variable—►►
```

### Description

This statement replaces the value of the CURRENT RULES special register with the value of the string constant or host variable. The value must be a character string that is 3 bytes in length, and the value must be 'DB2' or 'STD'. An error occurs if any other values are specified.

### Notes

For the effect of the values 'DB2' and 'STD' on the execution of certain SQL statements, see "CURRENT RULES" on page 90.

### Example

Set the SQL rules to be followed to DB2.

```
EXEC SQL SET CURRENT RULES = 'DB2';
```

## SET CURRENT SQLID

### SET CURRENT SQLID

The SET CURRENT SQLID statement assigns a value to the CURRENT SQLID special register.

#### Invocation

This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared. The value to which special register CURRENT SQLID is set is used as the SQL authorization ID and the implicit qualifier for dynamic SQL statements only if DYNAMICRULES run behavior is in effect. The CURRENT SQLID value is ignored for the other DYNAMICRULES behaviors.

#### Authorization

If any of the authorization IDs of the process has SYSADM authority, CURRENT SQLID can be set to any value. Otherwise, the specified value must be equal to one of the authorization IDs of the application process. This rule always applies, even when SET CURRENT SQLID is a static statement.

#### Syntax

```
►►SET CURRENT SQLID = [USER
[string-constant]
host-variable]►►
```

#### Description

The value of CURRENT SQLID is replaced by the value of USER, *string-constant*, or *host-variable*. The value specified by a *string-constant* or *host-variable* must be a character string that is not longer than 8 bytes. If the length of the value is less than 8, it is padded on the right with blanks so that it is a string of 8 bytes. Unless some authorization ID of the process has SYSADM authority, the value must be equal to one of the authorization IDs of the process.

#### Notes

The value of CURRENT SQLID is called the SQL authorization ID. The SQL authorization ID is:

- The authorization ID used for authorization checking on dynamically prepared CREATE, GRANT, and REVOKE SQL statements
- The owner of a table space, database, storage group, or synonym created by a dynamically issued CREATE statement
- The implicit qualifier of all table, view, alias, and index names specified in dynamic SQL statements

SET CURRENT SQLID does not change the primary authorization ID of the process.

If the SET CURRENT SQLID statement is executed in a stored procedure or user-defined function package that has a dynamic SQL behavior other than run behavior, the SET CURRENT SQLID statement does not affect the authorization ID that is used for dynamic SQL statements in the package. The dynamic SQL

behavior determines the authorization ID. For more information, see the discussion of DYNAMICRULES in Chapter 2 of *DB2 Command Reference*.

The initial value of the SQL authorization ID is established during connection or sign-on processing. The value specified in the SET CURRENT SQLID is the SQL authorization ID until one of the following events occurs:

- The SQL authorization ID is changed by the execution of a new SET CURRENT SQLID statement.
- A SIGNON or re-SIGNON request is received from a CICS transaction subtask or an IMS independent region.
- The DB2 connection is ended.

```


When the value of the PATH special register depends on the value of the
CURRENT SQLID special register, any changes to the CURRENT SQLID special
register are not reflected in the value of the PATH special register until a commit
operation is performed or a SET PATH statement is issued to change the SQL path
to use the new value of the CURRENT SQLID.
```

SET CURRENT SQLID is executed by the database server and is therefore classified as a non-local SET statement in DRDA.

## Examples

*Example 1:* Set the CURRENT SQLID to the primary authorization ID.

```
SET CURRENT SQLID=USER;
```

*Example 2:* Example 2: Assume that the value of CURRENT SQLID is 'Jane' and that the default value of the PATH special register was established using that value (that is, the value of PATH is "SYSIBM", "SYSFUN", "SYSPROC", "Jane"). Change the value of the CURRENT SQLID special register to 'John'.

```
SET CURRENT SQLID = 'JOHN';
```

To change the SQL path to use the updated CURRENT SQLID value of 'John', issue a SET PATH statement to change the value of the PATH special register. Alternatively, a commit would cause PATH to be re-initialized. Otherwise, the path remains "SYSIBM", "SYSFUN", "SYSPROC", "Jane", which might cause unqualified object names to resolve to 'Jane' when you want them to resolve to 'John'.

## SET host-variable assignment

### SET host-variable assignment

The SET host-variable assignment statement assigns values, either of expressions or NULL values, to host variables.

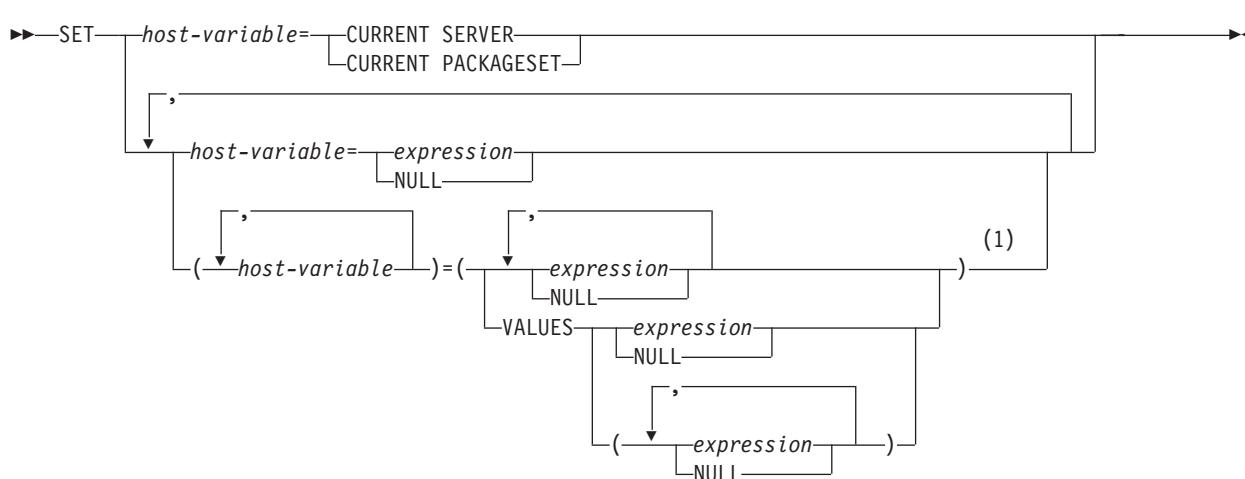
### Invocation

This statement can be embedded only in an application program. It is an executable statement that cannot be dynamically prepared.

### Authorization

The privileges that are held by the current authorization ID must include those required to execute any of the expressions.

### Syntax



#### Notes:

- 1 The number of *expressions* and NULL keywords must match the number of *host-variables*.

### Description

#### *host-variable*

Identifies one or more host variables or transition variables that are used to receive the corresponding *expression* or NULL value on the right side of the statement.

If the SET *host-variable* assignment statement is used in the triggered action of a CREATE TRIGGER statement, each *host-variable* must identify a transition variable. If the statement is used in any other context, each *host-variable* must identify a host variable.

The value to be assigned to each *host-variable* can be specified immediately following the item reference, for example, *host-variable = expression*, *host-variable=expression*. Or, sets of parentheses can be used to specify all the *host-variables* and then all the values, for example, *(host-variable, host-variable) = (expression, expression)*.

Each host variable must be defined in the program as described under the rules for declaring host variables. A parameter marker must not be specified in place of *host-variable*.

#### *expression*

Specifies the value to be assigned to the corresponding *host-variable*. The expression is any expression of the type described in “Expressions” on page 111, except it cannot contain a reference to *local* special registers (CURRENT SERVER or CURRENT PACKAGESET).

All expressions are evaluated before any result is assigned to a host variable. If an expression refers to a host variable that is used in the host variable list, the value of the variable in the expression is the value of the variable prior to any assignments.

Each assignment to a host variable is made according to the assignment rules described in “Assignment and comparison” on page 64. Assignments are made in sequence through the list. When the *host-variables* are enclosed within parentheses, for example,  $(\text{host-variable}, \text{host-variable}, \dots) = (\text{expression}, \text{expression}, \dots)$ , the first value is assigned to the first host variable in the list, the second value to the second host variable in the list, and so on.

#### **NULL**

Specifies the null value and can only be specified for host variables that have an associated indicator variable.

#### **VALUES**

Specifies the value to be assigned to the corresponding host variable. When more than one value is specified, the values must be enclosed in parentheses. Each value can be an expression or NULL, as described above. The following syntax is equivalent:

- $(\text{host-variable}, \text{host-variable}) = (\text{VALUES}(\text{expression}, \text{NULL}))$
- $(\text{host-variable}, \text{host-variable}) = (\text{expression}, \text{NULL})$

Local special registers can be referenced only in a VALUES host-variable statement that results in the assignment of a single host variable and not those that result in setting more than one value.

## Notes

The default encoding scheme for the data is the value in the bind option ENCODING, which is the option for application encoding. If this statement is used with functions such as LEN or SUBSTR that are operating on LOB locators, and the LOB data that is specified by the locator is in a different encoding scheme from the ENCODING bind option, LOB materialization and character conversion occur. To avoid LOB materialization and character conversion, select the LOB data from the SYSIBM.SYSDUMMYA, SYSIBM.SYSDUMMYE, or SYSIBM.SYSDUMMYU sample table.

Normally a locator can be used with a LOB and CLOBs are compatible with CHAR types, but it is not necessarily true that a locator can be used with a CHAR. For more information on using locators, see Part 2 of *DB2 Application Programming and SQL Guide*.

## Examples

*Example 1:* Set the host variable HVL to the value of the CURRENT PATH special register.

```
SET :HVL = CURRENT PATH;
```

## SET host-variable assignment

*Example 2:* Set the host variable PATH to the contents of the SQL PATH special register, the host variable XTIME to the local time at the current server, and the host variable MEM to the current member of the data sharing environment.

```
SET :SERVER = CURRENT PATH,
 :XTIME = CURRENT TIME,
 :MEM = CURRENT MEMBER;
```

*Example 3:* Set the host variable DETAILS to a portion of a LOB value, using a LOB expression with a LOB locator to refer the extracted portion of the value.

```
SET :DETAILS = SUBSTR(:LOCATOR,1,35);
```

If the LOB data that is specified by the LOB locator LOCATOR is in a different encoding scheme from the value of the ENCODING bind option, and you want to avoid LOB materialization and character conversion, use the following statement instead of the SET statement:

```
SELECT SUBSTR(:LOCATOR,1,35)
 INTO :DETAILS
 FROM SYSIBM.SYSDUMMYU;
```

*Example 4:* Set host variable HV1 to the results of external function CALC\_SALARY and host variable HV2 to the value of special register CURRENT PATH. Use an indicator value with HV1 in case CALC\_SALARY returns a null value.

```
SET (:HV1:IND1, :HV2) =
 (CALC_SALARY(:HV3, :HF4), CURRENT PATH);
```

## SET PATH

The SET PATH statement assigns a value to the CURRENT PATH special register.

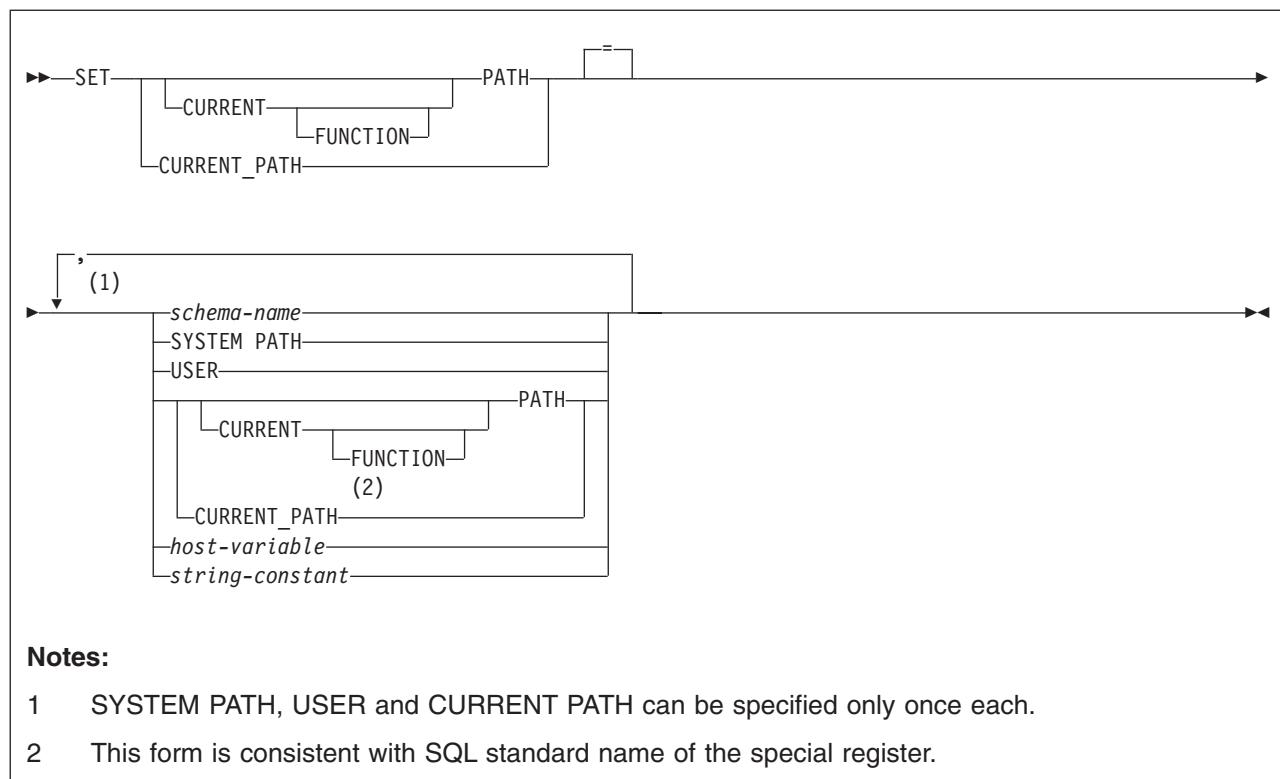
### Invocation

This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared.

### Authorization

None required.

### Syntax



### Description

The value of PATH is replaced by the values specified.

#### *schema-name*

Identifies a schema. DB2 does not verify that the schema exists. For example, a schema name that is misspelled is not detected, which could affect the way subsequent SQL operates.

#### **SYSTEM PATH**

Specifies the schema names "SYSIBM", "SYSFUN"<sup>39</sup>, "SYSPROC". SYSTEM PATH can be specified only once.

39. SYSFUN is a schema used for additional functions shipped on other servers in the DB2 product family. Although DB2 for OS/390 and z/OS does not use the SYSFUN schema, it can be useful to have SYSFUN in the path when doing distributed processing that involves a server that uses the SYSFUN schema.

## SET PATH

### USER

Specifies the value of the USER special register. USER can be specified only once.

### PATH

Specifies the value of the CURRENT PATH special register before the execution of this statement. PATH can be specified only once.

#### *host-variable*

A variable with a data type of CHAR or VARCHAR. The value of *host-variable* must not be null and must represent a valid schema name.

The schema name must:

- Be left justified within the host variable
- Be padded on the right with blanks if its length is less than that of the host variable

#### *string-constant*

A character string constant that represents a valid schema name.

If the schema name specified in *string-constant* will also be specified in other SQL statements and the schema name does not conform to the rules for ordinary identifiers, the schema name must be specified as a delimited identifier in the other SQL statements.

## Notes

**Restrictions on SET PATH:** These restrictions apply to the SET PATH statement:

- If the same schema name appears more than once in the path, the first occurrence of the name is used and a warning is issued.
- The length of the CURRENT PATH special register limits the number of schema names that can be specified. DB2 builds the string for the special register by taking each schema name specified and removing any trailing blanks from it, adding two delimiters around it, and adding one comma after each schema name except the last one. The length of the resulting string cannot exceed 254 bytes.

**Specifying SYSIBM and SYSPROC:** Schemas SYSIBM and SYSPROC do not need to be specified in the special register. If either of these schemas is not explicitly specified in the CURRENT PATH special register, the schema is implicitly assumed at the front of the SQL path; if both are not specified, they are assumed in the order of SYSIBM, SYSPROC (see “Schemas and the SQL path” on page 40 for an example). Only the schemas that are explicitly specified in the CURRENT PATH register are included in the 254 byte limit.

To avoid having SYSIBM or SYSPROC implicitly added to the front of the SQL path, explicitly specify them in the path when setting the value of the register. If you specify them at the end of the path, DB2 will check all the other schemas in the path first.

**Specifying USER versus "USER":** There is a difference between specifying USER with and without escape characters. To indicate that the value of the USER special register should be used in the SQL path, specify the keyword USER. If you specify USER as a delimited identifier instead (for example, "USER"), it is interpreted as a schema name of "USER". For example, assuming that the current value of the USER special register is SMITH, SET PATH = SYSIBM, SYSPROC, USER, "USER" results in SYSIBM, SYSPROC, SMITH, USER being used in the SQL path.

**Specifying a schema name in an SQL procedure:** Because a host variable (SQL variable) in an SQL procedure does not begin with a colon, DB2 uses the following rules to determine whether a value that is specified in a SET PATH=*name* statement is a variable or a *schema-name*:

- If *name* is the same as a parameter or SQL variable in the SQL procedure, DB2 uses *name* as a parameter or SQL variable and assigns the value in *name* to PATH.
- If *name* is not the same as a parameter or SQL variable in the SQL procedure, DB2 uses *name* as a *schema-name* and assigns the value *name* to PATH.

**The use of the path to resolve object names:** For information on when the SQL path is used to resolve unqualified data type, function, and procedure names and when the PATH provides the SQL path, see “Schemas and the SQL path” on page 40.

**DRDA classification:** The SET PATH statement is executed by the database server and, therefore, is classified as a non-local SET statement in DRDA.

## Examples

*Example 1:* Set the CURRENT PATH special register to the list of schemas: "SCHEMA1", "SCHEMA#2", "SYSIBM".

```
SET PATH = SCHEMA1,"SCHEMA#2", SYSIBM;
```

If the special register provides the SQL path, then SYSPROC, which was not explicitly specified in the special register, is implicitly assumed at the front of the SQL path, making the effective value of the path:

```
SYSPROC, SCHEMA1, SCHEMA#2, SYSIBM
```

*Example 2:* Add schema SMITH and SYSPROC to the value of the CURRENT PATH special register that was set in Example 1.

```
SET PATH = CURRENT PATH, SMITH, SYSPROC;
```

The value of the special register becomes:

```
"SCHEMA1", "SCHEMA#2", "SYSIBM", "SMITH", "SYSPROC"
```

## SET transition-variable assignment

### SET transition-variable assignment

The SET transition-variable assignment statement assigns values, either of expressions or NULL values, to transition variables.

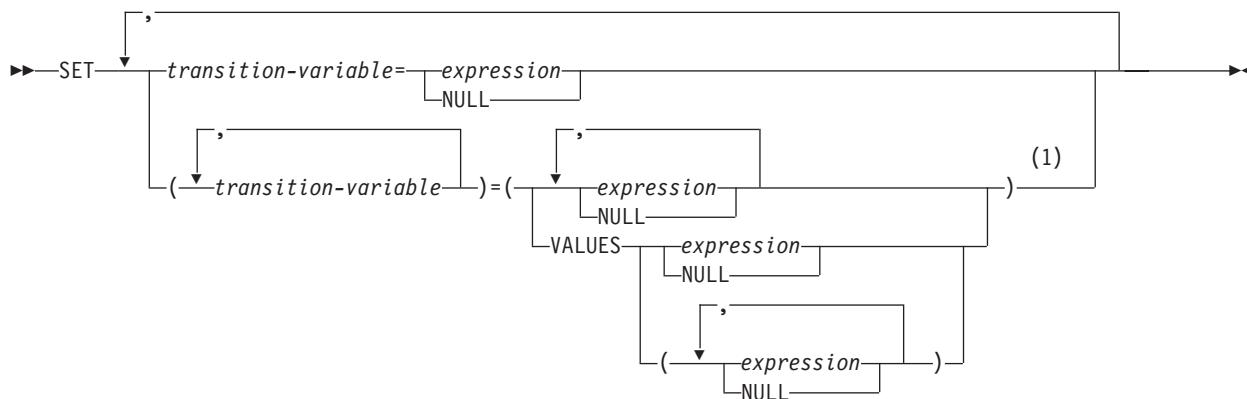
### Invocation

This statement can be used as a triggered SQL statement in the triggered action of a before trigger whose granularity is FOR EACH ROW.

### Authorization

The privileges that are held by the current authorization ID must include those required to execute any of the expressions or assignments to transition variables.

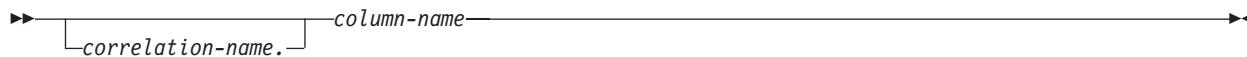
### Syntax



#### Notes:

- 1 The number of *expressions* and *NULL* keywords must match the number of *transition-variables*.

#### transition-variable:



### Description

#### *transition-variable*

Identifies a column in the set of affected rows for the trigger that is used to receive the corresponding *expression* or *NULL* value on the right side of the statement.

The value to be assigned to each *transition-variable* can be specified immediately following the transition variable, for example, *transition-variable = expression*, *transition-variable=expression*. Or, sets of parentheses can be used to specify all the *transition-variables* and then all the values, for example, *(transition-variable, transition-variable) = (expression, expression)*.

*correlation-name*

Identifies the correlation name given for referencing the NEW transition variables. The name must match the correlation name specified following NEW in the REFERENCING clause of the CREATE TRIGGER statement.

If OLD is not specified in the REFERENCING clause, *correlation-name* defaults to the correlation name following NEW.

*column-name*

Identifies the column to be updated. The name must identify a column of the subject table. The name can identify an identity column that is defined as GENERATED BY DEFAULT but not one defined as GENERATED ALWAYS. You must not specify the same column more than once.

The effect of a SET *transition-variable* statement is equivalent to the effect of an SQL UPDATE statement.

*expression*

Specifies the value to be assigned to the corresponding *transition-variable*. The expression is any expression of the type described in “Expressions” on page 111. A reference to a *local special register* is the value of that special register at the server when the trigger body is activated.

An expression can contain references to OLD and NEW transition variables that are qualified with a correlation name.

All expressions are evaluated before any result is assigned to a transition variable. If an expression refers to a transition variable that is used in the list of transition variables, the value of the variable in the expression is the value of the variable prior to any assignments.

Each assignment to a transition variable column is made according to the assignment rules described in “Assignment and comparison” on page 64. Assignments are made in sequence through the list. When the *transition-variables* are enclosed within parentheses, for example, *(transition-variable, transition-variable, ...)* = *(expression, expression, ...)*, the first value is assigned to the first transition variable in the list, the second value to the second transition variable in the list, and so on.

**NULL**

Specifies the null value and can only be specified for nullable transition variables.

**VALUES**

Specifies the value to be assigned to the corresponding transition variable.

When more than one value is specified, the values must be enclosed in parentheses. Each value can be an expression or NULL, as described above.

The following syntax is equivalent:

- *(transition-variable, transition-variable)* = *(VALUES(expression, NULL))*
- *(transition-variable, transition-variable)* = *(expression, NULL)*

**Notes**

Normally a locator can be used with a LOB and CLOBs are compatible with CHAR types, but it is not necessarily true that a locator can be used with a CHAR. For more information on using locators, see Part 2 of *DB2 Application Programming and SQL Guide*.

## SET transition-variable assignment

### Examples

*Example 1:* Assume that you want to create a before trigger that sets the salary and commission columns to default values for newly inserted rows in the EMPLOYEE table and that you will define the trigger only with NEW in the REFERENCING clause. Write the SET transition-variable statement that assigns the default values to the SALARY and COMMISSION columns.

```
SET (SALARY, COMMISSION) = (50000, 8000);
```

*Example 2:* Assume that you want to create a before trigger that detects any commission increases greater than 10% for updated rows in the EMPLOYEE table and limits the commission increase to 10%. You will define the trigger with both OLD and NEW in the REFERENCING clause. Write the SET transition-variable statement that limits an increase to the COMMISSION column to 10% .

```
SET NEWROW.COMMISSION = 1.1 * OLDROW.COMMISSION;
```

## SIGNAL SQLSTATE

The SIGNAL SQLSTATE statement is used to signal an error. It causes an error to be returned with the specified SQLSTATE and error description.

### Invocation

This statement can only be used in the triggered action of a trigger.

### Authorization

None required.

### Syntax

```
►►—SIGNAL SQLSTATE—sqlstate-string-constant—(—diagnostic-string-constant—)—————►►
```

### Description

#### *sqlstate-string-constant*

Represents an SQLSTATE. It must be a character string constant with exactly 5 characters that follow these rules for application-defined SQLSTATEs:

- Each character must be from the set of digits ('0' through '9') or non-accented uppercase letters ('A' through 'Z').
- The SQLSTATE class (first two characters) cannot be '00', '01' or '02' because these are not error classes.
- If the SQLSTATE class (first two characters) starts with the character '0' through '6' or 'A' through 'H', the subclass (last three characters) must start with a character in the range 'I' through 'Z'.
- If the SQLSTATE class (first two characters) starts with the character '7', '8', '9', or 'I' through 'Z', the subclass (last three characters) can be any of '0' through '9' or 'A' through 'Z'.

#### *diagnostic-string-constant*

A character string of up to 70 bytes that describes the error condition. If the string is longer than 70 bytes, it is truncated.

### Example

Consider a trigger for an order system that allows orders to be recorded in an ORDERS table (ORDERNO, CUSTNO, PARTNO, QUANTITY) only if there is sufficient stock in the PARTS tables. When there is insufficient stock for an order, SQLSTATE '75001' is returned along with an appropriate error description.

```
CREATE TRIGGER CK_AVAIL
 NO CASCADE BEFORE INSERT ON ORDERS
 REFERENCING NEW AS NEW_ORDER
 FOR EACH ROW MODE DB2SQL
 WHEN (NEW_ORDER.QUANTITY > (SELECT ON_HAND FROM PARTS
 WHERE NEW_ORDER.PARTNO = PARTS.PARTNO))
 BEGIN ATOMIC
 SIGNAL SQLSTATE '75001' ('Insufficient stock for order');
 END
```

## UPDATE

## UPDATE

The UPDATE statement updates the values of specified columns in rows of a table or view. Updating a row of a view updates a row of the table on which the view is based. The table or view can exist at the current server or at any DB2 subsystem with which the current server can establish a connection.

There are two forms of this statement:

- The *searched* UPDATE form is used to update one or more rows optionally determined by a search condition.
- The *positioned* UPDATE form is used to update exactly one row, as determined by the current position of a cursor.

## Invocation

This statement can be embedded in an application program or issued interactively. A positioned UPDATE can be embedded in an application program. Both forms are executable statements that can be dynamically prepared.

## Authorization

Authority requirements depend on whether the object identified in the statement is a user-defined table, a catalog table for which updates are allowed, or a view, and whether SQL standard rules are in effect:

**When a user-defined table is identified:** The privilege set must include at least one of the following:

- The UPDATE privilege on the table
- The UPDATE privilege on each column to be updated
- Ownership of the table
- DBADM authority on the database that contains the table
- SYSADM authority

**When a catalog table is identified:** The privilege set must include at least one of the following:

- The UPDATE privilege on each column to be updated
- DBADM authority on the catalog database
- SYSCTRL authority
- SYSADM authority

**When a view is identified:** The privilege set must include at least one of the following:

- The UPDATE privilege on the view
- The UPDATE privilege on each column to be updated
- SYSADM authority

**When SQL standard rules are in effect:** If SQL standard rules are in effect and an expression in the SET clause contains a reference to a column of the table or view, or if the search-condition in a searched UPDATE contains a reference to a column of the table or view, the privilege set must include at least one of the following:

- The SELECT privilege on the table or view
- SYSADM authority

SQL standard rules are in effect as follows:

- For static SQL statements, if the SQLRULES(STD) bind option was specified
- For dynamic SQL statements, if the CURRENT RULES special register is set to 'STD'

The owner of a view, unlike the owner of a table, might not have UPDATE authority on the view (or might have UPDATE authority without being able to grant it to others). The nature of the view itself can preclude its use for UPDATE. For more information, see the discussion of authority in “CREATE VIEW” on page 711.

If an expression that refers to a function is specified, the privilege set must include any authority that is necessary to execute the function.

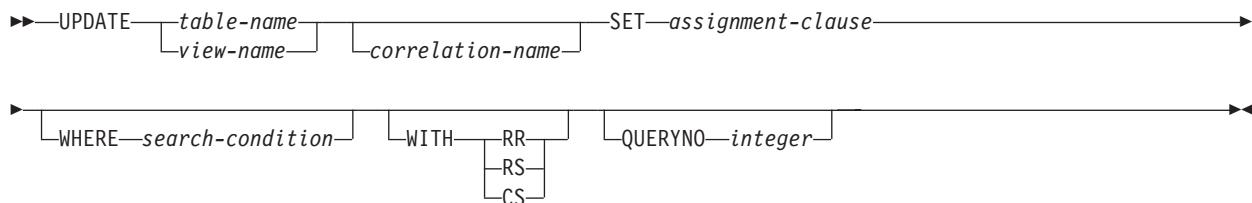
If a subselect is specified, the privilege set must include authority to execute the subselect. For more information about the subselect authorization rules, see “Authorization” on page 348.

**Privilege set:** If the statement is embedded in an application program, the privilege set is the privileges that are held by the authorization ID of the owner of the plan or package. If the statement is dynamically prepared, the privilege set is determined by the DYNAMICRULES behavior in effect (run, bind, define, or invoke) and is summarized in Table 40 on page 382. (For more information on these behaviors, including a list of the DYNAMICRULES bind option values that determine them, see “Authorization IDs and dynamic SQL” on page 43).

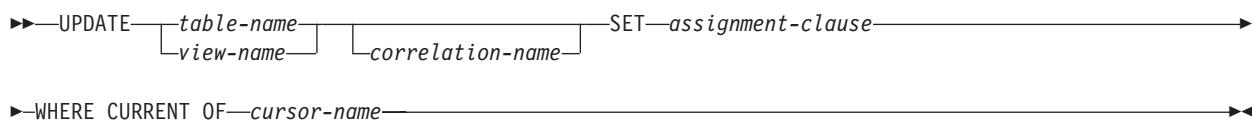
## UPDATE

### Syntax

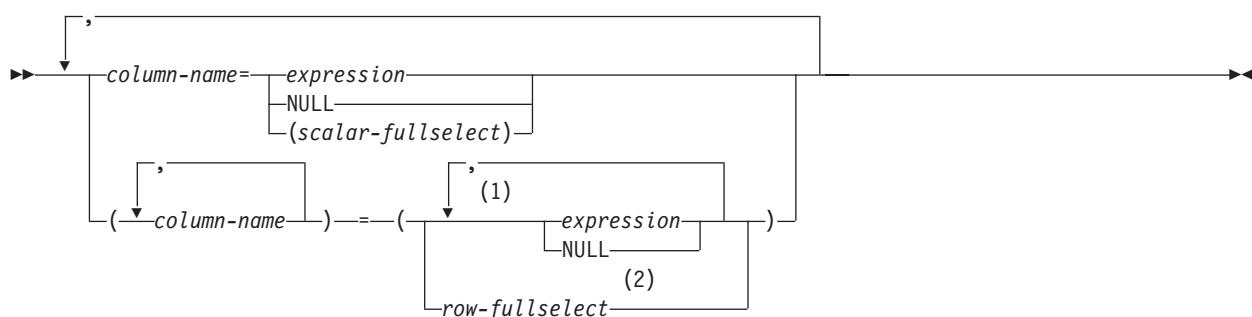
#### searched update:



#### positioned update:



#### assignment clause:



#### Notes:

- 1 The number of *expressions* and *NULL* keywords must match the number of *column-names*.
- 2 The number of columns in the select list must match the number of *column-names*.

### Description

*table-name* or *view-name*

Identifies the object of the UPDATE statement. The name must identify a table or view that exists at the DB2 subsystem identified by the implicitly or explicitly specified location name. The name must not identify:

- An auxiliary table
- A created temporary table or a view of a created temporary table
- A catalog table with no updatable columns or a view of a catalog table with no updatable columns
- A read-only view. (For a description of a read-only view, see "CREATE VIEW" on page 711.)

In the IMS or CICS environments, the DB2 subsystem that contains the identified table or view must not be a remote Version 2 Release 3 subsystem.

A catalog table or a view of a catalog table can be identified if every column identified in the SET clause is an updatable column. If a column of a catalog table is updatable, then its description in Appendix D, "DB2 catalog tables," on page 1005 indicates that the column can be updated. If the object table is SYSIBM.SYSSTRINGS, any column other than IBMREQD can be updated, but the rows selected for update must be rows provided by the user (the value of the IBMREQD column is N) and only certain values can be specified as explained in Appendix B (Volume 2) of *DB2 Administration Guide*.

*correlation-name*

Can be used within *search-condition* or positioned UPDATE to designate the table or view. (For an explanation of *correlation-name*, see "Correlation names" on page 95.)

**SET**

Introduces a list of one or more column names and the values to be assigned to the columns.

*column-name*

Identifies a column to be updated. *column-name* must identify a column of the specified table or view, but must not identify a ROWID column, an identity column that is defined as GENERATED ALWAYS, or a view column that is derived from a scalar function, constant, or expression. The column names must not be qualified, and the same column must not be specified more than once.

For a positioned update, allowable column names can be further restricted to those in a certain list. This list appears in the FOR UPDATE clause of the SELECT statement for the associated cursor. The clause can be omitted using the conditions described in "Positioned updates of columns" on page 153.

A view column derived from the same column as another column of the view can be updated, but both columns cannot be updated in the same UPDATE statement.

*expression*

Indicates the new value of the column. The *expression* is any expression of the type described in "Expressions" on page 111. It must not include a column function.

A *column-name* in an expression must identify a column of the table or view. For each row that is updated, the value of the column in the expression is the value of the column in the row before the row is updated.

**NULL**

Specifies the null value as the new value of the column. Specify NULL only for nullable columns.

*scalar-fullselect*

Specifies a fullselect that returns a single row with a single column. The column value is assigned to the corresponding *column-name*. If the fullselect returns no rows, the null value is assigned; an error occurs if the column to be updated is not nullable. An error also occurs if there is more than one row in the result.

## UPDATE

For a positioned update, if the table or view that is the object of the UPDATE statement is used in the fullselect, the column from the instance of the table or view in the fullselect cannot be the same as *column-name*, the column being updated.

The fullselect must not contain a GROUP BY or HAVING clause. If the fullselect refers to columns to be updated, the value of such a column in the fullselect is the value of the column in the row before the row is updated.

### *row-fullselect*

Specifies a fullselect that returns a single row. The column values are assigned to each of the corresponding *column-names*. If the fullselect returns no rows, the null value is assigned to each column; an error occurs if any column to be updated is not nullable. An error also occurs if there is more than one row in the result.

For a positioned update, if the table or view that is the object of the UPDATE statement is used in the fullselect, a column from the instance of the table or view in the fullselect cannot be the same as *column-name*, a column being updated.

The fullselect must not contain a GROUP BY or HAVING clause. If the fullselect refers to columns to be updated, the value of such a column in the fullselect is the value of the column in the row before the row is updated.

## WHERE

Specifies the rows to be updated. You can omit the clause, give a search condition, or name a cursor. If you omit the clause, all rows of the table or view are updated.

### *search-condition*

Is any search condition described in Chapter 2, “Language elements,” on page 27. Each *column-name* in the search condition, other than in a subquery, must identify a column of the table or view.

The search condition is applied to each row of the table or view and the updated rows are those for which the result of the *search-condition* is true. If the unique key or primary key is a parent key, the constraints are effectively checked at the end of the operation.

If the search condition contains a subquery, the subquery can be thought of as being executed each time the search condition is applied to a row, and the results used in applying the search condition. In actuality, a subquery with no correlated references is executed just once, whereas it is possible that a subquery with a correlated reference must be executed once for each row.

### **CURRENT OF** *cursor-name*

Identifies the cursor to be used in the update operation. The cursor name must identify a declared cursor as explained in “DECLARE CURSOR” on page 718.

If the UPDATE statement is embedded in a program, the DECLARE CURSOR statement must include a select-statement rather than a statement-name.

The object of the UPDATE statement must also be identified in the FROM clause of the SELECT statement of the cursor. The columns to be updated can be identified in the FOR UPDATE clause of that SELECT statement though they do not have to be identified. If the columns are not specified,

the columns that can be updated include all the updatable columns of the table or view that is identified in the first FROM clause of the fullselect.

The result table of the cursor must not be read-only. For an explanation of read-only result tables, see “Read-only cursors” on page 721. Note that the object of the UPDATE statement must not be identified as the object of the subquery in the WHERE clause of the SELECT statement of the cursor.

When the UPDATE statement is executed, the cursor must be positioned on the row to be updated.

If the application process has another cursor positioned on the updated row, the position of that cursor is changed to be before the next row.

The successful or unsuccessful execution of a positioned update operation does not change the position of the cursor. However, it is possible for an error to make the position of the error invalid, in which case, the cursor is closed. It is also possible for an update operation to cause a rollback, in which case, the cursor is closed.

#### WITH

Specifies the isolation level used when locating the rows to be updated by the statement.

**RR**    Repeatable read

**RS**    Read stability

**CS**    Cursor stability

The default isolation level of the statement is the isolation level of the package or plan in which the statement is bound, with the package isolation taking precedence over the plan isolation. When a package isolation is not specified, the plan isolation is the default.

#### QUERYNO *integer*

Specifies the number to be used for this SQL statement in EXPLAIN output and trace records. The number is used for the QUERYNO column of the plan table for the rows that contain information about this SQL statement. This number is also used in the QUERYNO column of the SYSIBM.SYSSTMT and SYSIBM.SYSPACKSTMT catalog tables.

If the clause is omitted, the number associated with the SQL statement is the statement number assigned during precompilation. Thus, if the application program is changed and then precompiled, that statement number might change.

Using the QUERYNO clause to assign unique numbers to the SQL statements in a program is helpful:

- For simplifying the use of optimization hints for access path selection
- For correlating SQL statement text with EXPLAIN output in the plan table

For information on using optimization hints, such as enabling the system for optimization hints and setting valid hint values, and for information on accessing the plan table, see Part 5 (Volume 2) of *DB2 Administration Guide*.

## Notes

**Update rules:** Update values must satisfy the following rules. If they do not, or if other errors occur during the execution of the UPDATE statement, no rows are updated and the position of the cursors are not changed.

- **Assignment.** Update values are assigned to columns using the assignment rules described in Chapter 2, “Language elements,” on page 27.

## UPDATE

- *Uniqueness constraints.* The updated row must conform to any constraints imposed on the table (or on the base table of the view) by any unique index on an updated column. For a multiple-row update of a unique key, the uniqueness constraint is effectively checked at the end of the operation.
- *Referential constraints.* A nonnull update value of a foreign key must be equal to some value of the parent key of the parent table of the relationship.
- *Check constraints.* The table (or base table of the view) might have one or more check constraints. Each row updated must conform to the conditions imposed by those check constraints. Thus, each check condition must be true or unknown.
- *Field and validation procedures.* The updated row must conform to any constraints imposed by any field or validation procedures on the table (or on the base table of the view).
- *Views and the WITH CHECK OPTION.* For views defined with WITH CHECK OPTION, an updated row must conform to the definition of the view. If the view you name is dependent on other views whose definitions include WITH CHECK OPTION, the updated rows must also conform to the definitions of those views. For an explanation of the rules governing this situation, see “CREATE VIEW” on page 711.

For views that are not defined with WITH CHECK OPTION, you can change the rows so that they no longer conform to the definition of the view. Such rows are updated in the base table of the view and no longer appear in the view.

- *Triggers.* An UPDATE statement might cause triggers to be activated. A trigger might cause other statements to be executed or raise error conditions based on the update values.

**Number of rows updated:** Normally, after an UPDATE statement completes execution, the value of SQLERRD(3) in the SQLCA is the number of rows updated. (For a complete description of the SQLCA, including exceptions to the preceding sentence, see “SQL communication area (SQLCA)” on page 979.)

**Nesting user-defined functions or stored procedures:** An UPDATE statement can implicitly or explicitly refer to user-defined functions or stored procedures. This is known as *nesting* of SQL statements. A user-defined function or stored procedure that is nested within the UPDATE must not access the table being updated.

**Locking:** Unless appropriate locks already exist, one or more exclusive locks are acquired by the execution of a successful UPDATE statement. Until a commit or rollback operation releases the locks, only the application process that performed the insert can access the updated row. If LOBs are not updated, application processes that are running with uncommitted read can also access the updated row. The locks can also prevent other application processes from performing operations on the table. However, application processes that are running with uncommitted read can access locked pages and rows.

Locks are not acquired on declared temporary tables.

**Updating keys of partitioning indexes:** If an updated column is a partitioning key or part of a partitioning key and the update causes a row to move to a different partition, DB2 tries to take exclusive control of the following objects to perform the update:

- The partition of the table space in which the row resides, the partition to which the row is moving, and all the partitions in between the two partitions
- The partition of the partitioning index in which the key resides, the partition to which the key is moving, and all the partitions in between the two partitions

- The nonpartitioning indexes defined on the table space

If DB2 cannot take control of these objects, the update fails.

**Datetime representation when using datetime registers:** As explained under “Datetime special registers” on page 84, when two or more datetime registers are implicitly or explicitly specified in a single SQL statement, they represent the same point in time. This is also true when multiple rows are updated.

**Rules for positioned UPDATE with a SENSITIVE STATIC scrollable cursor:**

When a SENSITIVE STATIC scrollable cursor has been declared, the following rules apply:

- *Update attempt of delete holes.* If, with a positioned update against a SENSITIVE STATIC scrollable cursor, an attempt is made to update a row that has been identified as a delete hole, an error occurs.
- *Update operations.* Positioned update operations with SENSITIVE STATIC scrollable cursors perform as follows:
  1. The SELECT list items in the target row of the base table of the cursor are compared with the values in the corresponding row of the result table (that is, the result table must still agree with the base table). If the values are not identical, then the update operation is rejected, and an error occurs. The operation may be attempted again after a successful FETCH SENSITIVE has occurred for the target row.
  2. The WHERE clause of the SELECT statement is re-evaluated to determine whether the current values in the base table still satisfy the search criteria. The values in the SELECT list are compared to determine that these values have not changed. If the WHERE clause evaluates as true, and the values in the SELECT have not changed, the update operation is allowed to proceed. Otherwise, the update operation is rejected, an error occurs, and an *update hole* appears in the cursor.
- *Update of update holes.* Update holes are not permanent. It is possible for another process, or a searched update in the same process, to update an update hole row so that it is no longer an update hole. Update holes become visible with a FETCH SENSITIVE for positioned updates and positioned deletes.
- *Result table.* After the base table is updated, the row is re-evaluated and updated in the temporary result table. At this time, it is possible that the positioned update changed the data such that the row does not qualify the search condition, in which case the row is marked as an update hole for subsequent FETCH operations.

## Examples

The following examples refer to the sample table DSN8710.EMP.

*Example 1:* Change employee 000190’s telephone number to 3565 in DSN8710.EMP.

```
UPDATE DSN8710.EMP
SET PHONENO='3565'
WHERE EMPNO='000190';
```

*Example 2:* Give each member of department D11 a 100-dollar raise.

```
UPDATE DSN8710.EMP
SET SALARY = SALARY + 100
WHERE WORKDEPT = 'D11';
```

## UPDATE

*Example 3:* Employee 000250 is going on a leave of absence. Set the employee's pay values (SALARY, BONUS, and COMMISSION) to null.

```
UPDATE DSN8710.EMP
SET SALARY = NULL, BONUS = NULL, COMM = NULL
WHERE EMPNO='000250';
```

Alternatively, the statement could also be written as follows:

```
UPDATE DSN8710.EMP
SET (SALARY, BONUS, COMM) = (NULL, NULL, NULL)
WHERE EMPNO='000250';
```

*Example 4:* Assume that a column named PROJSIZE has been added to DSN8710.EMP. The column records the number of projects for which the employee's department has responsibility. For each employee in department E21, update PROJSIZE with the number of projects for which the department is responsible.

```
UPDATE DSN8710.EMP
SET PROJSIZE = (SELECT COUNT(*)
 FROM DSN8710.PROJ
 WHERE DEPTNO = 'E21')
WHERE WORKDEPT = 'E21';
```

*Example 5:* Double the salary of the employee represented by the row on which the cursor C1 is positioned.

```
EXEC SQL UPDATE DSN8710.EMP
SET SALARY = 2 * SALARY
WHERE CURRENT OF C1;
```

*Example 6:* Assume that employee table EMP1 was created with the following statement:

```
CREATE TABLE EMP1
(EMP_ROWID ROWID GENERATED ALWAYS,
EMPNO CHAR(6),
NAME CHAR(30),
SALARY DECIMAL(9,2),
PICTURE BLOB(250K),
RESUME CLOB(32K));
```

Assume that host variable HV\_EMP\_ROWID contains the value of the ROWID column for employee with employee number '350000'. Using that ROWID value to identify the employee and user-defined function UPDATE\_RESUME, increase the employee's salary by \$1000 and update that employee's resume.

```
EXEC SQL UPDATE EMP1
SET SALARY = SALARY + 1000,
RESUME = UPDATE_RESUME(:HV_RESUME)
WHERE EMP_ROWID = :HV_EMP_ROWID;
```

*Example 7:* In employee table X, give each employee whose salary is below average a salary increase of 10%.

```
EXEC SQL UPDATE EMP X
SET SALARY = 1.10 * SALARY
WHERE SALARY < (SELECT AVG(SALARY) FROM EMP Y
WHERE X.JOBCODE = Y.JOBCODE);
```

*Example 8:* Raise the salary of the employees in department 'E11' whose salary is below average to the average salary.

```
| EXEC SQL UPDATE EMP T1
| SET SALARY = (SELECT AVG(T2.SALARY) FROM EMP T2)
| WHERE WORKDEPT = 'E11' AND
| SALARY < (SELECT AVG(T3.SALARY) FROM EMP T3);
```

*Example 9:* Give the employees in department 'E11' a bonus equal to 10% of their salary.

```
| EXEC SQL
| DECLARE C1 CURSOR FOR
| SELECT BONUS
| FROM DSN8710.EMP
| WHERE WORKDEPT = 'E12'
| FOR UPDATE OF BONUS;

| EXEC SQL
| UPDATE DSN8710.EMP
| SET BONUS = (SELECT .10 * SALARY FROM DSN8710.EMP Y
| WHERE EMPNO = Y.EMPNO)
| WHERE CURRENT OF C1;
```

## VALUES

## VALUES

The VALUES statement provides a method for invoking a user-defined function from a trigger. Transition variables and transition tables can be passed to the user-defined function.

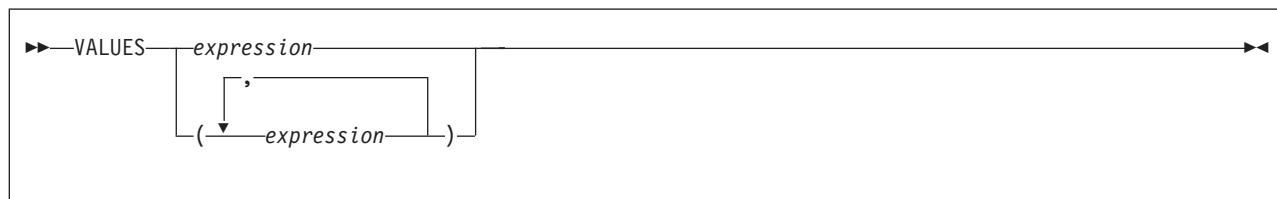
### Invocation

This statement can only be used in the triggered action of a trigger.

### Authorization

EXECUTE authority is needed on any user-defined function that is referenced in the VALUES clause.

### Syntax



### Description

#### VALUES

Specifies one or more expressions. If more than one expression is specified, the expressions must be enclosed within parentheses.

#### expression

Any expression of the type described in “Expressions” on page 111. The expression must not contain a host variable.

The expressions are evaluated, but the resulting values are discarded and are not assigned to any output variables.

If a user-defined function is specified as part of an expression, the user-defined function is invoked. If a negative SQLCODE is returned when the function is invoked, DB2 stops executing the trigger and rolls back any triggered actions that were performed.

### Example

*Example:* Create an after trigger EMPISRT1 that invokes user-defined function NEWEMP when the trigger is activated. An insert operation on table EMP activates the trigger. Pass transition variables for the new employee number, last name, and first name to the user-defined function.

```
CREATE TRIGGER EMPISRT1
 AFTER INSERT ON EMP
 REFERENCING NEW AS N
 FOR EACH ROW
 MODE DB2SQL
 BEGIN ATOMIC
 VALUES(NEWEMP(N.EMPNO, N.LASTNAME, N.FIRSTNAME));
 END
```

## VALUES INTO

The VALUES INTO statement assigns one or more values to host variables.

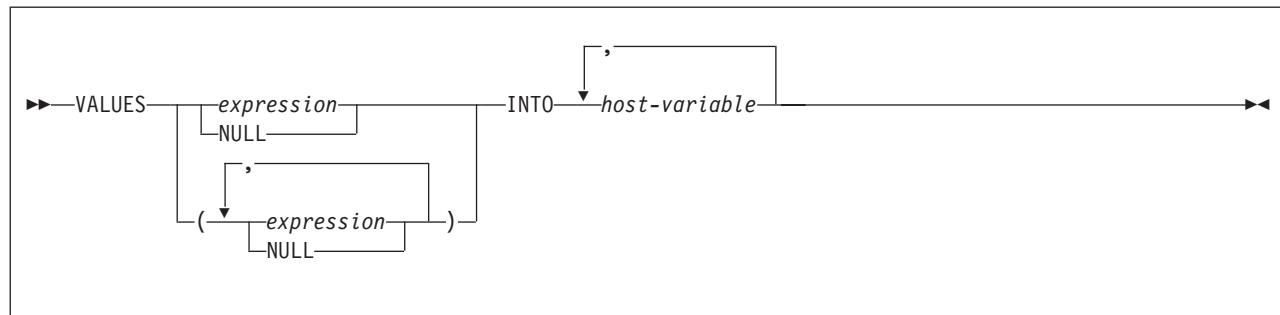
### Invocation

This statement can only be embedded in an application program. It is an executable statement that cannot be dynamically prepared.

### Authorization

EXECUTE authority is needed on any user-defined function that is referenced in the VALUES statement.

### Syntax



### Description

#### VALUES

Introduces one or more values. If more than one value is specified, the list of values must be enclosed within parentheses.

#### *expression*

Any expression of the type described in “Expressions” on page 111. The expression must not include a column name.

#### NULL

The null value. NULL can only be specified for host variables that have an associated indicator variable.

#### INTO

Introduces a list of host variables and host structures. The values that are specified in the VALUES clause are assigned to these host variables. The first value in the result row is assigned to the first host variable in the list, the second value to the second host variable, and so on. Each assignment is made according to the rules described in “Assignment and comparison” on page 64. Assignments are made in sequence through the list. If there are fewer host variables than values, the value 'W' is assigned to the SQLWARN3 field of the SQLCA. (See “SQL communication area (SQLCA)” on page 979.)

#### *host-variable*

Identifies one or more host structures or host variables that must be declared in accordance with the rules for declaring host structures and host variables. In the operational form of INTO, a host structure is replaced by a reference to each of its variables.

## VALUES INTO

### Notes

The default encoding scheme for the data is the value in the bind option ENCODING, which is the option for application encoding. If this statement is used with functions such as LEN or SUBSTR that are operating on LOB locators, and the LOB data that is specified by the locator is in a different encoding scheme from the ENCODING bind option, LOB materialization and character conversion occur. To avoid LOB materialization and character conversion, select the LOB data from the SYSIBM.SYSDUMMYA, SYSIBM.SYSDUMMYE, or SYSIBM.SYSDUMMYU sample table.

If an error occurs, no value is assigned to any host variable. However, if LOB values are involved, there is a possibility that the corresponding host variable was modified, but the variable's contents are unpredictable.

Local special registers can be referenced only in a VALUES INTO statement that results in the assignment of a single host variable and not those that result in setting more than one value.

Normally a locator can be used with a LOB and CLOBs are compatible with CHAR types, but it is not necessarily true that a locator can be used with a CHAR. For more information on using locators, see Part 2 of *DB2 Application Programming and SQL Guide*.

### Examples

*Example 1:* Assign the value of the CURRENT PATH special register to host variable HV1.

```
EXEC SQL VALUES(CURRENT PATH)
 INTO :HV1;
```

*Example 2:* Assign the value of the CURRENT MEMBER special register to host variable MEM.

```
EXEC SQL VALUES(CURRENT MEMBER)
 INTO :MEM;
```

*Example 3:* Assume that LOB locator LOB1 is associated with a CLOB value. Assign a portion of the CLOB value to host variable DETAILS using the LOB locator.

```
EXEC SQL VALUES (SUBSTR(:LOB1,1,35))
 INTO :DETAILS;
```

If the LOB data that is specified by the LOB locator LOB1 is in a different encoding scheme from the value of the ENCODING bind option, and you want to avoid LOB materialization and character conversion, use the following statement instead of the VALUES INTO statement:

```
EXEC SQL SELECT SUBSTR(:LOB1,1,35)
 INTO :DETAILS
 FROM SYSIBM.SYSDUMMYU;
```

#

## WHENEVER

The WHENEVER statement specifies the host language statement to be executed when a specified exception condition occurs.

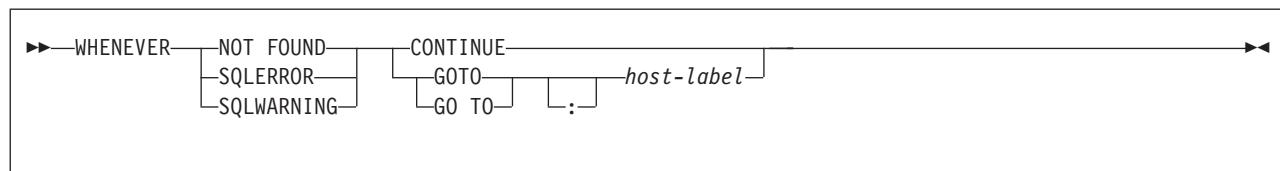
### Invocation

This statement can only be embedded in an application program, except in REXX programs. It is not an executable statement.

### Authorization

None required.

### Syntax



### Description

The NOT FOUND, SQLERROR, or SQLWARNING clause is used to identify the type of exception condition.

#### NOT FOUND

Identifies any condition that results in an SQLCODE of +100 (equivalently, an SQLSTATE code of '02000').

#### SQLERROR

Identifies any condition that results in a negative SQLCODE.

#### SQLWARNING

Identifies any condition that results in a warning condition (SQLWARN0 is W), or that results in a positive SQLCODE other than +100.

The CONTINUE or GO TO clause specifies the next statement to be executed when the identified type of exception condition exists.

#### CONTINUE

Specifies the next sequential statement of the source program.

#### GOTO or GO TO *host-label*

Specifies the statement identified by *host-label*. For *host-label*, substitute a single token, optionally preceded by a colon. The form of the token depends on the host language. In COBOL, for example, it can be *section-name* or an unqualified *paragraph-name*.

### Notes

There are three types of WHENEVER statements:

- WHENEVER NOT FOUND
- WHENEVER SQLERROR
- WHENEVER SQLWARNING

Every executable SQL statement in an application program is within the scope of one implicit or explicit WHENEVER statement of each type. The scope of a

## WHENEVER

WHENEVER statement is related to the listing sequence of the statements in the application program, not their execution sequence.

An SQL statement is within the scope of the last WHENEVER statement of each type that is specified before that SQL statement in the source program. If a WHENEVER statement of some type is not specified before an SQL statement, that SQL statement is within the scope of an implicit WHENEVER statement of that type in which CONTINUE is specified. If a WHENEVER statement is specified in a Fortran subprogram, its scope is that subprogram, not the source program.

## Examples

The following statements can be embedded in a COBOL program.

*Example 1:* Go to the label HANDLER for any statement that produces an error.

```
EXEC SQL WHENEVER SQLERROR GOTO HANDLER END-EXEC.
```

*Example 2:* Continue processing for any statement that produces a warning.

```
EXEC SQL WHENEVER SQLWARNING CONTINUE END-EXEC.
```

*Example 3:* Go to the label ENDDATA for any statement that does not return.

```
EXEC SQL WHENEVER NOT FOUND GO TO ENDDATA END-EXEC.
```

---

## Chapter 6. SQL procedure statements

An SQL procedure consists of a CREATE PROCEDURE statement with a procedure body. The procedure body contains the source statements for the stored procedure, which are called *SQL procedure statements*.

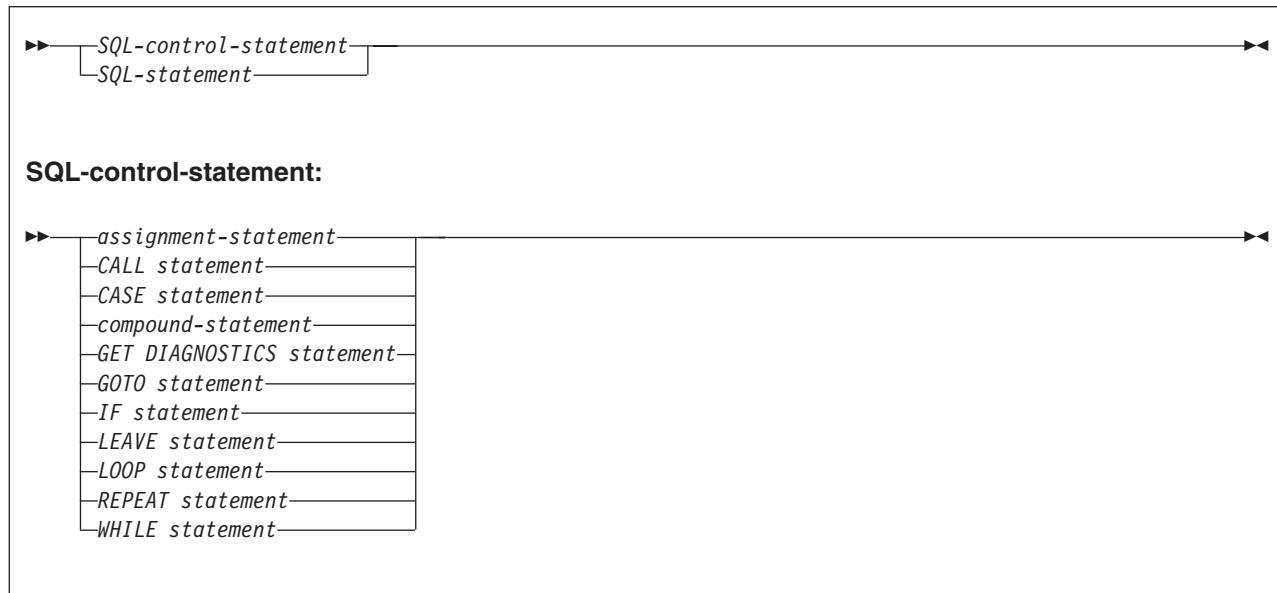
This chapter contains syntax diagrams, semantic descriptions, rules, and examples of the use of the statements that constitute the procedure body.

## SQL-procedure-statement

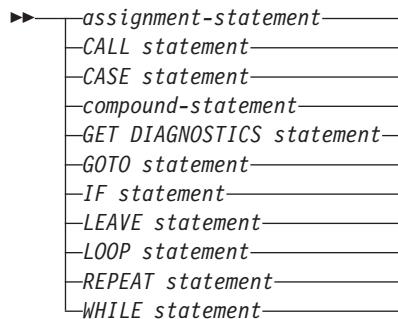
### SQL-procedure-statement

If an SQL control statement is specified as the procedure body, multiple statements can be specified within the control statement. These statements are defined as SQL procedure statements.

### Syntax



#### SQL-control-statement:



### Description

#### *SQL-control-statement*

Specifies an SQL statement that provides the capability to control logic flow, declare and set variables, and handle warnings and exceptions, as defined in this chapter. Control statements are supported in SQL procedures.

#### *SQL-statement*

Specifies an SQL statement as listed in Table 73 on page 975. These statements are described in Chapter 5, “Statements,” on page 377.

### Notes

**Comments:** Comments can be included within the body of an SQL procedure. A comment begins with /\* and ends with \*/. The following rules apply:

- The beginning characters /\* must be on the same line.
- The ending characters \*/ must be on the same line.
- Comments can be started wherever a space is valid.
- Comments can be continued to the next line.

**Resolving names:** The name of an SQL parameter or SQL variable in an SQL procedure can be the same as the name of an identifier used in certain SQL statements. If the name is not qualified, the following rules describe whether the name refers to the identifier or to the SQL parameter or SQL variable:

- In the SET PATH statement, the name is checked as an SQL parameter or SQL variable name. If not found as an SQL variable or SQL parameter name, it will then be used as an identifier.

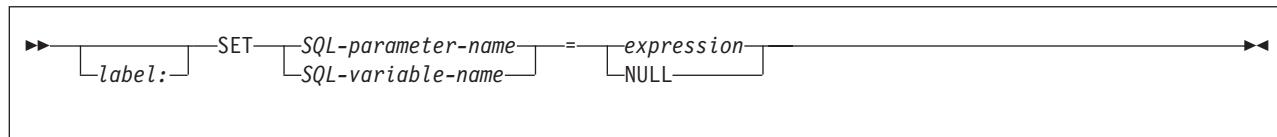
- | • In the CONNECT statement, the SET CONNECTION statement, and the RELEASE (connection) statement the name is used as an identifier.

## **assignment-statement (SQL procedure)**

# assignment-statement

The assignment statement assigns a value to an output parameter or to an SQL variable.

## Syntax



## Description

*label*

Specifies the label for the assignment statement. A label is an SQL ordinary identifier that is 1 to 64 bytes in length. The label must be unique within the SQL procedure.

*SQL-parameter-name*

Identifies the parameter that is the assignment target. The parameter must be specified in *parameter-declaration* in the CREATE PROCEDURE statement and must be defined as OUT or INOUT.

*SQL-variable-name*

**Variable Name**  
Identifies the SQL variable that is the assignment target. SQL variables can only be declared in a compound-statement and must be declared before it is used. For information on declaring SQL variables, see “compound-statement” on page 952.

*expression* or **NULL**

Specifies the expression or value that is the assignment source. See “Expressions” on page 111 for information on expressions.

## Notes

Assignment statements in SQL procedures must conform to the SQL assignment rules. For example, the data type of the target and source must be compatible. See “Assignment and comparison” on page 64 for assignment rules.

If an assignment statement is the only statement in the procedure body, the statement cannot end with a semicolon. Otherwise, the statement must end with a semicolon.

**Assigning string values:** When a string is assigned to a fixed-length variable and the length of the string is less than the length attribute of the target, the string is padded on the right with the necessary number of single-byte or double-byte blanks. When a string is assigned to a variable and the string is longer than the length attribute of the variable, a warning SQLSTATE is set and the value is truncated.

#  
# length attribute of the variable, a warning SQLSTATE is set and the value is  
# truncated.  
  
# The ENCODING bind option is not used during processing of assignments to string  
# variables. For example, assume that the system does not use mixed or DBCS, and  
# the system EBCDIC SBCS CCSID is 37. Character conversion will not occur on  
# assignment even if CCSID 500 is specified for the ENCODING bind parameter for  
# the package for the procedure.

The ENCODING bind option is not used during processing of assignments to string variables. For example, assume that the system does not use mixed or DBCS, and the system EBCDIC SBCS CCSID is 37. Character conversion will not occur on assignment even if CCSID 500 is specified for the ENCODING bind parameter for the package for the procedure.

#  
#  
  
**Assigning numeric values:** If truncation of the whole part of a number occurs on assignment to a numeric variable, a warning SQLSTATE is set and the value is truncated.

|  
|  
|  
|  
|  
|  
|  
**Assignment rules for procedure parameters:** An IN parameter can appear on the left or right side of an assignment statement. When control returns to the caller, the original value of an IN parameter is passed to the caller. An OUT parameter can also appear on the left or right side of an assignment statement. When control returns to the caller, the last value that is assigned to an OUT parameter is returned to the caller. For an INOUT parameter, the first value of the parameter is determined by the caller, and the last value that is assigned to the parameter is returned to the caller.  
|

## Examples

Increase the SQL variable p\_salary by 10 percent.

```
SET p_salary = p_salary + (p_salary * .10)
```

Set SQL variable p\_salary to the null value.

```
SET p_salary = NULL
```

Set SQL variable midinit to the first character of SQL variable midname.

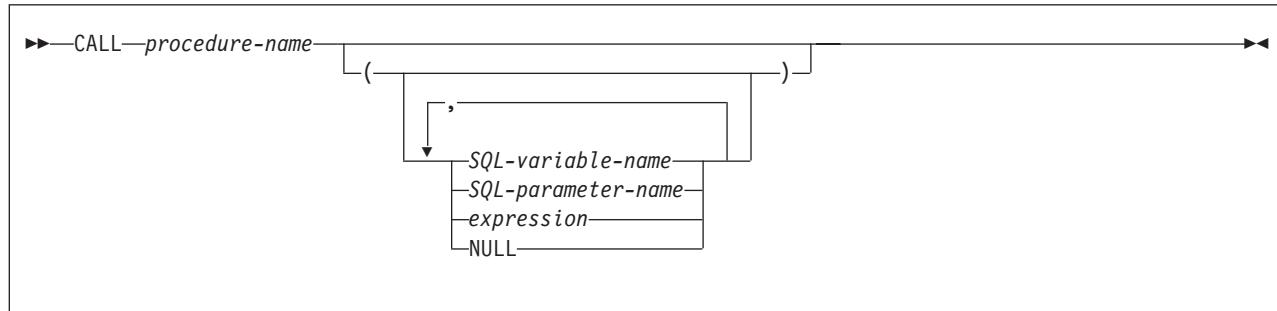
```
SET midinit = SUBSTR(midname,1,1)
```

## CALL (SQL procedure)

### CALL statement

The CALL statement invokes a stored procedure.

### Syntax



### Description

#### *procedure-name*

Identifies the stored procedure to call. The procedure name must identify a stored procedure that exists at the current server.

#### **Parameters (SQL-variable-name, SQL-parameter-name, expression, NULL)**

Identifies a list of values to be passed as parameters to the stored procedure. The number of parameters must be the same as the number of parameters defined for the stored procedure. See “CALL” on page 483 for more information.

Control is passed to the stored procedure according to the calling conventions for SQL procedures. When execution of the stored procedure is complete, the value of each parameter of the stored procedure is assigned to the corresponding parameter of the CALL statement defined as OUT or INOUT.

#### *SQL-variable-name or SQL-parameter-name*

Identifies a parameter to pass to or from the stored procedure. The data type must be compatible with the data type of the corresponding parameter in the stored procedure.

#### *expression*

The parameter is the result of the specified *expression*, which is evaluated before the stored procedure is invoked. If *expression* is a single *SQL-parameter-name* or *SQL-variable-name*, the corresponding parameter of the procedure can be defined as IN, INOUT, or OUT. Otherwise, the corresponding parameter of the procedure must be defined as IN. If the result of the *expression* can be the null value, either the description of the procedure must allow for null parameters or the corresponding parameter of the stored procedure must be defined as OUT.

The following additional rules apply depending on how the corresponding parameter was defined in the CREATE PROCEDURE statement for the procedure:

- IN *expression* can contain references to multiple SQL parameters or variables. In addition to the rules stated in “Expressions” on page 111 for *expression*, *expression* cannot include a column name or column function or a user-defined function that is sourced on a column function.
- INOUT or OUT *expression* can only be a single SQL parameter or variable.

**NULL**

The parameter is a null value. The corresponding parameter of the procedure must be defined as IN and the description of the procedure must allow for null parameters.

**Notes**

If a CALL statement is the only statement in the procedure body, the statement cannot end with a semicolon. Otherwise, the statement must end with a semicolon.

See “CALL” on page 483 for more information on the SQL CALL statement.

**Examples**

Call stored procedure proc1 and pass SQL variables as parameters.

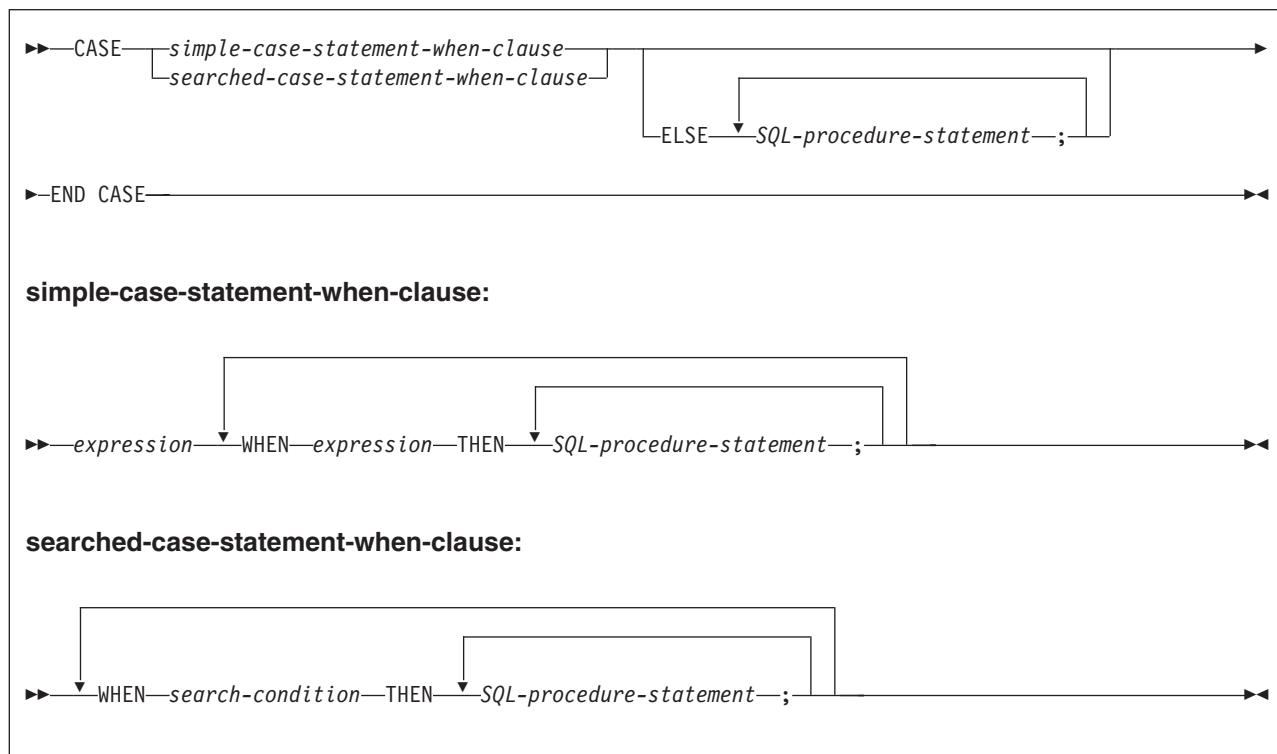
```
CALL proc1(v_empno, v_salary)
```

## CASE (SQL procedure)

### CASE statement

The CASE statement selects an execution path based on the evaluation of one or more conditions. A CASE statement operates in the same way as a CASE expression, which is discussed in “CASE expressions” on page 124.

### Syntax



### Description

#### CASE

Begins a *case-expression*.

#### simple-case-statement-when-clause

Specifies the *expression* prior to the first WHEN keyword that is tested for equality with the value of each *expression* that follows the WHEN keyword, and the result to be executed when those expressions are equal. If the comparison is true, the THEN statement is executed. If the result is unknown or false, processing continues to the next expression or the ELSE statement.

The data type of the *expression* prior to the first WHEN keyword must be comparable to the data types of each *expression* that follows the WHEN keywords.

#### searched-case-statement-when-clause

Specifies the *search-condition* that is applied to each row or group of table data presented for evaluation, and the result when that condition is true. If the search condition is true, the THEN statement is executed. If the condition is unknown or false, processing continues to the next search condition or the ELSE statement.

#### SQL-procedure-statement

Specifies a statement that follows the THEN and ELSE keyword. The statement

specifies the result of a *searched-case-statement-when-clause* or a *simple-case-statement-when-clause* that is true, or the result if no case is true. The statement must be one of the statements listed under “SQL-procedure-statement” on page 944.

*search-condition*

Specifies a condition that is true, false, or unknown about a row or group of table data. The search condition cannot contain a subselect.

**END CASE**

Ends a *case-statement*.

## Notes

If none of the conditions specified in the WHEN are true, and an ELSE is not specified, an error is issued when the statement executes and the execution of the CASE statement is terminated.

CASE statements that use a simple case statement WHEN clause can be nested up to three levels. CASE statements that use a searched statement WHEN clause have no limit to the number of nesting levels.

If a CASE statement is the only statement in the procedure body, the statement cannot end with a semicolon. Otherwise, the statement must end with a semicolon.

| Ensure that your CASE statement covers all possible execution conditions.

## Examples

Use a simple case statement WHEN clause to update column DEPTNAME in table DEPT, depending on the value of SQL variable v\_workdept.

```
CASE v_workdept
WHEN 'A00'
 THEN UPDATE DEPT SET
 DEPTNAME = 'DATA ACCESS 1';
WHEN 'B01'
 THEN UPDATE DEPT SET
 DEPTNAME = 'DATA ACCESS 2';
ELSE UPDATE DEPT SET
 DEPTNAME = 'DATA ACCESS 3';
END CASE
```

Use a searched case statement WHEN clause to update column DEPTNAME in table DEPT, depending on the value of SQL variable v\_workdept.

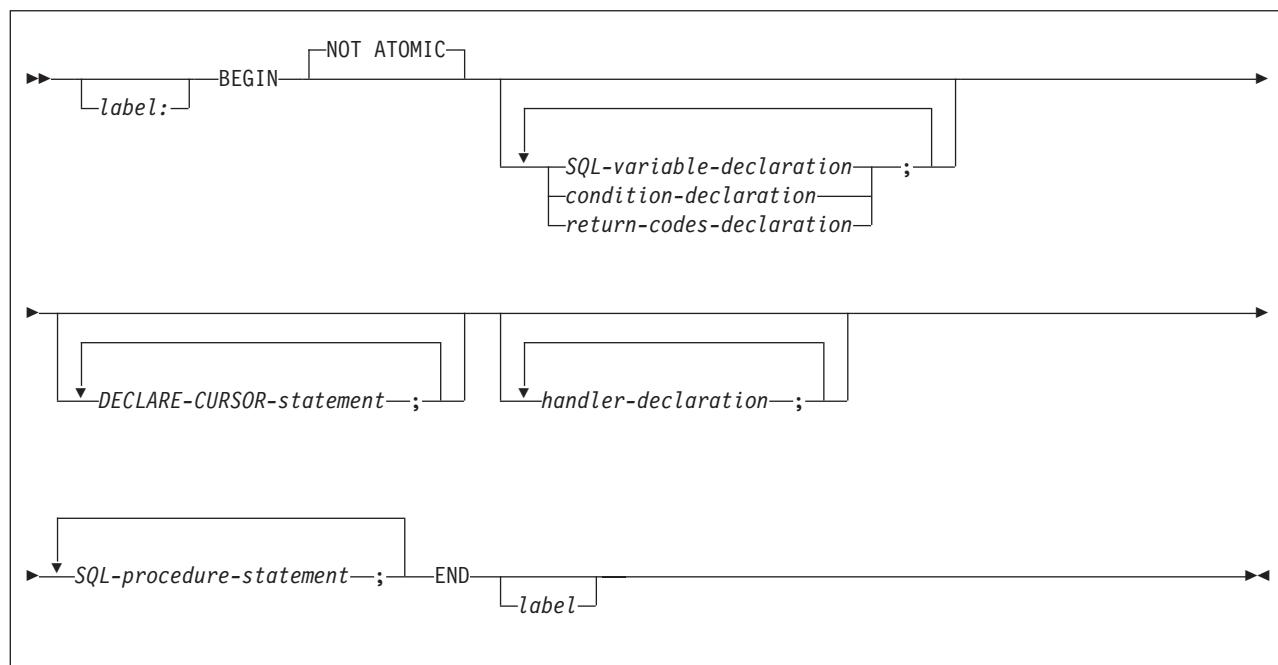
```
CASE
WHEN v_workdept < 'B01'
 THEN UPDATE DEPT SET
 DEPTNAME = 'DATA ACCESS 1';
WHEN v_workdept < 'C01'
 THEN UPDATE DEPT SET
 DEPTNAME = 'DATA ACCESS 2';
ELSE UPDATE DEPT SET
 DEPTNAME = 'DATA ACCESS 3';
END CASE
```

## compound-statement (SQL procedure)

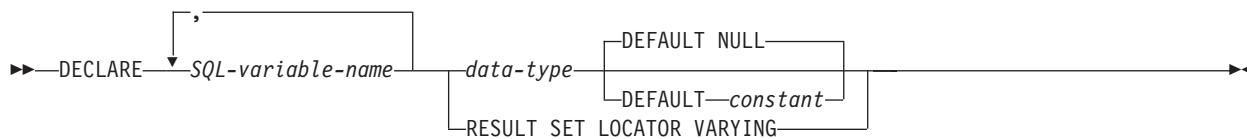
### compound-statement

A compound statement contains a group of statements and declarations for SQL variables, cursors, and condition handlers.

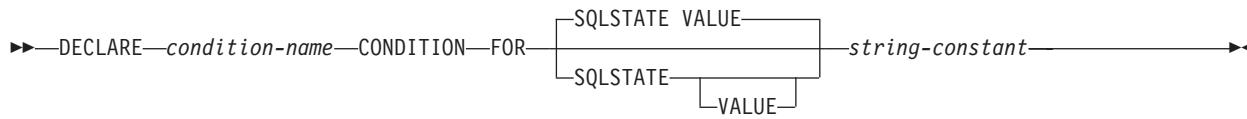
### Syntax

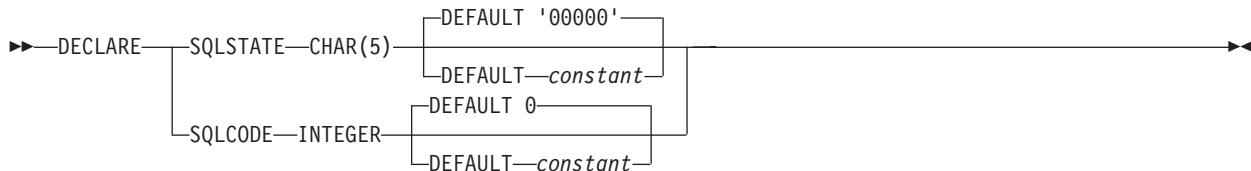
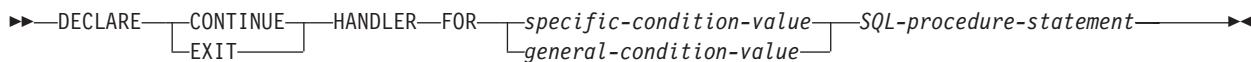


#### SQL-variable-declaration:



#### condition-declaration:



**return-codes-declaration:****handler-declaration:****specific-condition-value:****general-condition-value:**

## Description

*label*

Defines the label for the code block. If the beginning label is specified, it can be used to qualify SQL variables declared in the compound statement and can also be specified on a LEAVE statement. If the ending label is specified, it must be the same as the beginning label.

A label name cannot be the same as the name of the SQL procedure in which the label is used.

**NOT ATOMIC**

NOT ATOMIC indicates that an error within the compound statement does not cause the compound statement to be rolled back.

*SQL-variable-declaration*

Declares a variable that is local to the compound statement.

## **compound-statement (SQL procedure)**

### *SQL-variable-name*

A qualified or unqualified name that designates a variable in an SQL procedure body. The unqualified form of an SQL variable name is an SQL identifier of 1 to 64 bytes. If the SQL variable is a delimited identifier, the contents of the delimited identifier must conform to the rules for ordinary identifiers. The qualified form is an SQL procedure statement label followed by a period (.) and an SQL identifier.

DB2 folds all SQL variable names to uppercase. If an SQL reserved word is used as an SQL variable, the SQL variable must be delimited. SQL variable names should not be the same as column names. If an SQL statement contains an SQL variable or parameter and a column reference with the same name, DB2 interprets the name as an SQL variable or parameter name. To refer to the column, qualify the column name with the table name. Further, to avoid ambiguous variable references and to ensure compatibility with other DB2 platforms, qualify the SQL variable or parameter name with the label of the SQL procedure statement.

### *data-type*

Specifies the data type and length of the variable. SQL variables follow the same rules for default lengths and maximum lengths as SQL procedure parameters. See “CREATE PROCEDURE (SQL)” on page 636 for a description of SQL data types and lengths. An SQL variable cannot have a BLOB, CLOB, DBCLOB data type.

### **DEFAULT *constant* or NULL**

Defines the default for the SQL variable. The variable is initialized when the SQL procedure is called. If a default value is not specified, the variable is initialized to NULL.

### **RESULT\_SET\_LOCATOR VARYING**

Specifies the data type for a result set locator variable.

### *condition-declaration*

Declares a condition name and corresponding SQLSTATE value.

### *condition-name*

Specifies the name of the condition. The condition name is a long SQL identifier that must be unique within the procedure body and can be referenced only within the compound statement in which it is declared.

### **FOR SQLSTATE *string-constant***

Specifies the SQLSTATE that is associated with the condition. The string must be specified as five characters enclosed in single quotes, and cannot be '00000'.

### *return-codes-declaration*

Declares special variables called SQLSTATE and SQLCODE that are set automatically to the value returned after processing an SQL statement. Both the SQLSTATE and SQLCODE variables can be declared only in the outermost compound statement of the SQL procedure. Assignment to these variables is not prohibited; however, assignment is ignored by exception handlers, and processing the next SQL statement replaces the assigned value.

### **DECLARE-CURSOR-statement**

Declares a cursor. Each cursor in the procedure body must have a unique name. An OPEN statement must be specified to open the cursor, and a FETCH statement can be specified to read rows. The cursor can be referenced only from within the compound statement. For more information on declaring a cursor, see “DECLARE CURSOR” on page 718.

*handler-declaration*

Specifies a set of statements to execute when an exception or completion condition occurs in the compound statement. *SQL-procedure-statement* is the set of statements that execute when the handler receives control. See "SQL-procedure-statement" on page 944 for information on *SQL-procedure-statement*.

A handler is active only within the compound statement in which it is declared.

The actions that a handler can perform are:

**CONTINUE**

After the handler is invoked successfully, control is returned to the SQL statement that follows the statement that raised the exception. If the error that raised the exception is an IF, CASE, WHILE, or REPEAT statement, control returns to the statement that follows END IF, END CASE, END WHILE, or END REPEAT.

**EXIT**

After the handler is invoked successfully, control is returned to the end of the compound statement.

The conditions that can cause the handler to gain control are:

**SQLSTATE string**

Specifies an SQLSTATE for which the handler is invoked. The SQLSTATE cannot be '00000'.

*condition-name*

Specifies a condition name for which the handler is invoked. The condition name must be previously defined in a condition declaration.

**SQLEXCEPTION**

Specifies that the handler is invoked when an SQLEXCEPTION occurs. An SQLEXCEPTION is an SQLSTATE in which the class code is a value other than "00", "01", or "02". For more information on SQLSTATE values, see Appendix C of *DB2 Messages and Codes*.

**SQLWARNING**

Specifies that the handler is invoked when an SQLWARNING occurs. An SQLWARNING is an SQLSTATE value with a class code of "01".

**NOT FOUND**

Specifies that the handler is invoked when a NOT FOUND condition occurs. NOT FOUND corresponds to an SQLSTATE value with a class code of "02".

## Notes

The order of statements in a compound statement must be:

1. SQL variable, condition declarations, and return codes declarations
2. Cursor declarations
3. Handler declarations
4. SQL procedure statements

Compound statements cannot be nested.

Unlike host variables, SQL variables are not preceded by colons when they are used in SQL statements.

The following rules apply to handlers:

## compound-statement (SQL procedure)

- A handler declaration that contains SQLEXCEPTION, SQLWARNING, or NOT FOUND cannot contain additional SQLSTATE or condition names.
- Handler declarations within the same compound statement cannot contain duplicate conditions.
- A handler declaration cannot contain the same condition code or SQLSTATE value more than once, and cannot contain an SQLSTATE value and a condition name that represent the same SQLSTATE value.
- A handler is activated when it is the most appropriate handler for an exception or completion condition.
- If there is no handler for an SQL error, the error is passed to the caller in the SQLCA.
- A handler cannot be activated by an assignment statement that assigns a value to SQLSTATE.

The following rules and recommendations apply to the SQLCODE and SQLSTATE special variables:

- A null value cannot be assigned to SQLSTATE or SQLCODE.
- The SQLSTATE and SQLCODE variable values should be saved immediately to temporary variables if there is any intention to use the values. If a handler exists for SQLSTATE, this assignment must be done as the first statement to be processed in the handler to avoid having the value replaced by the next SQL procedure statement. If the condition raised by the SQL statement is handled, the value is changed by the first SQL statement contained in the handler.

If a compound statement is the only statement in the procedure body, the statement cannot end with a semicolon. Otherwise, the statement must end with a semicolon.

## Examples

Create a procedure body with a compound statement that performs the following actions:

- Declares SQL variables, a condition for SQLSTATE '02000', a handler for the condition, and a cursor
- Opens the cursor, fetches a row, and closes the cursor

```
CREATE PROCEDURE PROC1(OUT NOROWS INT) LANGUAGE SQL
BEGIN
 DECLARE v_firstname VARCHAR(12);
 DECLARE v_midinit CHAR(1);
 DECLARE v_lastname VARCHAR(15);
 DECLARE v_edlevel SMALLINT;
 DECLARE v_salary DECIMAL(9,2);
 DECLARE at_end INT DEFAULT 0;
 DECLARE not_found
 CONDITION FOR '02000';
 DECLARE c1 CURSOR FOR
 SELECT FIRSTNAME, MIDINIT, LASTNAME,
 EDLEVEL, SALARY
 FROM EMP;
 DECLARE CONTINUE HANDLER FOR not_found SET NOROWS=1;
 OPEN c1;
 FETCH c1 INTO v_firstname, v_midinit,
 v_lastname, v_edlevel, v_salary;
 CLOSE c1;
END
```

## GET DIAGNOSTICS statement

The GET DIAGNOSTICS statement obtains information about the previous SQL statement that was executed.

### Syntax

```
►—GET DIAGNOSTICS—SQL-variable-name—=—ROW_COUNT—►
```

### Description

#### *SQL-variable-name*

Identifies the SQL variable that is the assignment target. The SQL variable must be declared as an integer variable. For information on declaring SQL variables, see “compound-statement” on page 952.

#### ROW\_COUNT

Identifies the number of rows that are associated with the previous SQL statement that was executed. If the previous SQL statement is a DELETE, INSERT, or UPDATE statement, ROW\_COUNT identifies the number of rows that were deleted, inserted, or updated by the SQL statement. That number does not include rows that were deleted, inserted, or updated because of referential constraints or triggered actions. If the previous statement is another SQL statement, the value that is returned has no meaning.

### Notes

The GET DIAGNOSTICS statement does not change the contents of the SQLCA. If SQLCODE and SQLSTATE variables are declared in the SQL procedure, those variables contain the SQLCODE and SQLSTATE from the previous SQL statement.

### Examples

Use a GET DIAGNOSTICS statement to determine how many rows were updated by the previous SQL statement.

```
BEGIN
DECLARE rcount INTEGER;
UPDATE PROJ
SET PRSTAFF = PRSTAFF + 1.5
WHERE DEPTNO = deptnbr;
GET DIAGNOSTICS rcount = ROW_COUNT;
END
```

## GOTO (SQL procedure)

### GOTO statement

The GOTO statement is used to branch to a user-defined label within an SQL procedure.

### Syntax

```
►►GOTO—label————►►
```

### Description

#### *label*

Specifies a labelled statement at which processing is to continue.

The labelled statement and the GOTO statement must be in the same scope. The following rules apply to the scope:

- If the GOTO statement is defined in a compound statement, *label* must be defined inside the same compound statement. *label* cannot be in a nested compound statement.
- If the GOTO statement is defined in a handler, *label* must be defined in the same handler and follow the other scope rules.
- If the GOTO statement is defined outside of a handler, *label* must not be defined within a handler.

If *label* is not defined within a scope that the GOTO statement can reach, an error is returned.

A label name cannot be the same as the name of the SQL procedure in which the label is used.

### Notes

Use the GOTO statement sparingly. Because the GOTO statement interferes with the normal sequence of processing, it makes an SQL procedure more difficult to read and maintain. Before using a GOTO statement, determine whether some other statement, such as an IF statement or LEAVE statement, can be used instead.

### Examples

Use a GOTO statement to transfer control to the end of a compound statement if the value of an SQL variable is less than 600.

```
BEGIN
 DECLARE new_salary DECIMAL(9,2);
 DECLARE service DECIMAL(8,2);
 SELECT SALARY, CURRENT_DATE - HIREDATE
 INTO new_salary, service
 FROM EMP
 WHERE EMPNO = v_empno;
 IF service < 600
 THEN GOTO EXIT1;
 END IF;
 IF rating = 1
 THEN SET new_salary =
 new_salary + (new_salary * .10);
 ELSEIF rating = 2
 THEN SET new_salary =
```

## GOTO (SQL procedure)

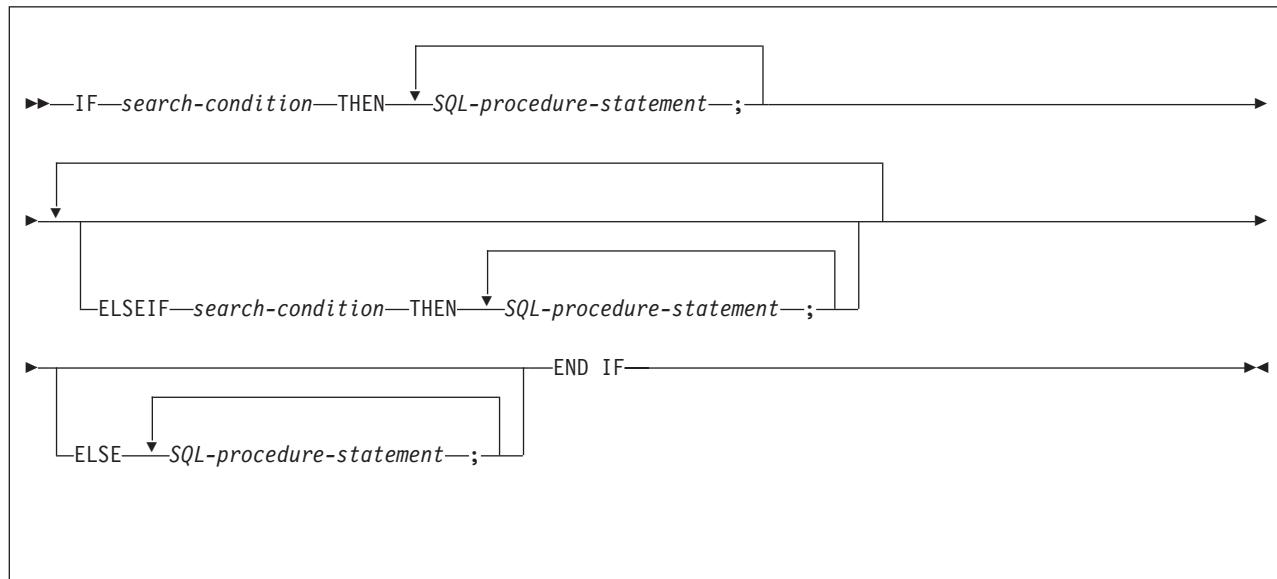
```
 new_salary + (new_salary * .05);
END IF;
UPDATE EMP
SET SALARY = new_salary
WHERE EMPNO = v_empno;
EXIT1: SET return_parm = service;
END
```

## IF (SQL procedure)

### IF statement

The IF statement selects an execution path based on the evaluation of a condition.

### Syntax



### Description

#### *search-condition*

Specifies the condition for which an SQL statement should be invoked. If the condition is unknown or false, processing continues to the next search condition until either a condition is true or processing reaches the ELSE clause.

#### *SQL-procedure-statement*

Specifies the statement to be invoked if the preceding *search-condition* is true. If no *search-condition* evaluates to true, then the *SQL-procedure-statement* following the ELSE keyword is invoked. The statement must be one of the statements listed under “SQL-procedure-statement” on page 944.

### Examples

Assign a value to the SQL variable new\_salary based on the value of SQL variable rating.

```
IF rating = 1
 THEN SET new_salary =
 new_salary + (new_salary * .10);
ELSEIF rating = 2
 THEN SET new_salary =
 new_salary + (new_salary * .05);
ELSE SET new_salary =
 new_salary + (new_salary * .02);
END IF
```

## LEAVE statement

The LEAVE statement transfers program control out of a loop or a compound statement.

### Syntax

```
►►—LEAVE—label—►►
```

### Description

*label*

Specifies the label of the compound statement or loop to exit.

A label name cannot be the same as the name of the SQL procedure in which the label is used.

### Notes

When a LEAVE statement transfers control out of a compound statement, all open cursors in the compound statement, except cursors that are used to return result sets, are closed.

If a LEAVE statement is the only statement in the procedure body, the statement cannot end with a semicolon. Otherwise, the statement must end with a semicolon.

### Examples

Use a LEAVE statement to transfer control out of a LOOP statement when a negative SQLCODE occurs.

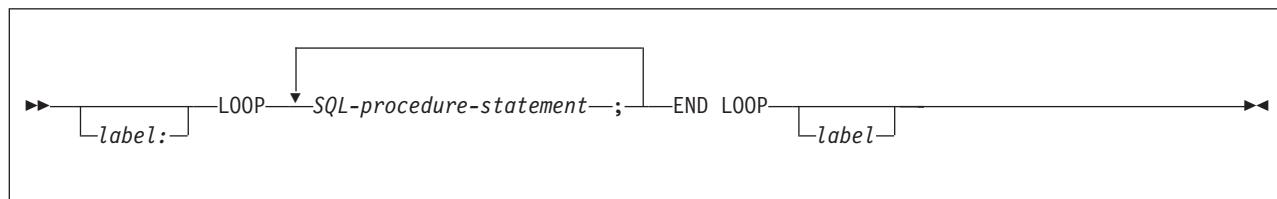
```
ftch_loop: LOOP
 FETCH c1 INTO
 v_firstname, v_midinit,
 v_lastname, v_edlevel, v_salary;
 IF SQLCODE=100 THEN LEAVE ftch_loop;
 END IF;
END LOOP
```

## LOOP (SQL procedure)

### LOOP statement

The LOOP statement executes a statement or group of statements multiple times.

### Syntax



### Description

#### *label*

Specifies the label for the LOOP statement. If the ending label is specified, the beginning label must be specified, and the two must match.

A label name cannot be the same as the name of the SQL procedure in which the label is used.

#### *SQL-procedure-statement*

Specifies the statements to be executed in the loop. The statement must be one of the statements listed under “SQL-procedure-statement” on page 944.

### Notes

If a LOOP statement is the only statement in the procedure body, the statement cannot end with a semicolon. Otherwise, the statement must end with a semicolon.

### Examples

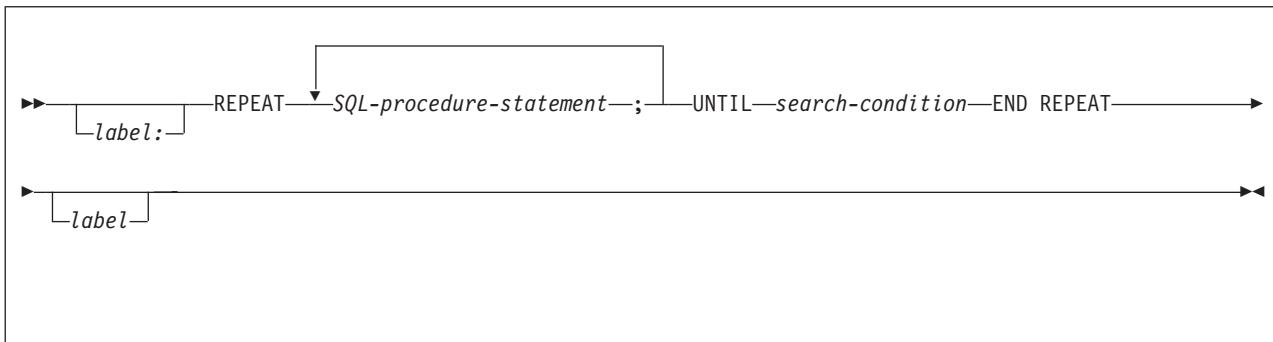
Use a LOOP statement to fetch rows from a table.

```
ftch_loop: LOOP
 FETCH c1 INTO
 v_firstname, v_midinit,
 v_lastname, v_edlevel, v_salary;
 IF SQLCODE<>0 THEN SET badsql=1;
 END IF;
END LOOP
```

## REPEAT statement

The REPEAT statement executes a statement or group of statements until a search condition is true.

### Syntax



### Description

#### *label*

Specifies the label for the REPEAT statement. If the ending label is specified, the beginning label must be specified, and the two must match.

A label name cannot be the same as the name of the SQL procedure in which the label is used.

#### *SQL-procedure-statement*

Specifies the statements to be executed. The statement must be one of the statements listed under “SQL-procedure-statement” on page 944.

#### *search-condition*

Specifies a condition that is evaluated after each execution of the SQL procedure statement. If the condition is true, the SQL procedure statement is not executed again.

### Notes

If a REPEAT statement is the only statement in the procedure body, the statement cannot end with a semicolon. Otherwise, the statement must end with a semicolon.

### Examples

Use a REPEAT statement to fetch rows from a table.

```

fetch_loop:
REPEAT
 FETCH c1 INTO
 v_firstname, v_midinit, v_lastname;
UNTIL
 SQLCODE <> 0
END REPEAT fetch_loop

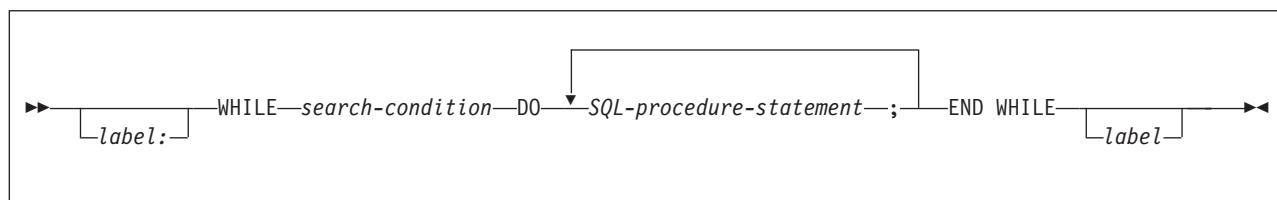
```

## WHILE (SQL procedure)

### WHILE statement

The WHILE statement repeats the execution of a statement or group of statements while a specified condition is true.

### Syntax



### Description

#### *label*

Specifies the label for the WHILE statement. If the ending label is specified, it must be the same as the beginning label.

A label name cannot be the same as the name of the SQL procedure in which the label is used.

#### *search-condition*

Specifies a condition that is evaluated before each execution of the loop. If the condition is true, the SQL procedure statement in the loop is executed.

#### *SQL-procedure-statement*

Specifies the statements to be executed in the loop. The statement must be one of the statements listed under “SQL-procedure-statement” on page 944.

### Notes

If a WHILE statement is the only statement in the procedure body, the statement cannot end with a semicolon. Otherwise, the statement must end with a semicolon.

### Examples

Use a WHILE statement to fetch rows from a table while SQL variable *at\_end*, which indicates whether the end of the table has been reached, is 0.

```
WHILE at_end = 0 DO
 FETCH c1 INTO
 v_firstname, v_midinit,
 v_lastname, v_edlevel, v_salary;
 IF SQLCODE=100 THEN SET at_end=1;
 END IF;
END WHILE
```

## Appendix A. Limits in DB2 for OS/390 and z/OS

System storage limits might preclude the limits specified here. The limit for items not specified below is system storage.

Table 65. Identifier length limits

| Item                                                                                                                                                                                     | Limit    |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------|
| Longest collection ID, correlation name, statement name, or name of an alias, column, cursor, index, table, check constraint, stored procedure, synonym, user-defined function, and view | 18 bytes |
| Longest authorization name, or name of a database, package, plan, referential constraint, schema, storage group, tablespace, or trigger                                                  | 8 bytes  |
| Longest host identifier                                                                                                                                                                  | 64 bytes |
| Longest server name or location identifier                                                                                                                                               | 16 bytes |

Table 66. Numeric limits

| Item                          | Limit                        |
|-------------------------------|------------------------------|
| Smallest SMALLINT value       | -32768                       |
| Largest SMALLINT value        | 32767                        |
| Smallest INTEGER value        | -2147483648                  |
| Largest INTEGER value         | 2147483647                   |
| Smallest REAL value           | About $-7.2 \times 10^{-75}$ |
| Largest REAL value            | About $7.2 \times 10^{-75}$  |
| Smallest positive REAL value  | About $5.4 \times 10^{-79}$  |
| Largest negative REAL value   | About $-5.4 \times 10^{-79}$ |
| Smallest FLOAT value          | About $-7.2 \times 10^{-75}$ |
| Largest FLOAT value           | About $7.2 \times 10^{-75}$  |
| Smallest positive FLOAT value | About $5.4 \times 10^{-79}$  |
| Largest negative FLOAT value  | About $-5.4 \times 10^{-79}$ |
| Smallest DECIMAL value        | $1 - 10^{31}$                |
| Largest DECIMAL value         | $10^{31} - 1$                |
| Largest decimal precision     | 31                           |

I Table 67. String length limits

| Item                                    | Limit                                                                                                                |
|-----------------------------------------|----------------------------------------------------------------------------------------------------------------------|
| Maximum length of CHAR                  | 255 bytes                                                                                                            |
| Maximum length of GRAPHIC               | 127 DBCS characters                                                                                                  |
| Maximum length <sup>40</sup> of VARCHAR | 4046 bytes for 4-KB pages<br>8128 bytes for 8-KB pages<br>16320 bytes for 16-KB pages<br>32704 bytes for 32-KB pages |

## Limits in DB2 for OS/390 and z/OS

*Table 67. String length limits (continued)*

| Item                                              | Limit                                                                                                                                                                                                            |
|---------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Maximum length <sup>40</sup> of VARGRAPHIC        | 4046 bytes (2023 DBCS characters) for 4-KB pages<br>8128 bytes (4064 DBCS characters) for 8-KB pages<br>16320 bytes (8160 DBCS characters for 16-KB pages<br>32704 bytes (16352 DBCS characters) for 32-KB pages |
| Maximum length of CLOB                            | 2 147 483 647 bytes (2 gigabytes - 1 byte)                                                                                                                                                                       |
| Maximum length of DBCLOB                          | 1 073 741 823 DBCS characters                                                                                                                                                                                    |
| Maximum length of BLOB                            | 2 147 483 647 bytes (2 gigabytes - 1 byte)                                                                                                                                                                       |
| Maximum length of a character constant            | 255 bytes                                                                                                                                                                                                        |
| Maximum length of a hexadecimal constant          | 254 digits                                                                                                                                                                                                       |
| Maximum length of a graphic string constant       | 124 DBCS characters                                                                                                                                                                                              |
| Maximum length of a concatenated character string | 2 147 483 647 bytes (2 gigabytes - 1 byte)                                                                                                                                                                       |
| Maximum length of a concatenated graphic string   | 1 073 741 824 DBCS characters                                                                                                                                                                                    |
| Maximum length of a concatenated binary string    | 2 147 483 647 bytes (2 gigabytes - 1 byte)                                                                                                                                                                       |

*Table 68. Datetime limits*

| Item                                      | Limit                      |
|-------------------------------------------|----------------------------|
| Smallest DATE value (shown in ISO format) | 0001-01-01                 |
| Largest DATE value (shown in ISO format)  | 9999-12-31                 |
| Smallest TIME value (shown in ISO format) | 00.00.00                   |
| Largest TIME value (shown in ISO format)  | 24.00.00                   |
| Smallest TIMESTAMP value                  | 0001-01-01-00.00.00.000000 |
| Largest TIMESTAMP value                   | 9999-12-31-24.00.00.000000 |

*Table 69. DB2 limits on SQL statements*

| Item                                                                                                                                                     | Limit                                                                                                                                                           |
|----------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Maximum number of columns in a table or view (the value depends on the complexity of the CREATE VIEW statement) or columns returned by a table function. | 750 or fewer<br>749 if the table is a dependent                                                                                                                 |
| Maximum number of base tables in a view, SELECT, UPDATE, INSERT, or DELETE                                                                               | 225                                                                                                                                                             |
| Maximum row and record sizes for a table                                                                                                                 | See “Maximum record size” on page 676 under CREATE TABLE                                                                                                        |
| Maximum number of volume IDs in a storage group                                                                                                          | 133                                                                                                                                                             |
| Maximum number of partitions in a partitioned table space or partitioned index                                                                           | 64 for table spaces that are not defined with LARGE or a DSSIZE greater than 2G<br>254 for table spaces that are defined with LARGE or a DSSIZE greater than 2G |

40. The maximum length can be achieved only if the column is the only column in the table. Otherwise, the maximum length depends on the amount of space remaining on a page.

41. If the scrollable cursor is read-only, the maximum number is 749 less the number of columns in the ORDER BY that are not in the select list. If the scrollable cursor is not read-only, the maximum number is 747.

Table 69. DB2 limits on SQL statements (continued)

| Item                                                                                                                                                                             | Limit                                                                                                                                                                                               |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Maximum size of a partition (table space or index)                                                                                                                               | For table spaces that are not defined with LARGE or a DSSIZE greater than 2G:<br>4 gigabytes, for 1 to 16 partitions<br>2 gigabytes, for 17 to 32 partitions<br>1 gigabyte, for 33 to 64 partitions |
|                                                                                                                                                                                  | For table spaces that are defined with LARGE:<br>4 gigabytes, for 1 to 254 partitions                                                                                                               |
|                                                                                                                                                                                  | For table spaces that are defined with a DSSIZE greater than 2G:<br>64 gigabytes, for 1 to 254 partitions                                                                                           |
| # Maximum size of a DBRM entry                                                                                                                                                   | 65536 bytes                                                                                                                                                                                         |
| Longest index key                                                                                                                                                                | 255 bytes less the number of key columns that allow nulls.                                                                                                                                          |
| # Maximum number of bytes used in the partitioning of a partitioned index                                                                                                        | 255 (This maximum limit is subject to further limitation, depending on the number of partitions in the table space. The number of partitions * (106 + limit key size) must be less than 65394.)     |
| #                                                                                                                                                                                |                                                                                                                                                                                                     |
| Maximum number of columns in an index key                                                                                                                                        | 64                                                                                                                                                                                                  |
| Maximum number of tables in a FROM clause                                                                                                                                        | 225 or fewer, depending on the complexity of the statement                                                                                                                                          |
| Maximum number of subqueries in a statement                                                                                                                                      | 14                                                                                                                                                                                                  |
| Maximum total length of host and indicator variables pointed to in an SQLDA                                                                                                      | 32767 bytes<br><br>2 147 483 647 bytes (2 gigabytes - 1 byte) for a LOB, subject to the limitations imposed by the application environment and host language                                        |
| Longest host variable used for insert or update                                                                                                                                  | 32704 bytes for a non-LOB<br><br>2 147 483 647 bytes (2 gigabytes - 1 byte) for a LOB, subject to the limitations imposed by the application environment and host language                          |
| # Maximum length of host variables in an application that uses DRDA access                                                                                                       | 32767 bytes<br><br>2 147 483 647 bytes (2 gigabytes - 1 byte) for a LOB, subject to the limitations imposed by the application environment and host language                                        |
| #                                                                                                                                                                                |                                                                                                                                                                                                     |
| # Maximum length of host variables in an application that uses private protocol                                                                                                  | 32704 bytes for a non-LOB<br><br>2 147 483 647 bytes (2 gigabytes - 1 byte) for a LOB, subject to the limitations imposed by the application environment and host language                          |
| #                                                                                                                                                                                |                                                                                                                                                                                                     |
| Longest SQL statement                                                                                                                                                            | 32765 bytes                                                                                                                                                                                         |
| # Maximum number of elements in a select list                                                                                                                                    | 750 or fewer, depending on whether the select list is for the result table of a scrollable cursor <sup>41</sup>                                                                                     |
| #                                                                                                                                                                                |                                                                                                                                                                                                     |
| Maximum number of predicates in a WHERE or HAVING clause                                                                                                                         | 750                                                                                                                                                                                                 |
| Maximum total length of columns of a query operation requiring a sort key (SELECT DISTINCT, ORDER BY, GROUP BY, UNION without the ALL keyword, and the DISTINCT column function) | 4000 bytes                                                                                                                                                                                          |
| Maximum length of a table check constraint                                                                                                                                       | 3800 bytes                                                                                                                                                                                          |

## Limits in DB2 for OS/390 and z/OS

*Table 69. DB2 limits on SQL statements (continued)*

| Item                                                                                                                                   | Limit                                                                                                                                                                  |
|----------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Maximum number of bytes that can be passed in a single parameter of an SQL CALL statement                                              | 32765 bytes for a non-LOB<br>2 147 483 647 bytes (2 gigabytes - 1 byte) for a LOB, subject to the limitations imposed by the application environment and host language |
| Maximum number of stored procedures, triggers, and user-defined functions that an SQL statement can implicitly or explicitly reference | 16 nesting levels                                                                                                                                                      |
| Maximum length of the SQL path                                                                                                         | 254 bytes                                                                                                                                                              |

*Table 70. DB2 system limits*

| Item                                                                 | Limit                                                                                                           |
|----------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------|
| Maximum number of concurrent DB2 or application agents               | Limited by the EDM pool size, buffer pool size, and the amount of storage used by each DB2 or application agent |
| Largest table or table space                                         | 16 terabytes                                                                                                    |
| Largest log space                                                    | $2^{48}$ bytes                                                                                                  |
| Largest active log data set                                          | 4 gigabytes -1                                                                                                  |
| Largest archive log data set                                         | 4 gigabytes -1                                                                                                  |
| Maximum number of active log copies                                  | 2                                                                                                               |
| Maximum number of archive log copies                                 | 2                                                                                                               |
| Maximum number of active log data sets (each copy)                   | 31                                                                                                              |
| Maximum number of archive log volumes (each copy)                    | 1000                                                                                                            |
| Maximum number of databases accessible to an application or end user | Limited by system storage and EDM pool size                                                                     |
| Largest EDM pool                                                     | The installation parameter maximum depends on available space                                                   |
| # Maximum number of databases                                        | 65271                                                                                                           |
| Maximum number of rows per page                                      | 255 for all table spaces except catalog and directory tables spaces, which have a maximum of 127                |
| Maximum simple or segmented data set size                            | 2 gigabytes                                                                                                     |
| Maximum partitioned data set size                                    | See item “maximum size of a partition” in Table 69 on page 966                                                  |
| Maximum LOB data set size                                            | 64 gigabytes                                                                                                    |
| # Largest simple or segmented table space                            | 64 gigabytes                                                                                                    |

---

## Appendix B. Characteristics of SQL statements in DB2 for OS/390 and z/OS

This appendix provides a summary of the actions that are allowed on SQL statements in DB2 for OS/390 and z/OS. It also contains a list of the SQL statements that can be executed in external user-defined functions and stored procedures and in SQL procedures.

---

### Actions allowed on SQL statements

Table 71 shows whether a specific DB2 statement can be executed, prepared interactively or dynamically, or processed by the requester, the server, or the precompiler. The letter Y means yes.

Table 71. Actions allowed on SQL statements in DB2 for OS/390 and z/OS

| SQL statement                             | Executable | Interactively<br>or<br>dynamically<br>prepared | Processed by |                |   |
|-------------------------------------------|------------|------------------------------------------------|--------------|----------------|---|
|                                           |            | Requesting<br>system                           | Server       | Precompiler    |   |
| ALLOCATE CURSOR                           | Y          | Y <sup>1</sup>                                 | Y            |                |   |
| ALTER                                     | Y          | Y <sup>2</sup>                                 |              | Y              |   |
| ASSOCIATE LOCATORS                        | Y          | Y <sup>1</sup>                                 | Y            |                |   |
| BEGIN DECLARE SECTION                     |            |                                                |              |                | Y |
| CALL                                      | Y          | Y <sup>3</sup>                                 |              | Y              |   |
| CLOSE                                     | Y          |                                                |              | Y              |   |
| COMMENT ON                                | Y          | Y                                              |              | Y              |   |
| COMMIT                                    | Y          | Y                                              |              | Y <sup>8</sup> |   |
| CONNECT (Type 1 and Type 2)               | Y          |                                                | Y            |                |   |
| CREATE                                    | Y          | Y <sup>2</sup>                                 |              | Y              |   |
| DECLARE CURSOR                            |            |                                                |              |                | Y |
| DECLARE GLOBAL<br>TEMPORARY TABLE         | Y          | Y                                              |              | Y              |   |
| DECLARE STATEMENT                         |            |                                                |              |                | Y |
| DECLARE TABLE                             |            |                                                |              |                | Y |
| DELETE                                    | Y          | Y                                              |              | Y              |   |
| I DESCRIBE prepared statement or<br>table | Y          |                                                |              | Y              |   |
| DESCRIBE CURSOR                           | Y          |                                                | Y            |                |   |
| DESCRIBE INPUT                            | Y          |                                                |              | Y              |   |
| DESCRIBE PROCEDURE                        | Y          |                                                | Y            |                |   |
| DROP                                      | Y          | Y <sup>2</sup>                                 |              | Y              |   |
| END DECLARE SECTION                       |            |                                                |              |                | Y |
| EXECUTE                                   | Y          |                                                |              | Y              |   |
| EXECUTE IMMEDIATE                         | Y          |                                                |              | Y              |   |
| EXPLAIN                                   | Y          | Y                                              |              | Y              |   |
| FETCH                                     | Y          |                                                |              | Y              |   |

## Characteristics of SQL statements in DB2 for OS/390 and z/OS

Table 71. Actions allowed on SQL statements in DB2 for OS/390 and z/OS (continued)

| SQL statement                                                  | Executable | Interactively or dynamically prepared | Processed by      |                |             |
|----------------------------------------------------------------|------------|---------------------------------------|-------------------|----------------|-------------|
|                                                                |            |                                       | Requesting system | Server         | Precompiler |
| FREE LOCATOR                                                   | Y          | Y <sup>1</sup>                        |                   | Y              |             |
| GRANT                                                          | Y          | Y <sup>2</sup>                        |                   | Y              |             |
| HOLD LOCATOR                                                   | Y          | Y <sup>1</sup>                        |                   | Y              |             |
| INCLUDE                                                        |            |                                       |                   |                | Y           |
| INSERT                                                         | Y          | Y                                     |                   | Y              |             |
| LABEL ON                                                       | Y          | Y                                     |                   | Y              |             |
| LOCK TABLE                                                     | Y          | Y                                     |                   | Y              |             |
| OPEN                                                           | Y          |                                       |                   | Y              |             |
| PREPARE                                                        | Y          |                                       |                   | Y <sup>4</sup> |             |
| RELEASE connection                                             | Y          |                                       | Y                 |                |             |
| RELEASE SAVEPOINT                                              | Y          | Y                                     |                   | Y              |             |
| RENAME                                                         | Y          | Y <sup>2</sup>                        |                   | Y              |             |
| REVOKE                                                         | Y          | Y <sup>2</sup>                        |                   | Y              |             |
| ROLLBACK                                                       | Y          | Y                                     |                   | Y <sup>8</sup> |             |
| SAVEPOINT                                                      | Y          | Y                                     |                   | Y              |             |
| SELECT INTO                                                    | Y          |                                       |                   | Y              |             |
| SET CONNECTION                                                 | Y          |                                       | Y                 |                |             |
| SET CURRENT APPLICATION                                        | Y          |                                       | Y                 |                |             |
| ENCODING SCHEME                                                |            |                                       |                   |                |             |
| SET CURRENT DEGREE                                             | Y          | Y                                     |                   | Y              |             |
| SET CURRENT LC_CTYPE                                           | Y          | Y                                     |                   | Y              |             |
| SET CURRENT OPTIMIZATION HINT                                  | Y          | Y                                     |                   | Y              |             |
| SET CURRENT PACKAGESET                                         | Y          |                                       | Y                 |                |             |
| SET CURRENT PRECISION                                          | Y          | Y                                     |                   | Y              |             |
| SET CURRENT RULES                                              | Y          | Y                                     |                   | Y              |             |
| SET CURRENT SQLID <sup>5</sup>                                 | Y          | Y                                     |                   | Y              |             |
| SET <i>host-variable</i> = CURRENT APPLICATION ENCODING SCHEME | Y          | Y                                     | Y                 |                |             |
| SET <i>host-variable</i> = CURRENT DATE                        | Y          |                                       |                   | Y              |             |
| SET <i>host-variable</i> = CURRENT DEGREE                      | Y          |                                       |                   | Y              |             |
| SET <i>host-variable</i> = CURRENT MEMBER                      | Y          |                                       |                   | Y              |             |
| SET <i>host-variable</i> = CURRENT PACKAGESET                  | Y          |                                       | Y                 |                |             |
| SET <i>host-variable</i> = CURRENT QUERY OPTIMIZATION LEVEL    | Y          |                                       |                   | Y              |             |

## Characteristics of SQL statements in DB2 for OS/390 and z/OS

*Table 71. Actions allowed on SQL statements in DB2 for OS/390 and z/OS (continued)*

| SQL statement                                      | Executable | Interactively<br>or<br>dynamically<br>prepared | Processed by         |        |             |
|----------------------------------------------------|------------|------------------------------------------------|----------------------|--------|-------------|
|                                                    |            |                                                | Requesting<br>system | Server | Precompiler |
| SET <i>host-variable</i> = CURRENT SERVER          | Y          |                                                | Y                    |        |             |
| SET <i>host-variable</i> = CURRENT SQLID           | Y          |                                                |                      | Y      |             |
| SET <i>host-variable</i> = CURRENT TIME            | Y          |                                                |                      | Y      |             |
| SET <i>host-variable</i> = CURRENT TIMESTAMP       | Y          |                                                |                      | Y      |             |
| SET <i>host-variable</i> = CURRENT TIMEZONE        | Y          |                                                |                      | Y      |             |
| SET <i>host-variable</i> = PATH                    | Y          |                                                |                      | Y      |             |
| SET PATH                                           | Y          | Y                                              |                      | Y      |             |
| SET <i>transition-variable</i> = CURRENT DATE      | Y          |                                                |                      | Y      |             |
| SET <i>transition-variable</i> = CURRENT DEGREE    | Y          |                                                |                      | Y      |             |
| SET <i>transition-variable</i> = CURRENT QUERY     | Y          |                                                |                      | Y      |             |
| OPTIMIZATION LEVEL                                 |            |                                                |                      |        |             |
| SET <i>transition-variable</i> = CURRENT SQLID     | Y          |                                                |                      | Y      |             |
| SET <i>transition-variable</i> = CURRENT TIME      | Y          |                                                |                      | Y      |             |
| SET <i>transition-variable</i> = CURRENT TIMESTAMP | Y          |                                                |                      | Y      |             |
| SET <i>transition-variable</i> = CURRENT TIMEZONE  | Y          |                                                |                      | Y      |             |
| SET <i>transition-variable</i> = PATH              | Y          |                                                |                      | Y      |             |
| SIGNAL SQLSTATE <sup>6</sup>                       | Y          |                                                |                      | Y      |             |
| UPDATE                                             | Y          | Y                                              |                      | Y      |             |
| VALUES <sup>6</sup>                                | Y          |                                                |                      | Y      |             |
| VALUES INTO <sup>7</sup>                           | Y          |                                                |                      | Y      |             |
| WHENEVER                                           |            |                                                |                      |        | Y           |

## Characteristics of SQL statements in DB2 for OS/390 and z/OS

Table 71. Actions allowed on SQL statements in DB2 for OS/390 and z/OS (continued)

| SQL statement | Executable                                                                                                                                                                                                                                                                    | Interactively<br>or<br>dynamically<br>prepared | Processed by |             |  |
|---------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------|--------------|-------------|--|
|               |                                                                                                                                                                                                                                                                               | Requesting<br>system                           | Server       | Precompiler |  |
| <b>Notes:</b> |                                                                                                                                                                                                                                                                               |                                                |              |             |  |
| 1.            | The statement can be dynamically prepared. It cannot be prepared interactively.                                                                                                                                                                                               |                                                |              |             |  |
| 2.            | The statement can be dynamically prepared only if DYNAMICRULES run behavior is implicitly or explicitly specified.                                                                                                                                                            |                                                |              |             |  |
| 3.            | The statement can be dynamically prepared, but only from an ODBC or CLI driver that supports dynamic CALL statements.                                                                                                                                                         |                                                |              |             |  |
| 4.            | The requesting system processes the PREPARE statement when the statement being prepared is ALLOCATE CURSOR or ASSOCIATE LOCATORS.                                                                                                                                             |                                                |              |             |  |
| 5.            | The value to which special register CURRENT SQLID is set is used as the SQL authorization ID and the implicit qualifier for dynamic SQL statements only when DYNAMICRULES run behavior is in effect. The CURRENT SQLID value is ignored for the other DYNAMICRULES behaviors. |                                                |              |             |  |
| 6.            | This statement can only be used in the triggered action of a trigger.                                                                                                                                                                                                         |                                                |              |             |  |
| 7.            | Local special registers can be referenced in a VALUES INTO statement if it results in the assignment of a single host-variable, not if it results in setting more than one value.                                                                                             |                                                |              |             |  |
| # 8.          | Some processing also occurs at the requester.                                                                                                                                                                                                                                 |                                                |              |             |  |

## SQL statements allowed in external functions and stored procedures

Table 72 shows which SQL statements in an external stored procedure or in an external user-defined function can execute. Whether the statements can be executed depends on the level of SQL data access with which the stored procedure or external function is defined (NO SQL, CONTAINS SQL, READS SQL DATA, or MODIFIES SQL DATA). The letter Y means yes.

In general, if an executable SQL statement is encountered in a stored procedure or function defined as NO SQL, SQLSTATE 38001 is returned. If the routine is defined to allow some level of SQL access, SQL statements that are not supported in any context return SQLSTATE 38003. SQL statements not allowed for routines defined as CONTAINS SQL return SQLSTATE 38004, and SQL statements not allowed for READS SQL DATA return SQL STATE 38002.

Table 72. SQL statements in external user-defined functions and stored procedures

| SQL statement               | Level of SQL access |                 |                   |                      |
|-----------------------------|---------------------|-----------------|-------------------|----------------------|
|                             | NO SQL              | CONTAINS<br>SQL | READS SQL<br>DATA | MODIFIES<br>SQL DATA |
| ALLOCATE CURSOR             |                     |                 | Y                 | Y                    |
| ALTER                       |                     |                 |                   | Y                    |
| ASSOCIATE LOCATORS          |                     |                 | Y                 | Y                    |
| BEGIN DECLARE SECTION       | Y <sup>1</sup>      | Y               | Y                 | Y                    |
| CALL                        |                     | Y <sup>2</sup>  | Y <sup>2</sup>    | Y <sup>2</sup>       |
| CLOSE                       |                     |                 | Y                 | Y                    |
| COMMENT ON                  |                     |                 |                   | Y                    |
| COMMIT <sup>3</sup>         |                     | Y               | Y                 | Y                    |
| CONNECT (Type 1 and Type 2) |                     | Y               | Y                 | Y                    |

## Characteristics of SQL statements in DB2 for OS/390 and z/OS

*Table 72. SQL statements in external user-defined functions and stored procedures (continued)*

| SQL statement                            | Level of SQL access |                |                |                   |
|------------------------------------------|---------------------|----------------|----------------|-------------------|
|                                          | NO SQL              | CONTAINS SQL   | READS SQL DATA | MODIFIES SQL DATA |
| CREATE                                   |                     |                |                | Y                 |
| DECLARE CURSOR                           | Y <sup>1</sup>      | Y              | Y              | Y                 |
| DECLARE GLOBAL TEMPORARY TABLE           |                     |                |                | Y                 |
| DECLARE STATEMENT                        | Y <sup>1</sup>      | Y              | Y              | Y                 |
| DECLARE TABLE                            | Y <sup>1</sup>      | Y              | Y              | Y                 |
| DELETE                                   |                     |                |                | Y                 |
| DESCRIBE                                 |                     |                | Y              | Y                 |
| DESCRIBE CURSOR                          |                     |                | Y              | Y                 |
| DESCRIBE INPUT                           |                     |                | Y              | Y                 |
| DESCRIBE PROCEDURE                       |                     |                | Y              | Y                 |
| DROP                                     |                     |                |                | Y                 |
| END DECLARE SECTION                      | Y <sup>1</sup>      | Y              | Y              | Y                 |
| EXECUTE                                  |                     | Y <sup>4</sup> | Y <sup>4</sup> | Y                 |
| EXECUTE IMMEDIATE                        |                     | Y <sup>4</sup> | Y <sup>4</sup> | Y                 |
| EXPLAIN                                  |                     |                |                | Y                 |
| FETCH                                    |                     |                | Y              | Y                 |
| FREE LOCATOR                             |                     | Y              | Y              | Y                 |
| GRANT                                    |                     |                |                | Y                 |
| HOLD LOCATOR                             |                     | Y              | Y              | Y                 |
| INCLUDE                                  | Y <sup>1</sup>      | Y              | Y              | Y                 |
| INSERT                                   |                     |                |                | Y                 |
| LABEL ON                                 |                     |                |                | Y                 |
| LOCK TABLE                               |                     | Y              | Y              | Y                 |
| OPEN                                     |                     |                | Y              | Y                 |
| PREPARE                                  |                     | Y              | Y              | Y                 |
| RELEASE connection                       |                     | Y              | Y              | Y                 |
| RELEASE SAVEPOINT <sup>6</sup>           |                     |                |                | Y                 |
| REVOKE                                   |                     |                |                | Y                 |
| ROLLBACK <sup>6, 7, 8</sup>              |                     | Y              | Y              | Y                 |
| ROLLBACK TO SAVEPOINT <sup>6, 7, 8</sup> |                     |                |                | Y                 |
| SAVEPOINT <sup>6</sup>                   |                     |                |                | Y                 |
| SELECT                                   |                     |                | Y              | Y                 |
| SELECT INTO                              |                     |                | Y              | Y                 |
| SET CONNECTION                           |                     | Y              | Y              | Y                 |
| SET host-variable Assignment             |                     | Y <sup>5</sup> | Y              | Y                 |
| SET special register                     |                     | Y              | Y              | Y                 |

## Characteristics of SQL statements in DB2 for OS/390 and z/OS

Table 72. SQL statements in external user-defined functions and stored procedures (continued)

| SQL statement                      | Level of SQL access |                |                |                   |
|------------------------------------|---------------------|----------------|----------------|-------------------|
|                                    | NO SQL              | CONTAINS SQL   | READS SQL DATA | MODIFIES SQL DATA |
| SET transition-variable Assignment |                     | Y <sup>5</sup> | Y              | Y                 |
| SIGNAL SQLSTATE                    |                     | Y              | Y              | Y                 |
| UPDATE                             |                     |                |                | Y                 |
| VALUES                             |                     |                | Y              | Y                 |
| VALUES INTO                        |                     | Y <sup>5</sup> | Y              | Y                 |
| WHENEVER                           | Y <sup>1</sup>      | Y              | Y              | Y                 |

### Notes:

1. Although the SQL option implies that no SQL statements can be specified, non-executable statements are not restricted.
2. The stored procedure that is called must have the same or more restrictive level of SQL data access than the current level in effect. For example, a routine defined as MODIFIES SQL DATA can call a stored procedure defined as MODIFIES SQL DATA, READS SQL DATA, or CONTAINS SQL. A routine defined as CONTAINS SQL can only call a procedure defined as CONTAINS SQL.
3. The COMMIT statement cannot be executed in a user-defined function. The COMMIT statement cannot be executed in a stored procedure if the procedure is in the calling chain of a user-defined function or trigger.
4. The statement specified for the EXECUTE statement must be a statement that is allowed for the particular level of SQL data access in effect. For example, if the level in effect is READS SQL DATA, the statement must not be an INSERT, UPDATE, or DELETE.
5. The statement is supported only if it does not contain a subquery or query-expression.
6. RELEASE SAVEPOINT, SAVEPOINT, and ROLLBACK (with the TO SAVEPOINT clause) cannot be executed from a user-defined function.
7. If the ROLLBACK statement (without the TO SAVEPOINT clause) is executed in a user-defined function, an error is returned to the calling program, and the application is placed in a *must rollback* state.
8. The ROLLBACK statement (without the TO SAVEPOINT clause) cannot be executed in a stored procedure if the procedure is in the calling chain of a user-defined function or trigger.

## SQL statements allowed in SQL procedures

Table 73 on page 975 lists the statements that are valid in an SQL procedure body, in addition to SQL procedure statements. The table lists the statements that can be used as the only statement in the SQL procedure and as the statements that can be nested in a compound statement. An SQL statement can be executed in an SQL procedure depending on whether MODIFIES SQL DATA, CONTAINS SQL, or READS SQL DATA is specified in the stored procedure definition. See Table 72 on page 972 for a list of SQL statements that can be executed for each of these parameter values.

## Characteristics of SQL statements in DB2 for OS/390 and z/OS

*Table 73. Valid SQL statements in an SQL procedure body*

| SQL statement                        | SQL statement is...                 |                                |
|--------------------------------------|-------------------------------------|--------------------------------|
|                                      | The only statement in the procedure | Nested in a compound statement |
| ALLOCATE CURSOR                      |                                     | Y                              |
| ALTER DATABASE                       | Y                                   | Y                              |
| ALTER FUNCTION                       | Y                                   | Y                              |
| ALTER INDEX                          | Y                                   | Y                              |
| ALTER PROCEDURE                      | Y                                   | Y                              |
| ALTER STOGROUP                       | Y                                   | Y                              |
| ALTER TABLE                          | Y                                   | Y                              |
| ALTER TABLESPACE                     | Y                                   | Y                              |
| ASSOCIATE LOCATORS                   |                                     | Y                              |
| BEGIN DECLARE SECTION                |                                     |                                |
| CALL                                 |                                     | Y                              |
| CLOSE                                |                                     | Y                              |
| COMMENT ON                           | Y                                   | Y                              |
| COMMIT <sup>1</sup>                  | Y                                   | Y                              |
| CONNECT (Type 1 and Type 2)          | Y                                   | Y                              |
| CREATE ALIAS                         | Y                                   | Y                              |
| CREATE DATABASE                      | Y                                   | Y                              |
| CREATE DISTINCT TYPE                 | Y                                   | Y                              |
| CREATE FUNCTION <sup>2</sup>         | Y                                   | Y                              |
| CREATE GLOBAL TEMPORARY TABLE        | Y                                   | Y                              |
| CREATE INDEX                         | Y                                   | Y                              |
| CREATE PROCEDURE <sup>2</sup>        | Y                                   | Y                              |
| CREATE STOGROUP                      | Y                                   | Y                              |
| CREATE SYNONYM                       | Y                                   | Y                              |
| CREATE TABLE                         | Y                                   | Y                              |
| CREATE TABLESPACE                    | Y                                   | Y                              |
| CREATE TRIGGER                       |                                     |                                |
| CREATE VIEW                          | Y                                   | Y                              |
| DECLARE CURSOR                       |                                     | Y                              |
| DECLARE GLOBAL TEMPORARY TABLE       | Y                                   | Y                              |
| DECLARE STATEMENT                    |                                     |                                |
| DECLARE TABLE                        |                                     |                                |
| DELETE                               | Y                                   | Y                              |
| DESCRIBE prepared statement or table |                                     |                                |
| DESCRIBE CURSOR                      |                                     |                                |
| DESCRIBE INPUT                       |                                     |                                |
| DESCRIBE PROCEDURE                   |                                     |                                |

## Characteristics of SQL statements in DB2 for OS/390 and z/OS

*Table 73. Valid SQL statements in an SQL procedure body (continued)*

| SQL statement                                   | The only statement in the procedure | Nested in a compound statement |
|-------------------------------------------------|-------------------------------------|--------------------------------|
| DROP                                            | Y                                   | Y                              |
| END DECLARE SECTION                             |                                     |                                |
| EXECUTE                                         |                                     | Y                              |
| EXECUTE IMMEDIATE                               | Y                                   | Y                              |
| EXPLAIN                                         |                                     |                                |
| FETCH                                           |                                     | Y                              |
| FREE LOCATOR                                    |                                     |                                |
| GRANT                                           | Y                                   | Y                              |
| HOLD LOCATOR                                    |                                     |                                |
| INCLUDE                                         |                                     |                                |
| INSERT                                          | Y                                   | Y                              |
| LABEL ON                                        | Y                                   | Y                              |
| LOCK TABLE                                      | Y                                   | Y                              |
| OPEN                                            |                                     | Y                              |
| PREPARE FROM                                    |                                     | Y                              |
| RELEASE connection                              | Y                                   | Y                              |
| RELEASE SAVEPOINT                               | Y                                   | Y                              |
| RENAME                                          | Y                                   | Y                              |
| REVOKE                                          | Y                                   | Y                              |
| ROLLBACK <sup>1</sup>                           | Y                                   | Y                              |
| ROLLBACK TO SAVEPOINT <sup>1</sup>              | Y                                   | Y                              |
| SAVEPOINT                                       | Y                                   | Y                              |
| SELECT                                          |                                     |                                |
| SELECT INTO                                     | Y                                   | Y                              |
| SET CONNECTION                                  | Y                                   | Y                              |
| SET host-variable Assignment <sup>3</sup>       |                                     |                                |
| SET special register <sup>3</sup>               | Y                                   | Y                              |
| SET transition-variable Assignment <sup>3</sup> |                                     |                                |
| SIGNAL SQLSTATE                                 |                                     |                                |
| UPDATE                                          | Y                                   | Y                              |
| VALUES                                          |                                     |                                |
| VALUES INTO                                     | Y                                   | Y                              |
| WHENEVER                                        |                                     |                                |

## Characteristics of SQL statements in DB2 for OS/390 and z/OS

Table 73. Valid SQL statements in an SQL procedure body (continued)

| SQL statement | SQL statement is...                                                                                                                                                                                                                                                                   |
|---------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|               | The only statement in the procedure                                                                                                                                                                                                                                                   |
|               | Nested in a compound statement                                                                                                                                                                                                                                                        |
| <b>Notes:</b> |                                                                                                                                                                                                                                                                                       |
| 1.            | The COMMIT statement and the ROLLBACK statement (without the TO SAVEPOINT clause) cannot be executed in a stored procedure if the procedure is in the calling chain of a user-defined function or trigger                                                                             |
| 2.            | CREATE FUNCTION with LANGUAGE SQL (specified either implicitly or explicitly) and CREATE PROCEDURE with LANGUAGE SQL are not allowed within the body of an SQL procedure.                                                                                                             |
| 3.            | SET host-variable assignment, SET transition-variable assignment, and SET special register are the SQL SET statements that are described in Chapter 5, "Statements," on page 377, not the SQL procedure assignment statement that is described in "assignment-statement" on page 946. |

## Characteristics of SQL statements in DB2 for OS/390 and z/OS

## Appendix C. SQLCA and SQLDA

### SQL communication area (SQLCA)

An SQLCA is a structure or collection of variables that is updated after each SQL statement executes. An application program that contains executable SQL statements must provide exactly one SQLCA. There are two exceptions:

- A program that is precompiled with the STDSQL(YES) option must not provide an SQLCA
- In some cases (as discussed below in In Fortran), a Fortran program must provide more than one SQLCA.

In all host languages except REXX, the SQL INCLUDE statement can be used to provide the declaration of the SQLCA.

*In COBOL and assembler:* The name of the storage area must be SQLCA.

*In PL/I, and C:* The name of the structure must be SQLCA. Every executable SQL statement must be within the scope of its declaration.

Unless noted otherwise, *C* is used to represent C/370™ and C/C++ for MVS/ESA programming languages.

*In Fortran:* The name of the COMMON area for the INTEGER variables of the SQLCA must be SQLCA1; the name of the COMMON area for the CHARACTER variables must be SQLCA2. An SQLCA definition is required for every subprogram that contains SQL statements. One is also needed for the main program if it contains SQL statements.

*In REXX:* DB2 generates the SQLCA automatically. A REXX procedure cannot use the INCLUDE statement. The REXX SQLCA has a somewhat different format from SQLCAs for the other languages. See “The REXX SQLCA” on page 985 for more information on the REXX SQLCA.

### Description of fields

The names in the following table are those provided by the SQL INCLUDE statement. For the most part, COBOL, C, PL/I, and assembler use the same names, and Fortran names are different. However, there is one instance where C, PL/I, and assembler names differ from COBOL.

Table 74. Fields of SQLCA

| assembler,<br>COBOL, or<br>PL/I Name | C<br>Name | Fortran<br>Name | Data<br>type | Purpose                                                          |
|--------------------------------------|-----------|-----------------|--------------|------------------------------------------------------------------|
| SQLCAID                              | sqlcaid   | Not used.       | CHAR(8)      | An “eye catcher” for storage dumps, containing the text ‘SQLCA’. |
| SQLCABC                              | sqlcbc    | Not used.       | INTEGER      | Contains the length of the SQLCA: 136.                           |

## SQLCA

Table 74. Fields of SQLCA (continued)

| assembler,<br>COBOL, or<br>PL/I Name | C<br>Name                | Fortran<br>Name | Data<br>type | Purpose                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
|--------------------------------------|--------------------------|-----------------|--------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| SQLCODE<br>(See note 1)              | SQLCODE                  | SQLCOD          | INTEGER      | Contains the SQL return code. (See note 2)<br><br><b>Code Means</b><br><b>0</b> Successful execution (though there might have been warning messages).<br><b>positive</b> Successful execution, but with a warning condition or other information.<br><b>negative</b> Error condition.                                                                                                                                                                                                     |
| SQLERRML<br>(See note 3)             | sqlerrml<br>(See note 3) | SQLTXL          | SMALLINT     | Length indicator for SQLERRMC, in the range 0 through 70. 0 means that the value of SQLERRMC is not pertinent.                                                                                                                                                                                                                                                                                                                                                                            |
| SQLERRMC<br>(See note 3)             | sqlerrmc<br>(See note 3) | SQLTXT          | VARCHAR(70)  | Contains one or more tokens, separated by X'FF', that are substituted for variables in the descriptions of error conditions.                                                                                                                                                                                                                                                                                                                                                              |
| SQLERRP                              | sqlerrp                  | SQLERP          | CHAR(8)      | Provides a product signature and, in the case of an error, diagnostic information such as the name of the module that detected the error. In all cases, the first three characters are 'DSN' for DB2 for OS/390 and z/OS.                                                                                                                                                                                                                                                                 |
| SQLERRD(1)                           | sqlerrd[0]               | SQLERR(1)       | INTEGER      | Contains the number of rows in a result table when the cursor position is after the last row (that is, when SQLCODE is equal to +100).<br><br>SQLERRD(1) can also contain an internal error code.                                                                                                                                                                                                                                                                                         |
| SQLERRD(2)                           | sqlerrd[1]               | SQLERR(2)       | INTEGER      | Contains the number of rows in a result table when the cursor position is after the last row (that is, when SQLCODE is equal to +100).<br><br>SQLERRD(2) can also contain an internal error code.                                                                                                                                                                                                                                                                                         |
| SQLERRD(3)                           | sqlerrd[2]               | SQLERR(3)       | INTEGER      | Contains the number of rows affected after INSERT, UPDATE, and DELETE (but not rows deleted as a result of CASCADE delete). Set to 0 if the SQL statement fails, indicating that all changes made in executing the statement were canceled. Set to -1 for a mass delete from a table in a segmented table space or for a table created by a CREATE GLOBAL TEMPORARY TABLE statement.<br><br>For SQLCODES -911 and -913, SQLERRD(3) contains the reason code for the timeout or deadlock . |

Table 74. Fields of SQLCA (continued)

| assembler,<br>COBOL, or<br>PL/I Name | C<br>Name  | Fortran<br>Name | Data<br>type | Purpose                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
|--------------------------------------|------------|-----------------|--------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| SQLERRD(4)                           | sqlerrd[3] | SQLERR(4)       | INTEGER      | Generally, contains timerons, a short floating-point value that indicates a rough relative estimate of resources required (See note 4). It does not reflect an estimate of the time required. When preparing a dynamically defined SQL statement, you can use this field as an indicator of the relative cost of the prepared SQL statement. For a particular statement, this number can vary with changes to the statistics in the catalog. It is also subject to change between releases of DB2 for OS/390 and z/OS. |
| SQLERRD(5)                           | sqlerrd[4] | SQLERR(5)       | INTEGER      | Contains the position or column of a syntax error for a PREPARE or EXECUTE IMMEDIATE statement.                                                                                                                                                                                                                                                                                                                                                                                                                        |
| SQLERRD(6)                           | sqlerrd[5] | SQLERR(6)       | INTEGER      | Contains an internal error code.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| SQLWARN0                             | SQLWARN0   | SQLWRN(0)       | CHAR(1)      | Contains a blank if no other indicator is set to a warning condition (that is, no other indicator contains a W or Z). Contains a W if at least one other indicator contains a W or Z.                                                                                                                                                                                                                                                                                                                                  |
| SQLWARN1                             | SQLWARN1   | SQLWRN(1)       | CHAR(1)      | Contains a W if the value of a string column was truncated when assigned to a host variable. Contains an N for non-scrollable cursors and S for scrollable cursors after the OPEN CURSOR or ALLOCATE CURSOR statement. If subsystem parameter DISABSCL is set to YES, the field will not be set to N for non-scrollable cursors.                                                                                                                                                                                       |
| SQLWARN2                             | SQLWARN2   | SQLWRN(2)       | CHAR(1)      | Contains a W if null values were eliminated from the argument of a column function; not necessarily set to W for the MIN function because its results are not dependent on the elimination of null values.                                                                                                                                                                                                                                                                                                             |
| SQLWARN3                             | SQLWARN3   | SQLWRN(3)       | CHAR(1)      | Contains a W if the number of result columns is larger than the number of host variables. Contains a Z if fewer locators were provided in the ASSOCIATE LOCATORS statement than the stored procedure returned.                                                                                                                                                                                                                                                                                                         |
| SQLWARN4                             | SQLWARN4   | SQLWRN(4)       | CHAR(1)      | Contains a W if a prepared UPDATE or DELETE statement does not include a WHERE clause. For a scrollable cursor, contains an I for insensitive cursors and S for sensitive static cursors after the OPEN CURSOR or ALLOCATE CURSOR statement; blank if cursor is not scrollable. If subsystem parameter DISABSCL is set to YES, the field will not be set to N for non-scrollable cursors.                                                                                                                              |

## SQLCA

| *Table 74. Fields of SQLCA (continued)*

| assembler,<br>COBOL, or<br>PL/I Name | C<br>Name | Fortran<br>Name | Data<br>type | Purpose                                                                                                                                                                                                                                                                                                                                                                                                                           |
|--------------------------------------|-----------|-----------------|--------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| SQLWARN5                             | SQLWARN5  | SQLWRN(5)       | CHAR(1)      | Contains a W if the SQL statement was not executed because it is not a valid SQL statement in DB2 for OS/390 and z/OS.<br><br>Contains a character value of 1 (read only), 2 (read and delete), or 4 (read, delete, and update) to reflect capability of the cursor after the OPEN CURSOR or ALLOCATE CURSOR statement. If subsystem parameter DISABSCL is set to YES, the field will not be set to N for non-scrollable cursors. |
| #                                    | #         | #               |              |                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| SQLWARN6                             | SQLWARN6  | SQLWRN(6)       | CHAR(1)      | Contains a W if the addition of a month or year duration to a DATE or TIMESTAMP value results in an invalid day (for example, June 31). Indicates that the value of the day was changed to the last day of the month to make the result valid.                                                                                                                                                                                    |
| SQLWARN7                             | SQLWARN7  | SQLWRN(7)       | CHAR(1)      | Contains a W if one or more nonzero digits were eliminated from the fractional part of a number used as the operand of a decimal multiply or divide operation.                                                                                                                                                                                                                                                                    |
| SQLWARN8                             | SQLWARN8  | SQLWRX(1)       | CHAR(1)      | Contains a W if a character that could not be converted was replaced with a substitute character.                                                                                                                                                                                                                                                                                                                                 |
| SQLWARN9                             | SQLWARN9  | SQLWRX(2)       | CHAR(1)      | Contains a W if arithmetic exceptions were ignored during COUNT or COUNT_BIG processing. Contains a Z if the stored procedure returned multiple result sets.                                                                                                                                                                                                                                                                      |
| SQLWARNA                             | SQLWARNA  | SQLWRX(3)       | CHAR(1)      | Contains a W if at least one character field of the SQLCA or the SQLDA names or labels is invalid due to a character conversion error.                                                                                                                                                                                                                                                                                            |
| SQLSTATE                             | sqlstate  | SQLSTT          | CHAR(5)      | Contains a return code for the outcome of the most recent execution of an SQL statement (See note 5).                                                                                                                                                                                                                                                                                                                             |

| **Note:**

1. With the precompiler option STDSQL(YES) in effect, SQLCODE is replaced by SQLCADE in SQLCA.
2. For the specific meanings of SQL return codes, see Part 1 of *DB2 Messages and Codes*.
3. In COBOL, SQLERRM includes SQLERRML and SQLERRMC. In PL/I and C, the varying-length string SQLERRM is equivalent to SQLERRML prefixed to SQLERRMC. In assembler, the storage area SQLERRM is equivalent to SQLERRML and SQLERRMC. See the examples for the various host languages in "The included SQLCA."
4. The use of timerons may require special handling because they are floating-point values in an INTEGER array. In PL/I, for example, you could first copy the value into a BIN FIXED(31) based variable that coincides with a BIN FLOAT(24) variable.
5. For a description of SQLSTATE values, see Appendix C of *DB2 Messages and Codes*.

## The included SQLCA

| The description of the SQLCA that is given by INCLUDE SQLCA is shown for each of the host languages.

**assembler:**

```

SQLCA DS 0F
SQLCAID DS CL8 ID
SQLCABC DS F BYTE COUNT
SQLCODE DS F RETURN CODE
SQLERRM DS H,CL70 ERR MSG PARMS
SQLERRP DS CL8 IMPL-DEPENDENT
SQLERRD DS 6F
SQLWARN DS 0C WARNING FLAGS
SQLWARN0 DS C'W' IF ANY
SQLWARN1 DS C'W' = WARNING
SQLWARN2 DS C'W' = WARNING
SQLWARN3 DS C'W' = WARNING
SQLWARN4 DS C'W' = WARNING
SQLWARN5 DS C'W' = WARNING
SQLWARN6 DS C'W' = WARNING
SQLWARN7 DS C'W' = WARNING
SQLEXT DS 0CL8
SQLWARN8 DS C
SQLWARN9 DS C
SQLWARNA DS C
SQLSTATE DS CL5

```

**C**

```

#ifndef SQLCODE
struct sqlca
{
 unsigned char sqlcaid[8];
 long sqlcabc;
 long sqlcode;
 short sqlerrml;
 unsigned char sqlerrmc[70];
 unsigned char sqlerrp[8];
 long sqlerrd[6];
 unsigned char sqlwarn[11];
 unsigned char sqlstate[5];
};

#define SQLCODE sqlca.sqlcode
#define SQLWARN0 sqlca.sqlwarn[0]
#define SQLWARN1 sqlca.sqlwarn[1]
#define SQLWARN2 sqlca.sqlwarn[2]
#define SQLWARN3 sqlca.sqlwarn[3]
#define SQLWARN4 sqlca.sqlwarn[4]
#define SQLWARN5 sqlca.sqlwarn[5]
#define SQLWARN6 sqlca.sqlwarn[6]
#define SQLWARN7 sqlca.sqlwarn[7]
#define SQLWARN8 sqlca.sqlwarn[8]
#define SQLWARN9 sqlca.sqlwarn[9]
#define SQLWARNA sqlca.sqlwarn[10]
#define SQLSTATE sqlca.sqlstate
#endif
struct sqlca sqlca;

```

## SQLCA

### COBOL:

```
01 SQLCA.
 05 SQLCAID PIC X(8).
 05 SQLCABC PIC S9(9) COMP-4.
 05 SQLCODE PIC S9(9) COMP-4.
 05 SQLERRM.
 49 SQLERRML PIC S9(4) COMP-4.
 49 SQLERRMC PIC X(70).
 05 SQLERRP PIC X(8).
 05 SQLERRD OCCURS 6 TIMES
 PIC S9(9) COMP-4.
 05 SQLWARN.
 10 SQLWARN0 PIC X.
 10 SQLWARN1 PIC X.
 10 SQLWARN2 PIC X.
 10 SQLWARN3 PIC X.
 10 SQLWARN4 PIC X.
 10 SQLWARN5 PIC X.
 10 SQLWARN6 PIC X.
 10 SQLWARN7 PIC X.
 05 SQLEXT.
 10 SQLWARN8 PIC X.
 10 SQLWARN9 PIC X.
 10 SQLWARNA PIC X.
 10 SQLSTATE PIC X(5).
```

### Fortran:

```
*
* THE SQL COMMUNICATIONS AREA
*
 INTEGER SQLCOD,
C SQLERR(6),
C SQLTXL*2
COMMON /SQLCA1/SQLCOD, SQLERR,SQLTXL
CHARACTER SQLERP*8,
C SQLWRN(0:7)*1,
C SQLTXT*70,
C SQLEXT*8,
C SQLWRX(1:3)*1,
C SQLSTT*5
COMMON /SQLCA2/SQLERP,SQLWRN,SQLTXT,SQLWRX,
C SQLSTT
EQUIVALENCE (SQLWRX,SQLEXT)
*
```

**PL/I:**

```

DECLARE
 1 SQLCA,
 2 SQLCAID CHAR(8),
 2 SQLCABC FIXED(31) BINARY,
 2 SQLCODE FIXED(31) BINARY,
 2 SQLERRM CHAR(70) VAR,
 2 SQLERRP CHAR(8),
 2 SQLERRD(6) FIXED(31) BINARY,
 2 SQLWARN,
 3 SQLWARN0 CHAR(1),
 3 SQLWARN1 CHAR(1),
 3 SQLWARN2 CHAR(1),
 3 SQLWARN3 CHAR(1),
 3 SQLWARN4 CHAR(1),
 3 SQLWARN5 CHAR(1),
 3 SQLWARN6 CHAR(1),
 3 SQLWARN7 CHAR(1),
 2 SQLEXT,
 3 SQLWARN8 CHAR(1),
 3 SQLWARN9 CHAR(1),
 3 SQLWARNA CHAR(1),
 3 SQLSTATE CHAR(5);

```

**The REXX SQLCA**

The REXX SQLCA consists of a set of variables, rather than a structure. DB2 makes the SQLCA available to your application automatically. Table 75 lists the variables in a REXX SQLCA.

*Table 75. Variables in a REXX SQLCA*

| Variable  | Contents                                                                                                                                                                                                                                                                                                                                                                                                                |
|-----------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| SQLCODE   | Contains the SQL return code.                                                                                                                                                                                                                                                                                                                                                                                           |
| SQLERRMC  | Contains one or more tokens, separated by X'FF', that are substituted for variables in the descriptions of error conditions.                                                                                                                                                                                                                                                                                            |
| SQLERRP   | Provides a product signature and, in the case of an error, diagnostic information such as the name of the module that detected the error. For DB2 for OS/390 and z/OS, the product signature is 'DSN'.                                                                                                                                                                                                                  |
| SQLERRD.1 | Contains the number of rows in a result table when the cursor position is after the last row (that is, when SQLCODE is equal to +100).<br><br>SQLERRD(1) can also contain an internal error code.                                                                                                                                                                                                                       |
| SQLERRD.2 | Contains the number of rows in a result table when the cursor position is after the last row (that is, when SQLCODE is equal to +100).<br><br>SQLERRD(2) can also contain an internal error code.                                                                                                                                                                                                                       |
| SQLERRD.3 | Contains the number of rows affected after INSERT, UPDATE, and DELETE (but not rows deleted as a result of CASCADE delete). Set to 0 if the SQL statement fails, indicating that all changes made in executing the statement were canceled. Set to -1 for a mass delete from a table in a segmented table space.<br><br>For SQLCODE -911 or -913, SQLERRD.3 can also contain the reason code for a timeout or deadlock. |

## SQLCA

Table 75. Variables in a REXX SQLCA (continued)

| Variable   | Contents                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
|------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| SQLERRD.4  | Generally, contains timerons, a short floating-point value that indicates a rough relative estimate of resources required. This value does not reflect an estimate of the time required to execute the SQL statement. After you prepare an SQL statement, you can use this field as an indicator of the relative cost of the prepared SQL statement. For a particular statement, this number can vary with changes to the statistics in the catalog. This value is subject to change between releases of DB2 for OS/390 and z/OS. |
| SQLERRD.5  | Contains the position or column of a syntax error for a PREPARE or EXECUTE IMMEDIATE statement.                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| SQLERRD.6  | Contains an internal error code.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| SQLWARN.0  | Contains a blank if no other indicator is set to a warning condition (that is, no other indicator contains a W or Z). Contains a W if at least one other indicator contains a W or Z.                                                                                                                                                                                                                                                                                                                                             |
| SQLWARN.1  | Contains a W if the value of a string column was truncated when assigned to a host variable. Contains an N for non-scrollable cursors and S for scrollable cursors after the OPEN CURSOR or ALLOCATE CURSOR statement.                                                                                                                                                                                                                                                                                                            |
| SQLWARN.2  | Contains a W if null values were eliminated from the argument of a column function; not necessarily set to W for the MIN function because its results are not dependent on the elimination of null values.                                                                                                                                                                                                                                                                                                                        |
| SQLWARN.3  | Contains a W if the number of result columns is larger than the number of host variables. Contains Z if the ASSOCIATE LOCATORS statement contains fewer locators than the stored procedure returned.                                                                                                                                                                                                                                                                                                                              |
| SQLWARN.4  | Contains a W if a prepared UPDATE or DELETE statement does not include a WHERE clause. For a scrollable cursor, contains an I for insensitive cursors and S for sensitive static cursors after the OPEN CURSOR or ALLOCATE CURSOR statement; otherwise, blank if cursor is not scrollable.                                                                                                                                                                                                                                        |
| SQLWARN.5  | Contains a W if the SQL statement was not executed because it is not a valid SQL statement in DB2 for OS/390 and z/OS. Contains a character value of 1 (read only), 2 (read and delete), or 4 (read, delete, and update) to reflect capability of the cursor after the OPEN CURSOR or ALLOCATE CURSOR statement.                                                                                                                                                                                                                  |
| SQLWARN.6  | Contains a W if the addition of a month or year duration to a DATE or TIMESTAMP value results in an invalid day (for example, June 31). Indicates that the value of the day was changed to the last day of the month to make the result valid.                                                                                                                                                                                                                                                                                    |
| SQLWARN.7  | Contains a W if one or more nonzero digits were eliminated from the fractional part of a number that was used as the operand of a decimal multiply or divide operation.                                                                                                                                                                                                                                                                                                                                                           |
| SQLWARN.8  | Contains a W if a character that could not be converted was replaced with a substitute character.                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| SQLWARN.9  | Contains a W if arithmetic exceptions were ignored during COUNT or COUNT_BIG processing. Contains a Z if the stored procedure returned multiple result sets.                                                                                                                                                                                                                                                                                                                                                                      |
| SQLWARN.10 | Contains a W if at least one character field of the SQLCA is invalid due to a character conversion error.                                                                                                                                                                                                                                                                                                                                                                                                                         |
| SQLSTATE   | Contains a return code for the outcome of the most recent execution of an SQL statement.                                                                                                                                                                                                                                                                                                                                                                                                                                          |

## SQL descriptor area (SQLDA)

An SQLDA is a collection of variables that is required for execution of the SQL DESCRIBE statement, and can be optionally used by the PREPARE, OPEN, FETCH, EXECUTE, and CALL statements. An SQLDA can be used in a DESCRIBE or PREPARE INTO statement, modified with the addresses of host variables, and then reused in a FETCH statement.

The meaning of the information in an SQLDA depends on the context in which it is used. For DESCRIBE and PREPARE INTO, DB2 sets the fields in the SQLDA to provide information to the application program. For OPEN, EXECUTE, FETCH, and CALL, the application program sets the fields in the SQLDA to provide DB2 with information:

#### **DESCRIBE *statement-name* or PREPARE INTO**

With the exception of SQLN, DB2 sets fields of the SQLDA to provide information to an application program about a prepared statement. Each SQLVAR occurrence describes a column of the result table.

#### **DESCRIBE TABLE**

With the exception of SQLN, DB2 sets fields of the SQLDA to provide information to an application program about the columns of a table or view. Each SQLVAR occurrence describes a column of the specified table or view.

#### **DESCRIBE CURSOR**

With the exception of SQLN, DB2 sets fields of the SQLDA to provide information to an application program about the result set that is associated with the specified cursor. Each SQLVAR occurrence describes a column of the result set.

#### **DESCRIBE INPUT**

With the exception of SQLN, DB2 sets fields of the SQLDA to provide information to an application program about the input parameter markers of a prepared statement. Each SQLVAR occurrence describes an input parameter marker.

#### **DESCRIBE PROCEDURE**

With the exception of SQLN, DB2 sets fields of the SQLDA to provide information to an application program about the result sets returned by the specified stored procedure. Each SQLVAR occurrence describes a returned result set.

#### **OPEN, EXECUTE, FETCH, and CALL**

The application program sets fields of the SQLDA to provide information about host variables or output buffers in the application program to DB2. Each SQLVAR occurrence describes a host variable or output buffer.

- For OPEN and EXECUTE, each SQLVAR occurrence describes an input value that is substituted for a parameter marker in the associated SQL statement that was previously prepared.
- For FETCH, each SQLVAR occurrence describes a host variable or buffer in the application program that is to be used to contain an output value from a row of the result.
- For CALL, each SQLVAR occurrence describes a host variable that corresponds to a parameter in the parameter list for the stored procedure.

For information on the way to use the SQLDA, see *DB2 Application Programming and SQL Guide*.

The following sections discuss the fields of the SQLDA and the format of the SQLDA for each language. Because the fields and format of the SQLDA for REXX is somewhat different from the SQLDAs for other languages, the REXX SQLDA is discussed separately.

## Field descriptions

An SQLDA consists of four variables, a header, and an arbitrary number of occurrences of a sequence of variables collectively named SQLVAR. In DESCRIBE and PREPARE INTO, each occurrence of the SQLVAR describes the column of a table. In FETCH, OPEN, EXECUTE, and CALL, each occurrence describes a host variable.

The next section describes the SQLDA header, and “SQLVAR entries” on page 989 describes the SQLVAR and how to determine how many SQLVAR entries to allocate in an SQLDA.

### The SQLDA Header

Table 76 describes the fields in the SQLDA header.

*Table 76. Fields of the SQLDA header*

| C name<br>assembler,<br>COBOL or<br>PL/I name | Data<br>type | Usage in DESCRIBE <sup>1</sup><br>and PREPARE INTO                                                                                                                                                                                                                                                                                                                                                                                                                                                                      | Usage in FETCH, OPEN,<br>EXECUTE, and CALL                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
|-----------------------------------------------|--------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| sqlaid<br>SQLDAID                             | CHAR(8)      | <p>An “eye catcher” for storage dumps, containing the text 'SQLDA '.</p> <p>The 7th byte of the field is a flag that can be used to determine if more than one SQLVAR entry is needed for each column. For details , see “Determining how many SQLVAR occurrences are needed” on page 991.</p> <p>For DESCRIBE CURSOR, the field is set to 'SQLRS'. If the cursor is declared WITH HOLD in a stored procedure, the high-order bit of the 8th byte is set to 1.</p> <p>For DESCRIBE PROCEDURE, it is set to 'SQLPR'.</p> | <p>A plus sign (+) in the 6th byte indicates that SQLNAME contains an overriding CCSID.</p> <p>A '2' in the 7th byte indicates the two SQLVAR entries were allocated for each column or parameter.</p> <p>A '3' in the 7th byte indicates that three SQLVAR entries were allocated for each column or parameter.</p> <p>Although three entries are never needed on input to DB2, an SQLDA with three entries might be used when the SQLDA was initialized by a DESCRIBE or PREPARE INTO with a USING BOTH clause.</p> <p>Otherwise, SQLDAID is not used.</p> |
| sqldabc<br>SQLDABC                            | INTEGER      | Length of the SQLDA, equal to SQLN×44+16.                                                                                                                                                                                                                                                                                                                                                                                                                                                                               | Length of the SQLDA, greater than or equal to SQLN×44+16.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |

Table 76. Fields of the SQLDA header (continued)

| C name<br>assembler,<br>COBOL or<br>PL/I name | Data<br>type | Usage in DESCRIBE <sup>1</sup><br>and PREPARE INTO                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       | Usage in FETCH, OPEN,<br>EXECUTE, and CALL                                                                              |
|-----------------------------------------------|--------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------|
| sqln<br>SQLN                                  | SMALLINT     | <p>Unchanged by DB2. The field must be set to a value greater than or equal to zero before the statement is executed. The field indicates the total number of occurrences of SQLVAR. At the very least, the number should be:</p> <ul style="list-style-type: none"> <li>• For DESCRIBE INPUT, the number of parameter markers to be described.</li> <li>• For other DESCRIBE or PREPARE INTO: the number of columns of the result, or a multiple of the columns of the result when there are multiple sets of SQLVAR entries because column labels are returned in addition to column names.</li> </ul> | Total number of occurrences of SQLVAR provided in the SQLDA. SQLN must be set to a value greater than or equal to zero. |
| sqld<br>SQLD                                  | SMALLINT     | <p>The number of columns described by occurrences of SQLVAR. Double that number if USING BOTH appears in the DESCRIBE or PREPARE INTO statement. Contains a 0 if the statement string is not a query.</p> <p>For DESCRIBE PROCEDURE, the number of result sets returned by the stored procedure. Contains a 0 if no result sets are returned.</p>                                                                                                                                                                                                                                                        | The number of host variables described by occurrences of SQLVAR.                                                        |

**Notes:**

1. The third column of this table represents several forms of the DESCRIBE statement:
  - For DESCRIBE *output* and PREPARE INTO, the column pertains to columns of the result table.
  - For DESCRIBE CURSOR, the column pertains to a result set associated with a cursor.
  - For DESCRIBE INPUT, the column pertains to parameter markers.
  - For DESCRIBE PROCEDURE, the column pertains to the result sets returned by the stored procedure.

**SQLVAR entries**

For each column or host variable described by the SQLDA, there are two types of SQLVAR entries:

**Base SQLVAR entry**

The base SQLVAR entry is always present. The fields of this entry contain the base information about the column or host variable such as data type code, length attribute (except for LOBs), column name (or label), host variable address, and indicator variable address.

**Extended SQLVAR entry**

The extended SQLVAR entry is needed (for each column) if the result includes any LOB or distinct type<sup>42</sup> columns. For distinct types, the

42. DESCRIBE INPUT does not return information about distinct types.

## SQLDA

extended SQLVAR contains the distinct type name. For LOBs, the extended SQLVAR contains the length attribute of the host variable and a pointer to the buffer that contains the actual length. If locators are used to represent LOBs, an extended SQLVAR is not necessary.

The extended SQLVAR entry is also needed for each column when the USING BOTH clause was specified, which indicates that both column names and labels are returned. (DESCRIBE *output* is the only statement with the USING BOTH clause).

The fields in the extended SQLVAR that return LOB and distinct type information do not overlap, and the fields that return LOB and label information do not overlap. Depending on the combination of labels, LOBs and distinct types, more than one extended SQLVAR entry per column may be required to return the information. See “Determining how many SQLVAR occurrences are needed” on page 991.

Table 77 shows how to map the base and extended SQLVAR entries. For an SQLDA that contains both base and extended SQLVAR entries, the base SQLVAR entries are in the first block, followed by a block of extended SQLVAR entries, which if necessary, are followed by a second block of extended SQLVAR entries. In each block, the number of occurrences of the SQLVAR entry is equal to the value in SQLD<sup>43</sup> even though many of the extended SQLVAR entries might be unused.

Table 77. Contents of SQLVAR arrays

| LOBs                                    | Distinct types <sup>1</sup> | 7th byte of SQLDAID SQLD |                       |                      | Minimum for SQLN <sup>2</sup> | Set of SQLVAR entries         |          |  |
|-----------------------------------------|-----------------------------|--------------------------|-----------------------|----------------------|-------------------------------|-------------------------------|----------|--|
|                                         |                             | First set (Base)         | Second set (Extended) | Third set (Extended) |                               |                               |          |  |
| <b>USING BOTH clause not specified:</b> |                             |                          |                       |                      |                               |                               |          |  |
| No                                      | No                          | Space                    | <i>n</i>              | <i>n</i>             | Column names or labels        | Not Used                      | Not Used |  |
| Yes <sup>3</sup>                        | Yes <sup>3</sup>            | 2                        | <i>n</i>              | $2n$                 | Column names or labels        | LOBs, distinct types, or both | Not used |  |
| <b>USING BOTH clause was specified:</b> |                             |                          |                       |                      |                               |                               |          |  |
| No                                      | No                          | Space                    | $2n$                  | $2n$                 | Column names                  | Labels                        | Not used |  |
| Yes                                     | No                          | 2                        | <i>n</i>              | $2n$                 | Column names                  | LOBs and labels               | Not used |  |
| No                                      | Yes                         | 3                        | <i>n</i>              | $3n$                 | Column names                  | Distinct types                | Labels   |  |
| Yes                                     | Yes                         | 3                        | <i>n</i>              | $3n$                 | Column names                  | LOBs and distinct types       | Labels   |  |

### Notes:

1. DESCRIBE INPUT does not return information about distinct types.
2. The number of columns or host variables that the SQLDA describes.
3. Either LOBs, distinct types, or both are present.
4. Here, the 7th byte is set to a space and SQLD is set to two times the number of columns in the result. For all other values of the 7th byte for USING BOTH, SQLD is set to the number of columns in the result, and the 7th byte can be used to determine how many SQLVAR entries are needed for each column of the result.

43. When an extended SQLVAR entry is present for each column for *labels* (and there are no LOB or distinct type columns in the result),

**Determining how many SQLVAR occurrences are needed:** The number of SQLVAR occurrences needed depends on the statement that the SQLDA was provided for and the data types of the columns or parameters being described. See the “Minimum Number of SQLVARs Needed” column in Table 77 on page 990.

If the USING BOTH clause was not specified for the statement and neither LOBs nor distinct types are present in the result, only one SQLVAR entry (a base entry) is needed for each column. The 7th byte of SQLDAID is set to a space. The SQLD is set to the number of columns in the result and represents the number of SQLVAR occurrences needed. If an insufficient number of SQLVAR occurrences were provided, DB2 returns a +236 warning in SQLCODE if the standards option was set. Otherwise, SQLCODE is zero.

If USING BOTH is specified and neither LOBs nor distinct types are present in the result, an extended SQLVAR entry per column is needed for the labels in addition to the base SQLVAR entry. The 7th byte of the SQLDAID is set to space. SQLD is set to twice the number of columns in the result and represents the combined number of base and extended SQLVAR occurrences needed.

If LOBs, distinct types, or both are present in the results, one extended SQLVAR entry is needed per column in addition to the base SQLVAR entry with one exception. The exception is that when the USING BOTH clause is specified and distinct types are present in the results, two extended SQLVAR entries per column are needed. When a **sufficient number of SQLVAR entries are provided in the SQLDA** for both the base and extended SQLVARs, information for the LOBs and distinct types is returned. The 7th byte of SQLDAID is set to the number of SQLVAR entries that were used for each column:

- 2 Two SQLVAR entries per column (a base and an extended)
- 3 Three SQLVAR entries per column (a base and two extended)

SQLD is set to the number of columns in the result. Therefore, the value of the 7th byte of SQLDAID multiplied by the value of SQLD is the total number SQLVAR entries that were provided.

Otherwise, when an **insufficient number of SQLVAR entries have been provided** when LOBs or distinct types are present, DB2 indicates that by returning one of the following warnings in SQLCODE. DB2 also sets the 7th byte of SQLDAID to indicate how many SQLVAR entries are needed for each column of the result.

- +237 There are insufficient SQLVAR entries to describe the data, and the data includes distinct types. In this case, there were enough *base* SQLVAR entries to describe the data, so the *base* SQLVAR entries are set. However, sufficient *extended* SQLVAR entries were not provided so the distinct type names are not returned.
- +238 There are insufficient SQLVAR entries to describe the data, and the data includes LOBs. In this case no information is returned in the SQLVAR entries.
- +239 There are insufficient SQLVAR entries to describe the data, and the data includes distinct types. There weren't even enough *base* SQLVAR entries. In this case no information is returned in the SQLVAR entries.

**Field descriptions of an occurrence of a base SQLVAR:** Table 78 on page 992 describes the contents of the fields of a base SQLVAR.

## SQLDA

Table 78. Fields in an occurrence of a base SQLVAR

| C name<br>assembler<br>COBOL, or<br>PL/I name | Data<br>type | Usage in DESCRIBE <sup>1</sup><br>and PREPARE INTO                                                                                                                                                                                                                                                                                                                                                                                                  | Usage in FETCH, OPEN,<br>EXECUTE, and CALL                                                                                                                                                                                                                                                                                  |
|-----------------------------------------------|--------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| sqltype<br>SQLTYPE                            | SMALLINT     | <p>Indicates the data type of the column or parameter and whether it can contain null values. For a description of the type codes, see Table 80 on page 996.</p> <p>For a distinct type, the data type on which the distinct type was based is placed in this field. The base SQLVAR provides no indication that this is part of the description of a distinct type.</p>                                                                            | <p>Indicates the data type of the host variable and whether an indicator variable is provided. Host variables for datetime values must be character string variables. For FETCH, a datetime type code means a fixed-length character string. For a description of the type codes, see “SQLTYPE and SQLLEN” on page 995.</p> |
| sqllen<br>SQLLEN                              | SMALLINT     | <p>The length attribute of the column or parameter. For datetime data, the length of the string representation of the value. See “SQLTYPE and SQLLEN” on page 995 for a description of allowable values.</p> <p>For LOBs, the value is 0 regardless of the length attribute of the LOB. Field SQLLONGLEN in the extended SQLVAR contains the length attribute.</p>                                                                                  | <p>The length attribute of the host variable. See “SQLTYPE and SQLLEN” on page 995 for a description of allowable values.</p> <p>For LOBs, the value is 0 regardless of the length attribute of the LOB. Field SQLLONGLEN in the extended SQLVAR contains the length attribute.</p>                                         |
| sqldata<br>SQLDATA                            | pointer      | <p>For string columns or parameters, SQLDATA contains X'0000zzzz', where zzzz is the associated CCSID. For character strings, SQLDATA can alternatively contain X'FFFF', which indicates bit data. Not used for other types of data.</p> <p>For datetime columns, SQLDATA can contain the CCSID of the string representation of the datetime value.</p> <p>For DESCRIBE PROCEDURE, the result set locator value associated with the result set.</p> | Contains the address of the host variable.                                                                                                                                                                                                                                                                                  |
| sqlind<br>SQLIND                              | pointer      | <p>Reserved</p> <p>For DESCRIBE PROCEDURE, it is set to -1.</p>                                                                                                                                                                                                                                                                                                                                                                                     | Contains the address of an associated indicator variable, if SQLTYPE is odd. Otherwise, the field is not used.                                                                                                                                                                                                              |

Table 78. Fields in an occurrence of a base SQLVAR (continued)

| C name<br>assembler<br>COBOL, or<br>PL/I name | Data<br>type | Usage in DESCRIBE <sup>1</sup><br>and PREPARE INTO                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         | Usage in FETCH, OPEN,<br>EXECUTE, and CALL                                                                                                                                                                                                                                                                                                                                                              |
|-----------------------------------------------|--------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| sqlname<br>SQLNAME                            | VARCHAR(30)  | <p>Contains the unqualified name or label of the column, or a string of length zero if the name or label does not exist. If the prepared statement includes a UNION or UNION ALL clause, SQLNAME contains the name or label, if any, of the corresponding column of the first operand of the UNION.</p> <p>For DESCRIBE PROCEDURE, SQLNAME contains the cursor name used by the stored procedure to return the result set. The values for SQLNAME appear in the order the cursors were opened by the stored procedure.</p> <p>For DESCRIBE INPUT, SQLNAME is not used.</p> | <p>Can contain a CCSID. DB2 interprets the third and fourth byte of SQLNAME as the CCSID of the host variable if all of the following are true:</p> <ul style="list-style-type: none"> <li>• The 6th byte of SQLDAID is '+'</li> <li>• SQLTYPE indicates the host variable is a string variable</li> <li>• The length of SQLNAME is 8</li> <li>• The first two bytes of SQLNAME are X'0000'.</li> </ul> |

**Notes:**

1. The third column of this table represents several forms of the DESCRIBE statement.
  - For DESCRIBE *output* and PREPARE INTO, the column pertains to columns of the result table.
  - For DESCRIBE CURSOR, the column pertains to a result set associated with a cursor.
  - For DESCRIBE INPUT, the column pertains to parameter markers.
  - For DESCRIBE PROCEDURE, the column pertains to the result sets returned by the stored procedure.

**Field descriptions of an occurrence of an extended SQLVAR:** Table 79 describes the contents of the fields of an extended SQLVAR entry.

Table 79. Fields in an occurrence of an extended SQLVAR

| C name<br>assembler,<br>COBOL, or<br>PL/I name | Data<br>type | Usage in DESCRIBE <sup>1</sup><br>and PREPARE INTO            | Usage in FETCH, OPEN,<br>EXECUTE, and CALL                                                                                                                                                                                                                   |
|------------------------------------------------|--------------|---------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| len.sqllonglen<br>SQLLONGL<br>SQLLONGLEN       | INTEGER      | The length attribute of a LOB (BLOB, CLOB, or DBCLOB) column. | The length attribute of a LOB (BLOB, CLOB, or DBCLOB) host variable. DB2 ignores the SQLLEN field in the base SQLVAR for these data types. The length attribute indicates the number of bytes for a BLOB or CLOB, and the number of characters for a DBCLOB. |
| *                                              | INTEGER      | Reserved.                                                     | Reserved.                                                                                                                                                                                                                                                    |

## SQLDA

Table 79. Fields in an occurrence of an extended SQLVAR (continued)

| C name<br>assembler,<br>COBOL, or<br>PL/I name | Data<br>type | Usage in DESCRIBE <sup>1</sup><br>and PREPARE INTO | Usage in FETCH, OPEN,<br>EXECUTE, and CALL                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
|------------------------------------------------|--------------|----------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| sqldatalen<br>SQLDATALEN                       | pointer      | Not used.                                          | <p>Used only for LOB (BLOB, CLOB, and DBCLOB) host variables.</p> <p>If the value of the field is null, the actual length of the LOB is stored in the 4 bytes immediately before the start of the data, and SQLDATA points to the first byte of the field length. The actual length indicates the number of bytes for a BLOB or CLOB, and the number of characters for a DBCLOB.</p> <p>If the value of the field is not null, the field contains a pointer to a 4-byte long buffer that contains the actual length <i>in bytes</i> (even for DBCLOBs) of the data in the buffer pointed to from the SQLDATA field in the matching base SQLVAR.</p> <p>Regardless of whether this field is used, field SQLLONGLEN must be set.</p> |

Table 79. Fields in an occurrence of an extended SQLVAR (continued)

| C name<br>assembler,<br>COBOL, or<br>PL/I name      | Data<br>type | Usage in DESCRIBE <sup>1</sup><br>and PREPARE INTO                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     | Usage in FETCH, OPEN,<br>EXECUTE, and CALL |
|-----------------------------------------------------|--------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------|
| sqldatatype_name<br>SQLNAME<br>SQLDATATYPE-<br>NAME | VARCHAR(30)  | <p>A SQLNAME field of the appropriate extended SQLVAR is set to one of the following as per Table 77 on page 990.</p> <ul style="list-style-type: none"> <li>• For a distinct type column, DB2 sets this to the fully qualified distinct type name. The first 8 bytes contain the schema name of the type, (extended to the right with spaces, if necessary). Byte 9 contains a period (.). Bytes 10 to 27 contain the low order portion of the type name, which is not extended to the right with spaces. Otherwise, schema name is SYSIBM and the low order portion of the name is the name of the type from the catalog.</li> <li>• DESCRIBE INPUT does not return information about distinct type types.</li> <li>• For a label, this field is set to the contents of the label.</li> </ul> <p>In the case that both a distinct type name and a label need to be returned in extended SQLVAR entries (USING BOTH has been specified), the distinct type name is returned in the first extended SQLVAR entry and the label in the second extended SQLVAR entry.</p> | <p>Not used.</p>                           |

**Notes:**

1. The third column of this table represents several forms of the DESCRIBE statement.
  - For DESCRIBE *output* and PREPARE INTO, the column pertains to columns of the result table.
  - For DESCRIBE CURSOR, the column pertains to a result set associated with a cursor.
  - For DESCRIBE INPUT, the column pertains to parameter markers.
  - For DESCRIBE PROCEDURE, the column pertains to the result sets returned by the stored procedure.

**SQLTYPE and SQLLEN:** The following table shows the values that may appear in the SQLTYPE and SQLLEN fields of the SQLDA. In DESCRIBE and PREPARE INTO, an even value of SQLTYPE means the column does not allow nulls, and an odd value means the column does allow nulls. In FETCH, OPEN, EXECUTE, and CALL, an even value of SQLTYPE means no indicator variable is provided, and an odd value means that SQLIND contains the address of an indicator variable.

## SQLDA

Table 80. SQLTYPE and SQLLEN values for DESCRIBE, PREPARE INTO, FETCH, OPEN, EXECUTE, and CALL

| SQLTYPE | For DESCRIBE and PREPARE INTO        |                                                   | For FETCH, OPEN, EXECUTE, and CALL                          |                                                   |
|---------|--------------------------------------|---------------------------------------------------|-------------------------------------------------------------|---------------------------------------------------|
|         | Column or Parameter<br>data type     | SQLLEN                                            | Host variable data<br>type                                  | SQLLEN                                            |
| 384/385 | date                                 | 10 <sup>1</sup>                                   | fixed-length character string representation of a date      | length attribute of the host variable             |
| 388/389 | time                                 | 8 <sup>2</sup>                                    | fixed-length character string representation of a time      | length attribute of the host variable             |
| 392/393 | timestamp                            | 26                                                | fixed-length character string representation of a timestamp | length attribute of the host variable             |
| 400/401 | N/A                                  | N/A                                               | NUL-terminated graphic string                               | length attribute of the host variable             |
| 404/405 | BLOB                                 | 0 <sup>3</sup>                                    | BLOB                                                        | Not used. <sup>3</sup>                            |
| 408/409 | CLOB                                 | 0 <sup>3</sup>                                    | CLOB                                                        | Not used. <sup>3</sup>                            |
| 412/413 | DBCLOB                               | 0 <sup>3</sup>                                    | DBCLOB                                                      | Not used. <sup>3</sup>                            |
| 448/449 | varying-length character string      | length attribute of the column                    | varying-length character string                             | length attribute of the host variable             |
| 452/453 | fixed-length character string        | length attribute of the column                    | fixed-length character string                               | length attribute of the host variable             |
| 456/457 | long varying-length character string | length attribute of the column                    | long varying-length character string                        | length attribute of the host variable             |
| 460/461 | N/A                                  | N/A                                               | NUL-terminated character string                             | length attribute of the host variable             |
| 464/465 | varying-length graphic string        | length attribute of the column                    | varying-length graphic string                               | length attribute of the host variable             |
| 468/469 | fixed-length graphic string          | length attribute of the column                    | fixed-length graphic string                                 | length attribute of the host variable             |
| 472/473 | long varying-length graphic string   | length attribute of the column                    | long graphic string                                         | length attribute of the host variable             |
| 480/481 | floating point                       | 4 for single precision,<br>8 for double precision | floating point                                              | 4 for single precision,<br>8 for double precision |
| 484/485 | packed decimal                       | precision in byte 1;<br>scale in byte 2           | packed decimal                                              | precision in byte 1;<br>scale in byte 2           |
| 496/497 | large integer                        | 4                                                 | large integer                                               | 4                                                 |
| 500/501 | small integer                        | 2                                                 | small integer                                               | 2                                                 |
| 504/505 | N/A                                  | N/A                                               | DISPLAY SIGN<br>LEADING SEPARATE                            | precision in byte 1;<br>scale in byte 2           |
| 904/905 | N/A                                  | N/A                                               | ROWID                                                       | 40                                                |
| 960/961 | N/A                                  | N/A                                               | BLOB locator                                                | 4                                                 |
| 964/965 | N/A                                  | N/A                                               | CLOB locator                                                | 4                                                 |
| 968/969 | N/A                                  | N/A                                               | DBCLOB locator                                              | 4                                                 |
| 972/973 | result set locator                   | 4                                                 | result set locator                                          | 4                                                 |
| 976/977 | table locator                        | 4                                                 | table locator                                               | 4                                                 |

Table 80. SQLTYPE and SQLLEN values for DESCRIBE, PREPARE INTO, FETCH, OPEN, EXECUTE, and CALL (continued)

| SQLTYPE | For DESCRIBE and PREPARE INTO |        | For FETCH, OPEN, EXECUTE, and CALL |        |
|---------|-------------------------------|--------|------------------------------------|--------|
|         | Column or Parameter data type | SQLLEN | Host variable data type            | SQLLEN |

**Notes:**

1. Might be different if a date installation exit is specified.
2. Might be different if a time installation exit is specified.
3. Field SQLLONGLEN in the extended SQLVAR contains the length attribute of the column.

**SQLDATA:** The following table identifies the CCSID values that appear in the SQLDATA field when the SQLVAR element describes a string column.

Table 81. CCSID values for SQLDATA

| Data type           | Subtype    | Bytes 1 and 2 | Bytes 3 and 4 |
|---------------------|------------|---------------|---------------|
| Character           | SBCS data  | X'0000'       | CCSID         |
| Character           | mixed data | X'0000'       | CCSID         |
| Character           | BIT data   | X'0000'       | X'FFFF'       |
| Graphic             | N/A        | X'0000'       | CCSID         |
| Any other data type | N/A        | N/A           | N/A           |

## Unrecognized and unsupported SQLTYPES

The values that appear in the SQLTYPE field of the SQLDA are dependent on the level of data type support available at the sender as well as at the receiver of the data. This support is particularly important as new data types are added to the product.

New data types may or may not be supported by the sender or receiver of the data and may or may not be recognized by the sender or receiver of the data. Depending on the situation, the new data type may be returned, or a compatible data type that is agreed to by both the sender and the receiver of the data may be returned or an error may occur.

When the sender and receiver agree to use a compatible data type, the following table indicates the mapping that takes place. This mapping takes place when at least one of the sender or receiver does not support the data type provided. The unsupported data type can be provided by either the application or the database manager.

Table 82. Compatible data types for unsupported data types

| Data type | Compatible data type     |
|-----------|--------------------------|
| BIGINT    | DECIMAL(19,0) (1)        |
| ROWID     | VARCHAR(40) FOR BIT DATA |

**Notes:**

1. BIGINT is supported by other DB2 platforms.

Note that no indication is given in the SQLDA that the data type is substituted.

## SQLDA

### The included SQLDA

The description of the SQLDA that is given by INCLUDE SQLDA is shown below. Only assembler, C, C++, COBOL<sup>44</sup>, and PL/I C are supported.

#### assembler:

```
SQLTRIPL EQU C'3'
SQLDOUBL EQU C'2'
SQLSINGL EQU C' '
*
 SQLSECT SAVE
*
SQLDA DSECT
SQLDAID DS CL8 ID
SQLDABC DS F BYTE COUNT
SQLN DS H COUNT SQLVAR/SQLVAR2 ENTRIES
SQLD DS H COUNT VARS (TWICE IF USING BOTH)
*
SQLVAR DS OF BEGIN VARS
SQLVARN DSECT ,
SQLTYPE DS H DATA TYPE CODE
SQLLEN DS OH LENGTH
SQLPRCSN DS X DEC PRECISION
SQLSCALE DS X DEC SCALE
SQLDATA DS A ADDR OF VAR
SQLIND DS A ADDR OF IND
SQLNAME DS H,CL30 DESCRIBE NAME
SQLVSIZ EQU *-SQLDATA
SQLSIZV EQU *-SQLVARN
*
SQLDA DSECT
SQLVAR2 DS OF BEGIN EXTENDED FIELDS OF VARS
SQLVAR2N DSECT ,
SQLLONGL DS F LENGTH
SQLRSVDL DS F RESERVED
SQLDATAL DS A ADDR OF LENGTH IN BYTES
SQLTNAME DS H,CL30 DESCRIBE NAME
*
 SQLSECT RESTORE
```

In the above declaration, SQLSECT SAVE is a macro invocation that remembers the current CSECT name. SQLSECT RESTORE is a macro invocation that continues that CSECT.

#### C and C++:

---

44. Excluding OS/VS COBOL

```

#ifndef SQLDASIZE /* Permit duplicate Includes */
/**/
struct sqlvar
 { short sqltype;
 short sqllen;
 char *sqldata;
 short *sqlind;
 struct sqlname
 { short length;
 char data[30];
 } sqlname;
 };
/**/
struct sqlvar2
 { struct
 { long sqllonglen;
 unsigned long reserved;
 } len;
 char *sqldatalen;
 struct sqldistinct_type
 { short length;
 char data[30];
 } sqldatatype_name;
 };
/**/
struct sqlda
 { char sqldaid[8];
 long sqldabc;
 short sqln;
 short sqld;
 struct sqlvar sqlvar[1];
 };
/**/
/***/
/* Macros for using the sqlvar2 fields. */
/***/
/**/
/***/
/* '2' in the 7th byte of sqldaid indicates a doubled number of */
/* sqlvar entries. */
/* '3' in the 7th byte of sqldaid indicates a tripled number of */
/* sqlvar entries. */
/***/
#define SQLDOUBLED '2'
#define SQLTRIPLED '3'
#define SQLSINGLED ' '
/**/

```

## SQLDA

```
/**/
/* GETSQLDOUBLED(daptr) returns 1 if the SQLDA pointed to by */
/* daptr has been doubled, or 0 if it has not been doubled. */
/**/
#define GETSQLDOUBLED(daptr) \
 (((daptr)->sqldaid[6] == (char) SQLDOUBLED) ? \
 (1) : \
 (0))
/**/
/**/
/* GETSQLTRIPLED(daptr) returns 1 if the SQLDA pointed to by */
/* daptr has been tripled, or 0 if it has not been tripled. */
/**/
#define GETSQLTRIPLED(daptr) \
 (((daptr)->sqldaid[6] == (char) SQLTRIPLED) ? \
 (1) : \
 (0))
/**/
/**/
/* SETSQLDOUBLED(daptr, SQLDOUBLED) sets the 7th byte of sqldaid */
/* to '2'. */
/* SETSQLDOUBLED(daptr, SQLSINGLED) sets the 7th byte of sqldaid */
/* to be a ' '. */
/**/
#define SETSQLDOUBLED(daptr, newvalue) \
 (((daptr)->sqldaid[6] = (newvalue)))
/**/
/**/
/* SETSQLTRIPLED(daptr) sets the 7th byte of sqldaid */
/* to '3'. */
/**/
#define SETSQLTRIPLED(daptr) \
 (((daptr)->sqldaid[6] = (SQLTRIPLED)))
/**/
/**/
/* GETSQLDALONGLEN(daptr,n) returns the data length of the nth */
/* entry in the sqlda pointed to by daptr. Use this only if the */
/* sqlda was doubled or tripled and the nth SQLVAR entry has a */
/* LOB datatype. */
/**/
#define GETSQLDALONGLEN(daptr,n) \
 (long) (((struct sqlvar2 *) &((daptr);->sqlvar[(n) + \
 ((daptr)->sqld)])) \ \
 ->len.sqllonglen)
/**/
```

```

/*****************/
/* SETSQLDALONGLEN(daptr,n,len) sets the sqllonglen field of the */
/* sqlda pointed to by daptr to len for the nth entry. Use this only */
/* if the sqlda was doubled or tripled and the nth SQLVAR entry has */
/* a LOB datatype. */
/*****************/
#define SETSQLDALONGLEN(daptr,n,length) { \
 struct sqlvar2 *var2ptr; \
 var2ptr = (struct sqlvar2 *) \
 &((daptr);->sqlvar[(n) + ((daptr)->sqld)]); \
 var2ptr->len.sqllonglen = (long) (length); \
}
/**/
/*****************/
/* GETSQLDALENPTR(daptr,n) returns a pointer to the data length for */
/* the nth entry in the sqlda pointed to by daptr. Unlike the inline */
/* value (union sql8bytelen len), which is 8 bytes, the sqldatalen */
/* pointer field returns a pointer to a long (4 byte) integer. */
/* If the SQLDATALEN pointer is zero, a NULL pointer is be returned. */
/* */
/* NOTE: Use this only if the sqlda has been doubled or tripled. */
/*****************/
#define GETSQLDALENPTR(daptr,n) (\
 (((struct sqlvar2 *) &(daptr);->sqlvar[(n) + (daptr)->sqld]] \ \
 ->sqldatalen == NULL) ? \
 ((long *) NULL) : \
 ((long *) ((struct sqlvar2 *) \
 &(daptr);->sqlvar[(n) + (daptr)->sqld]] \ \
 ->sqldatalen))
*/
/*****************/
/* SETSQLDALENPTR(daptr,n,ptr) sets a pointer to the data length for */
/* the nth entry in the sqlda pointed to by daptr. */
/* Use this only if the sqlda has been doubled or tripled. */
/*****************/
#define SETSQLDALENPTR(daptr,n,ptr) { \
 struct sqlvar2 *var2ptr; \
 var2ptr = (struct sqlvar2 *) \
 &((daptr);->sqlvar[(n) + ((daptr)->sqld)]); \
 var2ptr->sqldatalen = (char *) ptr; \
}
/**/
#define SQLDASIZE(n) \
 (sizeof(struct sqlda) + ((n)-1) * sizeof(struct sqlvar))
#endif /* SQLDASIZE */

```

## SQLDA

### COBOL (IBM COBOL and VS COBOL II only):

```
01 SQLDA.
 05 SQLDAID PIC X(8).
 05 SQLDABC PIC S9(9) BINARY.
 05 SQLN PIC S9(4) BINARY.
 05 SQLD PIC S9(4) BINARY.
 05 SQLVAR OCCURS 0 TO 750 TIMES DEPENDING ON SQLN.
 10 SQLVAR1.
 15 SQLTYPE PIC S9(4) BINARY.
 15 SQLLEN PIC S9(4) BINARY.
 15 FILLER REDEFINES SQLLEN.
 20 SQLPRECISION PIC X.
 20 SQLSCALE PIC X.
 15 SQLDATA POINTER.
 15 SQLIND POINTER.
 15 SQLNAME.
 49 SQLNAMEL PIC S9(4) BINARY.
 49 SQLNAMEC PIC X(30).
 10 SQLVAR2 REDEFINES SQLVAR1.
 15 SQLVAR2-RESERVED-1 PIC S9(9) BINARY.
 15 SQLLONGLEN REDEFINES SQLVAR2-RESERVED-1
 PIC S9(9) BINARY.
 15 SQLVAR2-RESERVED-2 PIC S9(9) BINARY.
 15 SQLDATALEN POINTER.
 15 SQLDATATYPE-NAME.
 20 SQLDATATYPE-NAMEL PIC S9(4) BINARY.
 20 SQLDATATYPE-NAMEC PIC X(30).
```

### PL/I:

```
DECLARE
 1 SQLDA BASED(SQLDAPTR),
 2 SQLDAID CHAR(8),
 2 SQLDABC FIXED(31) BINARY,
 2 SQLN FIXED(15) BINARY,
 2 SQLD FIXED(15) BINARY,
 2 SQLVAR(SQLSIZE REFER(SQLN)),
 3 SQLTYPE FIXED(15) BINARY,
 3 SQLLEN FIXED(15) BINARY,
 3 SQLDATA POINTER,
 3 SQLIND POINTER,
 3 SQLNAME CHAR(30) VAR;
/* */
DECLARE
 1 SQLDA2 BASED(SQLDAPTR),
 2 SQLDAID2 CHAR(8),
 2 SQLDABC2 FIXED(31) BINARY,
 2 SQLN2 FIXED(15) BINARY,
 2 SQLD2 FIXED(15) BINARY,
 2 SQLVAR2(SQLSIZE REFER(SQLN2)),
 3 SQLBIGLEN,
 4 SQLLONGL FIXED(31) BINARY,
 4 SQLRSVDL FIXED(31) BINARY,
 3 SQLDATALEN POINTER,
 3 SQLTNAME CHAR(30) VAR;
/* */
DECLARE SQLSIZE FIXED(15) BINARY;
DECLARE SQLDAPTR POINTER;
DECLARE SQLTRIPLED CHAR(1) INITIAL('3');
DECLARE SQLDOUBLED CHAR(1) INITIAL('2');
DECLARE SQLSINGLED CHAR(1) INITIAL(' ');
```

## Identifying an SQLDA in C or C++

A *descriptor-name* can be specified in the CALL, DESCRIBE, EXECUTE, FETCH, and OPEN statements. When the host application is written in C or C++, *descriptor-name* can be a pointer variable with pointer notation.

For example, *descriptor-name* could be declared as

```
sqlda *outsqlda;
```

Afterwords, it could be used in a statement like the following:

```
EXEC SQL DESCRIBE STMT1 INTO DESCRIPTOR :*outsqlda;
```

## The REXX SQLDA

A REXX SQLDA consists of a set of REXX variables with a common stem. The stem must be a REXX variable name that contains no periods and is the same as the value of *descriptor-name* that you specify when you use the SQLDA in an SQL statement. DB2 does not support the INCLUDE SQLDA statement in REXX.

Table 83 shows the variable names in a REXX SQLDA. The values in the second column of the table are values that DB2 inserts into the SQLDA when the statement executes. Except where noted otherwise, the values in the third column of the table are values that the application must put in the SQLDA before the statement executes.

Table 83. Fields of a REXX SQLDA

| Variable name    | Usage in DESCRIBE and PREPARE INTO                                                                                                                                                                                                                                                                                                             | Usage in FETCH, OPEN, EXECUTE, and CALL                                 |
|------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------|
| <i>stem.SQLD</i> | <p>The number of columns that are described in the SQLDA. Double that number if USING BOTH appears in the DESCRIBE or PREPARE INTO statement. Contains a 0 if the statement string is not a query.</p> <p>For DESCRIBE PROCEDURE, the number of result sets returned by the stored procedure. Contains a 0 if no result sets are returned.</p> | <p>The number of host variables that are used by the SQL statement.</p> |

Each SQLDA contains *stem.SQLD* of the following variables. Therefore,  $1 \leq n \leq \text{stem.SQLD}$ . There is one occurrence of each variable for each column of the result table or host variable that is described by the SQLDA. This set of variables is equivalent to the SQLVAR structure in SQLDAs for other languages.

|                       |                                                                                                                                                                                                                                                                                                                                                                                      |                                                                                                                                                                                                                                                                                                                             |
|-----------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>stem.n.SQLTYPE</i> | <p>Indicates the data type of the column or parameter and whether it can contain null values. For a description of the type codes, see "SQLTYPE and SQLLEN" on page 995.</p> <p>For a distinct type, the data type on which the distinct type was based is placed in this field. The base SQLVAR provides no indication that this is part of the description of a distinct type.</p> | <p>Indicates the data type of the host variable and whether an indicator variable is provided. Host variables for datetime values must be character string variables. For FETCH, a datetime type code means a fixed-length character string. For a description of the type codes, see "SQLTYPE and SQLLEN" on page 995.</p> |
|-----------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

## SQLDA

Table 83. Fields of a REXX SQLDA (continued)

| Variable name                     | Usage in DESCRIBE and PREPARE INTO                                                                                                                                                                                                                                       | Usage in FETCH, OPEN, EXECUTE, and CALL                                                                                                                                                                                                                                                         |
|-----------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>stem.n.SQLLEN</i>              | For a column other than a DECIMAL or NUMERIC column, the length attribute of the column or parameter. For datetime data, the length of the string representation of the value. See "SQLTYPE and SQLLEN" on page 995 for a description of allowable values.               | For a host variable that does not have a decimal data type, the length attribute of the host variable. See "SQLTYPE and SQLLEN" on page 995 for a description of allowable values.                                                                                                              |
| <i>stem.n.SQLLEN.SQLPRECISION</i> | For a DECIMAL or NUMERIC column, the precision of the column or parameter.                                                                                                                                                                                               | For a host variable with a decimal data type, the precision of the host variable.                                                                                                                                                                                                               |
| <i>stem.n.SQLLEN.SQLSCALE</i>     | For a DECIMAL or NUMERIC column, the scale of the column or parameter.                                                                                                                                                                                                   | For a host variable with a decimal data type, the scale of the host variable.                                                                                                                                                                                                                   |
| <i>stem.n.SQLCCSID</i>            | For a string column or parameter, the CCSID of the column or parameter.                                                                                                                                                                                                  | For a string host variable, the CCSID of the host variable.                                                                                                                                                                                                                                     |
| <i>stem.n.SQLLOCATOR</i>          | For DESCRIBE PROCEDURE, the value of a result set locator.                                                                                                                                                                                                               | Not used.                                                                                                                                                                                                                                                                                       |
| <i>stem.n.SQLDATA</i>             | Not used.                                                                                                                                                                                                                                                                | Before EXECUTE or OPEN, contains the value of an input host variable. The application must supply this value.<br><br>After FETCH, contains the values of an output host variable.                                                                                                               |
| <i>stem.n.SQLIND</i>              | Not used.                                                                                                                                                                                                                                                                | Before EXECUTE or OPEN, contains a negative number to indicate that the input host variable in <i>stem.n.SQLDATA</i> is null. The application must supply this value.<br><br>After FETCH, contains a negative number if the value of the output host variable in <i>stem.n.SQLDATA</i> is null. |
| <i>stem.n.SQLNAME</i>             | The name of the <i>n</i> th column in the result table. For DESCRIBE PROCEDURE, contains the cursor name that is used by the stored procedure to return the result set. The values for SQLNAME appear in the order that the cursors were opened by the stored procedure. | Not used.                                                                                                                                                                                                                                                                                       |

## Appendix D. DB2 catalog tables

DB2 for OS/390 and z/OS maintains a set of tables (in database DSNDB06) called the DB2 catalog. This appendix describes that catalog by describing the columns of each catalog table.

The catalog tables describe such things as table spaces, tables, columns, indexes, privileges, application plans, and packages. Authorized users can query the catalog; however, it is primarily intended for use by DB2 and is therefore subject to change. All catalog tables are qualified by SYSIBM. Do not use this qualifier for user-defined tables.

The catalog tables are updated by DB2 during normal operations in response to certain SQL statements, commands, and utilities.

### Use as a programming interface

Not all catalog table columns are part of the general-use programming interface. Whether a column is part of this interface is indicated in a column labeled "Use" in the table that describes the column. The values that "Use" can assume are as follows:

#### Value Meaning

|   |                                                         |
|---|---------------------------------------------------------|
| G | Column is part of the general-use programming interface |
| S | Column is part of the product-sensitive interface       |
| I | Column is for internal use only                         |
| N | Column is not used                                      |

For columns for which "Use" is N or I, the name of the column and its description do not appear in the column's explanation.

### Release dependency indicators

Some objects depend on functions in particular releases of DB2. If you are running on a release of DB2 and fall back to a previous release, an object that depends on the more recent release becomes frozen. The object is marked with a release dependency indicator and is unavailable until remigration. The release dependency indicator, which is listed in the IBMREQD column of the catalog tables, shows the release of DB2 upon which the objects depends. Release dependency indicators in IBMREQD are defined by the following values:

#### Value Meaning

|   |                                                                                     |
|---|-------------------------------------------------------------------------------------|
| B | Version 1R3 dependency indicator, not from the machine-readable material (MRM) tape |
| C | Version 2R1 dependency indicator, not from MRM tape                                 |
| D | Version 2R2 dependency indicator, not from MRM tape                                 |
| E | Version 2R3 dependency indicator, not from MRM tape                                 |
| F | Version 3R1 dependency indicator, not from MRM tape                                 |
| G | Version 4 dependency indicator, not from MRM tape                                   |
| H | Version 5 dependency indicator, not from MRM tape                                   |
| I | Version 6 dependency indicator, not from MRM tape                                   |
| J | Version 6 dependency indicator, not from MRM tape                                   |
| K | Version 7 dependency indicator, not from MRM tape                                   |
| N | Not from MRM tape, no dependency                                                    |

---

## **Table spaces and indexes**

The table below shows to what table spaces the catalog tables are assigned, and what indexes they have. The pages that follow describe the columns in each table arranged alphabetically by table name. The indexes are in ascending order, except where noted.

## DB2 Catalog Tables

| TABLE SPACE<br>DSNDB06. ... | TABLE<br>SYSIBM. ... | Page     | INDEX<br>SYSIBM. ...                                                                                                                                                                   | INDEX FIELDS                                                    |
|-----------------------------|----------------------|----------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------|
| SYSCOPY                     | SYSCOPY              | 1040     | DSNUCH01                                                                                                                                                                               | DBNAME.TSNAME.START_RBA. <sup>1</sup><br>TIMESTAMP <sup>1</sup> |
|                             |                      |          | DSNUCX01                                                                                                                                                                               | DSNAME                                                          |
|                             |                      |          |                                                                                                                                                                                        |                                                                 |
| SYSDBASE                    | SYSCOLAUTH           | 1027     |                                                                                                                                                                                        |                                                                 |
|                             | SYSCOLUMNS           | 1032     | DSNDCX01                                                                                                                                                                               | TBCREATOR.TBNAME.NAME                                           |
|                             |                      |          | DSNDCX02                                                                                                                                                                               | TYPESCHEMA.TYPENAME                                             |
|                             | SYSFIELDS            | 1052     |                                                                                                                                                                                        |                                                                 |
|                             | SYSFOREIGNKEYS       | 1053     |                                                                                                                                                                                        |                                                                 |
|                             | SYSINDEXES           | 1054     | DSNDXX01                                                                                                                                                                               | CREATOR.NAME                                                    |
|                             |                      |          | DSNDXX02                                                                                                                                                                               | DBNAME.INDEXSPACE                                               |
|                             |                      |          | DSNDXX03                                                                                                                                                                               | TBCREATOR.TBNAME.CREATOR.<br>NAME                               |
|                             | SYSINDEXPART         | 1058     | DSNDRX01                                                                                                                                                                               | IXCREATOR.IXNAME.PARTITION                                      |
|                             |                      |          | DSNDRX02                                                                                                                                                                               | STORNAME                                                        |
|                             | SYSKEYS              | 1070     | DSNDKX01                                                                                                                                                                               | IXCREATOR.IXNAME.COLNAME                                        |
|                             |                      |          |                                                                                                                                                                                        |                                                                 |
| SYSRELS                     | 1097                 | DSNDLX01 | REFTBCREATOR.REFTBNAME                                                                                                                                                                 |                                                                 |
| SYSSYNONYMS                 | 1117                 | DSNDYX01 | CREATOR.NAME                                                                                                                                                                           |                                                                 |
| SYSTABAUTH                  | 1118                 | DSNATX01 | GRANTOR                                                                                                                                                                                |                                                                 |
|                             |                      | DSNATX02 | GRANTEE.TCREATOR.TTNAME.<br>GRANTEEETYPE.UPDATECOLS.<br>ALTERAUTH.DELETEAUTH.<br>INDEXAUTH.INSERTAUTH.<br>SELECTAUTH.UPDATEAUTH.<br>CAPTUREAUTH.REFERENCESAUTH.<br>REFCOLS.TRIGGERAUTH |                                                                 |
|                             |                      | DSNATX03 | GRANTEE.GRANTEEETYPE.COLLID<br>CONTOKEN                                                                                                                                                |                                                                 |
|                             | SYSTABLEPART         | 1121     | DSNDPX01                                                                                                                                                                               | DBNAME.TSNAME.PARTITION                                         |
|                             |                      |          | DSNDPX02                                                                                                                                                                               | STORNAME                                                        |
|                             | SYSTABLES            | 1126     | DSNDTX01                                                                                                                                                                               | CREATOR.NAME                                                    |
|                             |                      |          | DSNDTX02                                                                                                                                                                               | DBID.OBID.CREATOR.NAME                                          |
|                             | SYSTABLESPACE        | 1131     | DSNDSX01                                                                                                                                                                               | DBNAME.NAME                                                     |
|                             |                      |          |                                                                                                                                                                                        |                                                                 |
| SYSDBAUT                    | SYSDATABASE          | 1044     | DSNDDH01                                                                                                                                                                               | NAME                                                            |
|                             |                      |          | DSNDDX02                                                                                                                                                                               | GROUP_MEMBER                                                    |
|                             | SYSDBAUTH            | 1047     | DSNADH01                                                                                                                                                                               | GRANTEE.NAME                                                    |
|                             |                      |          | DSNADX01                                                                                                                                                                               | GRANTOR.NAME                                                    |
|                             |                      |          |                                                                                                                                                                                        |                                                                 |
| SYSDDF                      | IPNAMES              | 1016     | DSNFPX01                                                                                                                                                                               | LINKNAME                                                        |
|                             | LOCATIONS            | 1017     | DSNFCX01                                                                                                                                                                               | LOCATION                                                        |
|                             | LULIST               | 1018     | DSNFLX01                                                                                                                                                                               | LINKNAME.LUNAME                                                 |
|                             |                      |          | DSNFLX02                                                                                                                                                                               | LUNAME                                                          |
|                             | LUMODES              | 1019     | DSNFMX01                                                                                                                                                                               | LUNAME.MODENAME                                                 |
|                             | LUNAMES              | 1020     | DSNFNX01                                                                                                                                                                               | LUNAME                                                          |
|                             | MODESELECT           | 1022     | DSNFDX01                                                                                                                                                                               | LUNAME.AUTHID <sup>1</sup> .PLANNNAME <sup>1</sup>              |
|                             | USERNAMES            | 1143     | DSNFEX01                                                                                                                                                                               | TYPE.AUTHID <sup>1</sup> .LINKNAME <sup>1</sup>                 |

## DB2 Catalog Tables

| TABLE SPACE<br>DSNDB06. ... | TABLE<br>SYSIBM. ... | Page | INDEX<br>SYSIBM. ... | INDEX FIELDS                               |
|-----------------------------|----------------------|------|----------------------|--------------------------------------------|
| SYSGPAUT                    | SYSRESAUTH           | 1098 | DSNAGH01             | GRANTEE.QUALIFIER.<br>NAME.OBTYPE          |
|                             |                      |      | DSNAGX01             | GRANTOR.QUALIFIER.<br>NAME.OBTYPE          |
| SYSGROUP                    | SYSSTOGROUP          | 1114 | DSNSSH01             | NAME                                       |
|                             | SYSVOLUMES           | 1142 |                      |                                            |
| SYSGRTNS                    | SYSROUTINES_OPTS     | 1106 | DSNROX01             | SCHEMA.ROUTINENAME.<br>BUILDDATE.BUILDTIME |
|                             | SYSROUTINES_SRC      | 1107 | DSNRSX01             | ROUTINENAME                                |
|                             |                      |      | DSNRSX02             | SCHEMA.ROUTINENAME.<br>BUILDDATE.<br>SEQNO |
| SYSHIST                     | SYSCOLDIST_HIST      | 1029 | DSNHFX01             | TBOWNER.TBNAME.<br>NAMESTATSTIME           |
|                             | SYSCOLUMNS_HIST      | 1037 | DSNHEX01             | TBCREATOR.TBNAME.<br>NAMESTATSTIME         |
|                             | SYSINDEXES_HIST      | 1057 | DSNHHX01             | TBCREATOR.TBNAME.<br>NAMESTATSTIME         |
|                             |                      |      | DSNHHX02             | CREATOR.NAME                               |
|                             | SYSINDEXPART_HIST    | 1061 | DSNHGX01             | IXCREATOR.IXNAME.<br>PARTITIONSTATSTIME    |
|                             | SYSINDEXSTATS_HIST   | 1063 | DSNHIX01             | OWNER.NAME.<br>PARTITIONSTATSTIME          |
|                             | SYSLOBSTATS_HIST     | 1072 | DSNHJX01             | DBNAME.NAMESTATSTIME                       |
|                             | SYSTABLEPART_HIST    | 1124 | DSNHCX01             | DBNAME.TSNAME.<br>PARTITIONSTATSTIME       |
|                             | SYSTABLES_HIST       | 1130 | DSNHDX01             | CREATOR.NAMESTATSTIME                      |
|                             | SYTABSTATS_HIST      | 1135 | DSNHBX01             | OWNER.NAME.<br>PARTITIONSTATSTIME          |
| SYSJAVA                     | SYSJARCONTENTS       | 1065 | DSNJCX01             | JARSCHEMA.JAR_ID                           |
|                             | SYSJAROBJECTS        | 1067 | DSNJOX01             | JARSCHEMA.JAR_ID                           |
|                             | SYSJAVAOPTS          | 1068 | DSNJVX01             | JARSCHEMA.JAR_ID                           |
| SYSJAUXA LOB                | SYSJARDATA           | 1066 | DSNJDX01             | JAR_DATA                                   |
| SYSJAXB LOB                 | SYSJARCLASS_SOURCE   | 1064 | DSNJSX01             | CLASS_SOURCE                               |
| SYSOBJ                      | SYSAUXRELS           | 1023 | DSNOXX01             | TBOWNER.TBNAME                             |
|                             |                      |      | DSNOXX02             | AUXTBOWNER.AUXTBNAME                       |
|                             | SYSCONSTDEP          | 1039 | DSNCCX01             | BSCHEMA.BNAME.BTYPE                        |
|                             |                      |      | DSNCCX02             | DTBCREATOR.DTBNAME                         |
|                             | SYSDATATYPES         | 1046 | DSNODX01             | SCHEMA.NAME                                |
|                             |                      |      | DSNODX02             | DATATYPEID <sup>1</sup>                    |

| TABLE SPACE<br>DSNDB06. ... | TABLE<br>SYSIBM. ... | Page | INDEX<br>SYSIBM. ... | INDEX FIELDS                                                                                                                                                                                                                                                                                        |
|-----------------------------|----------------------|------|----------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                             | SYSKEYCOLUSE         | 1069 | DSNCUX01             | TBCREATOR.TBNAME.<br>CONSTNAME.COLSEQ.                                                                                                                                                                                                                                                              |
|                             | SYSPARMS             | 1084 | DSNOPX01             | SCHEMA.SPECIFICNAME.<br>ROUTINETYPE.ROWTYPE<br>ORDINAL                                                                                                                                                                                                                                              |
|                             |                      |      | DSNOPX02             | TYPESCHEMA.TYPENAME.<br>ROUTINETYPE.CAST_FUNCTION.<br>OWNER.SCHEMA.SPECIFICNAME                                                                                                                                                                                                                     |
|                             |                      |      | DSNOPX03             | TYPESCHEMA.TYPENAME                                                                                                                                                                                                                                                                                 |
|                             | SYSROUTINEAUTH       | 1099 | DSNOAX01             | GRANTOR.SCHEMA.<br>SPECIFICNAME.ROUTINETYPE.<br>GRANTEETYPE.EXECUTEAUTH                                                                                                                                                                                                                             |
|                             |                      |      | DSNOAX02             | GRANTEE.SCHEMA.SPECIFICNAME.<br>ROUTINETYPE.GRANTEETYPE.<br>EXECUTEAUTH.GRANTEDTS                                                                                                                                                                                                                   |
|                             |                      |      | DSNOAX03             | SCHEMA.SPECIFICNAME<br>ROUTINETYPE                                                                                                                                                                                                                                                                  |
|                             | SYSROUTINES          | 1100 | DSNOFX01             | NAME.PARM_COUNT.<br>ROUTINETYPE.PARM_SIGNATURE.<br>SCHEMA.PARM1.PARM2.PARM3.<br>PARM4.PARM5.PARM6.PARM7.<br>PARM8.PARM9.PARM10.PARM11.<br>PARM12.PARM13.PARM14.PARM15.<br>PARM16.PARM17.PARM18.PARM19.<br>PARM20.PARM21.PARM22.PARM23.<br>PARM24.PARM25.PARM26.PARM27.<br>PARM28.PARM29.PARM30      |
|                             |                      |      | DSNOFX02             | SCHEMA.SPECIFICNAME.<br>ROUTINETYPE                                                                                                                                                                                                                                                                 |
|                             |                      |      | DSNOFX03             | NAME.SCHEMA.CAST_FUNCTION.<br>PARM_COUNT.PARM_SIGNATURE.<br>PARM1                                                                                                                                                                                                                                   |
|                             |                      |      | DSNOFX04             | ROUTINE_ID <sup>1</sup>                                                                                                                                                                                                                                                                             |
|                             |                      |      | DSNOFX05             | SOURCESCHEMA.SOURCESPECIFIC.<br>ROUTINETYPE                                                                                                                                                                                                                                                         |
|                             |                      |      | DSNOFX06             | SCHEMA.NAME.ROUTINETYPE.<br>PARM_COUNT                                                                                                                                                                                                                                                              |
|                             |                      |      | DSNOFX07             | NAME.PARM_COUNT.<br>ROUTINETYPE. SCHEMA.<br>PARM_SIGNATURE.<br>PARM1.PARM2.PARM3.<br>PARM4.PARM5.PARM6.PARM7.<br>PARM8.PARM9.PARM10.PARM11.<br>PARM12.PARM13.PARM14.PARM15.<br>PARM16.PARM17.PARM18.PARM19.<br>PARM20.PARM21.PARM22.PARM23.<br>PARM24.PARM25.PARM26.PARM27.<br>PARM28.PARM29.PARM30 |
|                             |                      |      | DSNOFX08             | JARSCHEMA.<br>JAR_ID                                                                                                                                                                                                                                                                                |
|                             | SYSSCHEMAAUTH        | 1108 | DSNSKX01             | GRANTEE.SCHEMA_NAME                                                                                                                                                                                                                                                                                 |
|                             |                      |      | DSNSKX02             | GRANTOR                                                                                                                                                                                                                                                                                             |
|                             | SYSTABCONST          | 1120 | DSNCNX01             | TBCREATOR.TBNAME.CONSTNAME                                                                                                                                                                                                                                                                          |
|                             |                      |      | DSNCNX02             | IXOWNER.IXNAME                                                                                                                                                                                                                                                                                      |
|                             | SYSTRIGGERS          | 1136 | DSNOTX01             | SCHEMA.NAME.SEQNO                                                                                                                                                                                                                                                                                   |
|                             |                      |      | DSNOTX02             | TBOWNER.TBNAME                                                                                                                                                                                                                                                                                      |

## DB2 Catalog Tables

| TABLE SPACE<br>DSNDB06. ... | TABLE<br>SYSIBM. ... | Page | INDEX<br>SYSIBM. ... | INDEX FIELDS                                                       |
|-----------------------------|----------------------|------|----------------------|--------------------------------------------------------------------|
|                             |                      |      | DSNOTX03             | SCHEMA.TRIGNAME                                                    |
| SYSPKAGE                    | SYSPACKAGE           | 1073 | DSNKKX01             | LOCATION.COLLID.NAME.<br>VERSION                                   |
|                             |                      |      | DSNKKX02             | LOCATION.COLLID.NAME.<br>CONTOKEN                                  |
|                             | SYSPACKAUTH          | 1078 | DSNKAX01             | GRANTOR.LOCATION.COLLID.NAME                                       |
|                             |                      |      | DSNKAX02             | GRANTEE.LOCATION.COLLID.<br>NAME.BINDAUTH.COPYAUTH.<br>EXECUTEAUTH |
|                             |                      |      | DSNKAX03             | LOCATION.COLLID.NAME                                               |
|                             | SYSPACKDEP           | 1079 | DSNKDX01             | DLOCATION.DCOLLID.DNAME.<br>DCONTOKEN                              |
|                             |                      |      | DSNKDX02             | BQUALIFIER.BNAME.BTYPE                                             |
|                             |                      |      | DSNKDX03             | BQUALIFIER.BNAME.BTYPE.<br>DTYPE                                   |
|                             | SYSPROCEDURES        | 1094 | DSNKCX01             | PROCEDURE.AUTHID <sup>1</sup> .LUNAME <sup>1</sup>                 |
|                             | SYSPACKLIST          | 1080 | DSNKLX01             | LOCATION.COLLID.NAME                                               |
|                             |                      |      | DSNKLX02             | PLANNAMSEQNO.LOCATION.<br>COLLID.NAME                              |
|                             | SYSPACKSTMT          | 1081 | DSNKSX01             | LOCATION.COLLID.NAME.<br>CONTOKEN.SEQNO                            |
|                             | SYSPKSYSTEM          | 1086 | DSNKYX01             | LOCATION.COLLID.NAME.<br>CONTOKEN.SYSTEM.ENABLE                    |
|                             | SYSPLSYSTEM          | 1093 | DSNKPX01             | NAME.SYSTEM.ENABLE                                                 |
| SYSPLAN                     | SYSDBRM              | 1049 |                      |                                                                    |
|                             | SYSPLAN              | 1087 | DSNPPH01             | NAME                                                               |
|                             | SYSPLANAUTH          | 1091 | DSNAPH01             | GRANTEE.NAME.EXECUTEAUTH                                           |
|                             |                      |      | DSNAPX01             | GRANTOR                                                            |
|                             | SYSPLANDEP           | 1092 | DSNGGX01             | BCREATOR.BNAME.BTYPE                                               |
|                             | SYSSTMT              | 1111 |                      |                                                                    |
| SYSSEQ                      | SYSSEQUENCES         | 1109 | DSNSQX01             | SCHEMA.NAME                                                        |
|                             |                      |      | DSNSQX02             | SEQUENCEID <sup>1</sup>                                            |
| SYSSEQ2                     | SYSSEQUENCESDEP      | 1110 | DSNSRX01             | DCREATOR.DNAME.DCOLNAME                                            |
| SYSSTATS                    | SYSCOLDIST           | 1028 | DSNTNX01             | TBOWNER.TBNAME.NAME                                                |
|                             | SYSCOLDISTSTATS      | 1030 | DSNTPX01             | TBOWNER.TBNAME.NAME<br>PARTITION                                   |
|                             | SYSCOLSTATS          | 1031 | DSNTCX01             | TBOWNER.TBNAME.NAME<br>PARTITION                                   |
|                             | SYSINDEXSTATS        | 1062 | DSNTXX01             | OWNER.NAME.PARTITION                                               |
|                             | SYSLOBSTATS          | 1071 | DSNLNX01             | DBNAME.NAME                                                        |
|                             | SYSTABSTATS          | 1134 | DSNTTX01             | OWNER.NAME.PARTITION                                               |
| SYSSTR                      | SYSCHECKDEP          | 1024 | DSNSDX01             | TBOWNER.TBNAME.CHECKNAME<br>COLNAME                                |
|                             | SYSCHECKS            | 1025 | DSNSCX01             | TBOWNER.TBNAME.CHECKNAME                                           |
|                             | SYSCHECKS2           | 1026 | DSNCHX01             | TBOWNER.TBNAME.CHECKNAME                                           |

| TABLE SPACE<br>DSNDB06. ... | TABLE<br>SYSIBM. ... | Page | INDEX<br>SYSIBM. ... | INDEX FIELDS             |
|-----------------------------|----------------------|------|----------------------|--------------------------|
|                             | SYSSTRINGS           | 1115 | DSNSSX01             | OUTCCSID.INCCSID.IBMREQD |
| SYSUSER                     | SYSUSERAUTH          | 1137 | DSNAUH01             | GRANTEE.GRANTEDTS        |
|                             |                      |      | DSNAUX02             | GRANTOR                  |
| SYSVIEWS                    | SYSVIEWDEP           | 1140 | DSNGGX02             | BCREATOR.BNAME.BTYPE     |
|                             |                      |      | DSNGGX03             | BSHEMA.BNAME.BTYPE       |
| I                           | SYSVIEWS             | 1141 | DSNVVX01             | CREATOR.NAME.SEQNO.TYPE  |

**Note:**

1. Index field is in descending order

## SQL statements allowed on the catalog

The following SQL statements can be used to change the value of certain options for existing catalog indexes and table spaces, and to add indexes to any of the catalog tables.

| SQL statement | Index       | Allowable clauses and usage notes                                                                                                                                                                                                                       |
|---------------|-------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| ALTER INDEX   | IBM-defined | <p>Only these clauses are allowed:</p> <p>CLOSE<br/>FREEPAGE<br/>GBPCACHE<br/>PCTFREE<br/>PIECESIZE<br/>COPY</p> <p>You cannot alter the GBPCACHE value for indexes DSNDXX01, DSNDXX02, and DSNDXX03, which are on catalog table SYSIBM.SYSINDEXES.</p> |
| ALTER TABLE   |             | The only clause allowed is DATA CAPTURE CHANGES.                                                                                                                                                                                                        |

## DB2 Catalog Tables

| SQL statement    | Index        | Allowable clauses and usage notes                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
|------------------|--------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| ALTER TABLESPACE |              | <p>Only these clauses are allowed:</p> <p>CLOSE<br/>FREEPAGE<br/>GBPCACHE<br/>LOCKMAX<br/>MAXROWS<br/>PCTFREE<br/>TRACKMOD</p> <p>You cannot alter the GBPCACHE or MAXROWS value of some catalog table spaces. Do not specify GBPCACHE for the following table spaces:</p> <ul style="list-style-type: none"> <li>• DSNDB06.SYSDBASE</li> <li>• DSNDB06.SYSDBAUT</li> <li>• DSNDB06.SYSPKAGE</li> <li>• DSNDB06.SYSPLAN</li> <li>• DSNDB06.SYSUSER (exception: the attribute can be altered if authorization includes installation SYSADM authority.)</li> </ul> <p>Do not specify MAXROWS for the following table spaces:</p> <ul style="list-style-type: none"> <li>• DSNDB06.SYSDBASE</li> <li>• DSNDB06.SYSDBAUT</li> <li>• DSNDB06.SYSGROUP</li> <li>• DSNDB06.SYSPLAN</li> <li>• DSNDB06.SYSVIEWS</li> </ul> <p>You can specify the LOCKSIZE keyword on the ALTER TABLESPACE statement for any catalog table spaces that are not LOB table spaces and that do not contain links. The following table spaces contain links:</p> <ul style="list-style-type: none"> <li>• DSNDB06.SYSDBASE</li> <li>• DSNDB06.SYSDBUT</li> <li>• DSNDB06.SYSGROUP</li> <li>• DSNDB06.SYSPLAN</li> <li>• DSNDB06.SYSVIEWS</li> </ul> |
| CREATE INDEX     | User-created | <p>All clauses are allowed, except for:</p> <p>CLOSE YES<br/>CLUSTER<br/>UNIQUE<br/>DEFER YES (only on tables SYSINDEXES, SYSINDEXPART, and SYSKEYS)</p> <p>The only value allowed for BUFFERPOOL is BP0.</p> <p>You can create up to 100 indexes on the catalog.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| ALTER INDEX      | User-created | All clauses are allowed, except for BUFFERPOOL.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| DROP INDEX       | User-created | The statement has no clauses.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |

## Reorganizing the catalog

The REORG TABLESPACE utility can be run on all the table spaces in the catalog database (DSNDB06) to reclaim unused or wasted space, which can affect performance. The utility observes the PCTFREE and FREEPAGE values specified in the ALTER INDEX statement for all the catalog indexes and the following table spaces:

- DSNDB06.SYSCOPY
- DSNDB06.SYSDDF
- DSNDB06.SYSGPAUT
- DSNDB06.SYSGRTNS
- DSNDB06.SYSHIST
- DSNDB06.SYSJAVA
- DSNDB06.SYSJAUXA
- DSNDB06.SYSJAUXB
- DSNDB06.SYSOBJ
- DSNDB06.SYSPKAGE
- DSNDB06.SYSSEQ
- DSNDB06.SYSEQ2
- DSNDB06.SYSSTR
- DSNDB06.SYSSTATS
- DSNDB06.SYSUSER
- DSNDB01.SCT02
- DSNDB01.SPT01

For details on running REORG TABLESPACE, see *DB2 Utility Guide and Reference*.

### New and changed catalog tables

Descriptions of the following catalog tables have been added:

- SYSIBM.SYSCHECK2
- SYSIBM.SYSCOLDIST\_HIST
- SYSIBM.SYSCOLUMNS\_HIST
- SYSIBM.SYSINDEXES\_HIST
- SYSIBM.SYSINDEXPART\_HIST
- SYSIBM.SYSINDEXSTATS\_HIST
- SYSIBM.SYSJARCLASS\_SOURCE
- SYSIBM.SYSJARCONTENTS
- SYSIBM.SYSJARDATA
- SYSIBM.SYSJAROBJECTS
- SYSIBM.SYSJAVAOPTS
- SYSIBM.SYSKEYCOLUSE
- SYSIBM.SYSLOBSTATS\_HIST
- SYSIBM.SYSROUTINES\_OPTS
- SYSIBM.SYSROUTINES\_SRC
- SYSIBM.SYSTABCONST
- SYSIBM.SYSTABLEPART\_HIST
- SYSIBM.SYSTABLES\_HIST
- SYSIBM.SYSTABSTATS\_HIST

The following tables have new or revised columns, column values, or column descriptions to support the new function in DB2 Version 7:

| Table name   | New column                                                              | Revised column  |
|--------------|-------------------------------------------------------------------------|-----------------|
| LUNAMES      |                                                                         | SYSMODENAME     |
| SYSCOPY      | COPYPAGESF<br>NPAGESF<br>CPAGESF<br>JOBNAME<br>AUTHID                   |                 |
| SYSDATABASE  |                                                                         | ENCODING_SCHEME |
| SYSDATATYPES |                                                                         | ENCODING_SCHEME |
| SYSDBRM      |                                                                         | DEC31           |
| SYSINDEXES   | SPACEF<br>REMARKS                                                       |                 |
| SYSINDEXPART | SPACEF<br>DSNUM<br>EXTENTS<br>PSEUDO_DEL_ENTRIES<br>LEAFNEAR<br>LEAFFAR |                 |
| SYSPACKAGE   | ENCODING_CCSID<br>IMMEDWRITE<br>RELBOUND                                | DEC31           |
| SYSPACKSTMT  | EXPLAINABLE<br>QUERYNO                                                  |                 |
| SYSPARMS     |                                                                         | ENCODING_SCHEME |
| SYSPLAN      | ENCODING_CCSID<br>IMMEDWRITE<br>RELBOUND                                |                 |
| SYSPLANDEP   |                                                                         | BTYPE           |

| Table name    | New column                                                     | Revised column                 |
|---------------|----------------------------------------------------------------|--------------------------------|
| SYSROUTINES   | JAVA_SIGNATURE<br>CLASS<br>JARSCHEMA<br>JAR_ID<br>SPECIAL_REGS | ASUTIME<br>WLM_ENVIRONMENT     |
| SYSSEQUENCES  |                                                                | MAXVALUE<br>MINVALUE<br>CYCLE  |
| SYSSTMT       | EXPLAINABLE<br>QUERYNO                                         |                                |
| SYSTABLEPART  | SPACEF<br>DSNUM<br>EXTENTS                                     |                                |
| SYSTABLES     | NPAGESF<br>SPACEF<br>AVGROWLEN<br>RELCREATED                   | ENCODING_SCHEME<br>TABLESTATUS |
| SYSTABLESPACE |                                                                | ENCODING_SCHEME                |
| SYSTRIGGERS   | TRIGNAME                                                       |                                |
| SYSVIEWDEP    | DTYPE                                                          |                                |
| SYSVIEWS      | RELCREATED<br>TYPE                                             |                                |

## SYSIBM.IPNAMES

### SYSIBM.IPNAMES table

Defines the remote DRDA servers DB2 can access using TCP/IP. Rows in this table can be inserted, updated, and deleted.

| Column name  | Data type                                | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 | Use |
|--------------|------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| LINKNAME     | CHAR(8)<br>NOT NULL                      | The value specified in this column must match the value specified in the LINKNAME column of the associated row in SYSIBM.LOCATIONS.                                                                                                                                                                                                                                                                                                                                                                         | G   |
| SECURITY_OUT | CHAR(1)<br>NOT NULL WITH<br>DEFAULT 'A'  | This column defines the DRDA security option that is used when local DB2 SQL applications connect to any remote server associated with this TCP/IP host:                                                                                                                                                                                                                                                                                                                                                    | G   |
|              | A                                        | The option is “already verified”. Outbound connection requests contain an authorization ID and no password. The authorization ID used for an outbound request is either the DB2 user’s authorization ID or a translated ID, depending upon the value of the USERNAMES column.                                                                                                                                                                                                                               |     |
|              | R                                        | The option is “RACF PassTicket”. Outbound connection requests contain a userid and a RACF PassTicket. The value specified in the LINKNAME column is used as the RACF PassTicket application name for the remote server.                                                                                                                                                                                                                                                                                     |     |
|              | P                                        | The authorization ID used for an outbound request is either the DB2 user’s authorization ID or a translated ID, depending upon the value of the USERNAMES column.                                                                                                                                                                                                                                                                                                                                           |     |
|              | O                                        | The option is “password”. Outbound connection requests contain an authorization ID and a password. The password is obtained from the SYSIBM.USERNAMES table.                                                                                                                                                                                                                                                                                                                                                |     |
|              | blank                                    | The USERNAMES column must specify “O”.                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |     |
| USERNAMES    | CHAR(1)<br>NOT NULL WITH<br>DEFAULT      | This column controls outbound authorization ID translation. Outbound translation is performed when an authorization ID is sent by DB2 to a remote server.                                                                                                                                                                                                                                                                                                                                                   | G   |
|              | O                                        | An outbound ID is subject to translation. Rows in the SYSIBM.USERNAMES table are used to perform ID translation.                                                                                                                                                                                                                                                                                                                                                                                            |     |
|              | blank                                    | No translation or “come from” checking is performed on inbound IDs.                                                                                                                                                                                                                                                                                                                                                                                                                                         |     |
| IBMREQD      | CHAR(1)<br>NOT NULL WITH<br>DEFAULT 'N'  | A value of Y indicates that the row came from the basic machine-readable material (MRM) tape. For all other values, see “Release dependency indicators” on page 1005.                                                                                                                                                                                                                                                                                                                                       | G   |
| IPADDR       | VARCHAR(254)<br>NOT NULL WITH<br>DEFAULT | This column contains the IP address or domain name of a remote TCP/IP host. The IPADDR column must be specified as follows:                                                                                                                                                                                                                                                                                                                                                                                 | G   |
|              |                                          | <ul style="list-style-type: none"> <li>• If the IPADDR contains a left justified character string containing four numeric values delimited by decimal points, DB2 assumes the value is an IP address in dotted decimal format. For example, ‘123.456.78.91’ would be interpreted as a dotted decimal IP address.</li> <li>• All other values are interpreted as a TCP/IP domain name, which can be resolved by the TCP/IP gethostbyname socket call. TCP/IP domain names are not case sensitive.</li> </ul> |     |

**SYSIBM.LOCATIONS table**

Contains a row for every accessible remote server. The row associates a LOCATION name with the TCP/IP or SNA network attributes for the remote server. Requesters are not defined in this table. Rows in this table can be inserted, updated, and deleted.

| Column name | Data type                               | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     | Use |
|-------------|-----------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| LOCATION    | CHAR(16)<br>NOT NULL                    | A unique location name for the accessible server. This is the name by which the remote server is known to local DB2 SQL applications.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           | G   |
| LINKNAME    | CHAR(8)<br>NOT NULL                     | Identifies the VTAM or TCP/IP attributes associated with this location. For any LINKNAME specified, one or both of the following statements must be true: <ul style="list-style-type: none"> <li>• A row exists in SYSIBM.LUNAMES whose LUNAME matches the value specified in the SYSIBM.LOCATIONS LINKNAME column. This row specifies the VTAM communication attributes for the remote location.</li> <li>• A row exists in SYSIBM.IPNAMES whose LINKNAME matches the value specified in the SYSIBM.LOCATIONS LINKNAME column. This row specifies the TCP/IP communication attributes for the remote location.</li> </ul>                                                                                                                                                                                                                                                                                                                                                                                      | G   |
| IBMREQD     | CHAR(1)<br>NOT NULL WITH<br>DEFAULT 'N' | A value of Y indicates that the row came from the basic machine-readable material (MRM) tape. For all other values, see "Release dependency indicators" on page 1005.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           | G   |
| PORT        | CHAR(32)<br>NOT NULL WITH<br>DEFAULT    | TCP/IP is used for outbound DRDA connections when the following statement is true: <ul style="list-style-type: none"> <li>• A row exists in SYSIBM.IPNAMES, where the LINKNAME column matches the value specified in the SYSIBM.LOCATIONS LINKNAME column.</li> </ul> <p>If the above mentioned row is found, the value of the PORT column is interpreted as follows:</p> <ul style="list-style-type: none"> <li>• If PORT is blank, the default DRDA port (446) is used.</li> <li>• If PORT is nonblank, the value specified for PORT can take one of two forms:               <ul style="list-style-type: none"> <li>– If the value in PORT is left justified with 1-5 numeric characters, the value is assumed to be the TCP/IP port number of the remote database server.</li> <li>– Any other value is assumed to be a TCP/IP service name, which can be converted to a TCP/IP port number using the TCP/IP getservbyname socket call. TCP/IP service names are not case sensitive.</li> </ul> </li> </ul> | G   |
| TPN         | VARCHAR(64)<br>NOT NULL WITH<br>DEFAULT | Used only when the local DB2 begins an SNA conversation with another server. When used, TPN indicates the SNA LU 6.2 transaction program name (TPN) that will allocate the conversation. A length of zero for the column indicates the default TPN. For DRDA conversations, this is the DRDA default, which is X'07F6C4C2'. For DB2 private protocol conversations, this column is not used.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    | G   |
|             |                                         | For an SQL/DS™ server, TPN should contain the resource ID of the SQL/DS machine.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |     |

---

**SYSIBM.LULIST table**

Allows multiple LU names to be specified for a given LOCATION. Insert rows into this table when you want to define a remote DB2 data sharing group. The same value for LUNAME column cannot appear in both the SYSIBM.LUNAMES table and the SYSIBM.LULIST table. Rows in this table can be inserted, updated, and deleted.

| Column name | Data type                               | Description                                                                                                                                                                                                                                                                                                        | Use |
|-------------|-----------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| LINKNAME    | CHAR(8)<br>NOT NULL                     | The value of the LINKNAME column in the SYSIBM.LOCATIONS table with which this row is associated. This is also the value of the LUNAME column in the SYSIBM.LUNAMES table. The values of the other columns in the SYSIBM.LUNAMES row apply to the LU identified by the LUNAME column in this row of SYSIBM.LULIST. | G   |
| LUNAME      | CHAR(8)<br>NOT NULL                     | The VTAM® logical unit name (LUNAME) of the remote database system. This LUNAME must not exist in the LUNAME column of SYSIBM.LUNAMES.                                                                                                                                                                             | G   |
| IBMREQD     | CHAR(1)<br>NOT NULL WITH<br>DEFAULT 'N' | A value of Y indicates that the row came from the basic machine-readable material (MRM) tape. For all other values, see "Release dependency indicators" on page 1005.                                                                                                                                              | G   |

---

## SYSIBM.LUMODES table

Each row of the table provides VTAM with conversation limits for a specific combination of LUNAME and MODENAME. The table is accessed only during the initial conversation limit negotiation between DB2 and a remote LU. This negotiation is called *change-number-of-sessions* (CNOS) processing. Rows in this table can be inserted, updated, and deleted.

| Column name | Data type                               | Description                                                                                                                                                                                                | Use |
|-------------|-----------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| LUNAME      | CHAR(8)<br>NOT NULL                     | LU name of the server involved in the CNOS processing.                                                                                                                                                     | G   |
| MODENAME    | CHAR(8)<br>NOT NULL                     | Name of a logon mode description in the VTAM logon mode table.                                                                                                                                             | G   |
| CONVLIMIT   | SMALLINT<br>NOT NULL                    | Maximum number of active conversations between the local DB2 and the other system for this mode. Used to override the number in the DSESLIM parameter of the VTAM APPL definition statement for this mode. | G   |
| IBMREQD     | CHAR(1)<br>NOT NULL WITH<br>DEFAULT 'N' | A value of Y indicates that the row came from the basic machine-readable material (MRM) tape. For all other values, see "Release dependency indicators" on page 1005.                                      | G   |

## SYSIBM.LUNAMES

### SYSIBM.LUNAMES table

The table must contain a row for each remote SNA client or server that communicates with DB2. Rows in this table can be inserted, updated, and deleted.

| Column name                                     | Data type                               | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      | Use |
|-------------------------------------------------|-----------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| LUNAME                                          | CHAR(8)<br>NOT NULL                     | Name of the LU for one or more accessible systems. A blank string indicates the row applies to clients whose LU name is not specifically defined in this table.<br><br>All other column values for a given row in this table are for clients and servers associated with the row's LU name.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      | G   |
| # SYSMODENAME<br>#<br>#<br>#<br><br>#<br>#<br># | CHAR(8)<br>NOT NULL<br>WITH DEFAULT     | Mode used to establish inter-system conversations. A blank indicates the default mode IBMDB2LM (used for DB2 private protocol access and for collecting syplex balancing information from remote data sharing groups).<br><br>If private protocols are used to access a remote DB2 LU or if the remote LU is a member of a DB2 data sharing group, use a separate mode other than the default mode.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              | G   |
| SECURITY_IN                                     | CHAR(1)<br>NOT NULL WITH<br>DEFAULT 'A' | This column defines the security options that are accepted by this DB2 when an SNA client connects to DB2:<br><br><b>V</b> The option is "verify". An incoming connection request must include one of the following: a userid and password, a userid and RACF PassTicket, or a Kerberos security ticket.<br><br><b>A</b> The option is "already verified". A request does not need a password, although a password is checked if it is sent.<br><br>With this option, an incoming connection request is accepted if it includes any of the following: a userid, a userid and password, a userid and RACF PassTicket, or a Kerberos security ticket.<br><br>If the USERNAMES column contains 'I' or 'B', RACF is not invoked to validate incoming connection requests that contain only a userid.                                                                                                                                                                                                                                                                                                                                 | G   |
| SECURITY_OUT                                    | CHAR(1)<br>NOT NULL WITH<br>DEFAULT 'A' | This column defines the security option that is used when local DB2 SQL applications connect to any remote server associated with this LUNAME:<br><br><b>A</b> The option is "already verified". Outbound connection requests contain an authorization ID and no password.<br><br>The authorization ID used for an outbound request is either the DB2 user's authorization ID or a translated ID, depending upon the value of the USERNAMES column.<br><br><b>R</b> The option is "RACF PassTicket". Outbound connection requests contain a userid and a RACF PassTicket. The server's LU name is used as the RACF PassTicket application name.<br><br>The authorization ID used for an outbound request is either the DB2 user's authorization ID or a translated ID, depending upon the value of the USERNAMES column.<br><br><b>P</b> The option is "password". Outbound connection requests contain an authorization ID and a password. The password is obtained from the SYSIBM.USERNAMES table or RACF, depending upon the value specified in the ENCRYPTPWDS column.<br><br>The USERNAMES column must specify 'B' or 'O'. | G   |

## SYSIBM.LUNAMES

| Column name     | Data type                               | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    | Use |
|-----------------|-----------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| ENCRYPTIONPSWDS | CHAR(1)<br>NOT NULL WITH<br>DEFAULT 'N' | <p>This column only applies to DB2 for OS/390 and z/OS partners. It is provided to support connectivity to prior releases of DB2 that are unable to support RACF PassTickets.</p> <p>For connections between DB2 Version 5 and later, we recommend using the SECURITY_OUT='R' option instead of the ENCRYPTIONPSWDS='Y' option.</p> <p><b>N</b> No, passwords are not in internal RACF encrypted format. This is the default.</p> <p><b>Y</b> Yes for outbound requests, the encrypted password is extracted from RACF and sent to the server. For inbound requests, the password is treated as encrypted.</p>                                                                                                                                                                                                                                                                                                                 | G   |
| MODESELECT      | CHAR(1)<br>NOT NULL WITH<br>DEFAULT 'N' | <p>Whether to use the SYBIBM.MODESELECT table:</p> <p><b>N</b> Use default modes: IBMDB2LM (for DB2 private protocol) and IBMRDB (for DRDA).</p> <p><b>Y</b> Searches SYSIBM.MODESELECT for appropriate mode name.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         | G   |
| USERNAMES       | CHAR(1)<br>NOT NULL WITH<br>DEFAULT     | <p>This column controls inbound and outbound authorization ID translation, and “come from” checking.</p> <p>Inbound translation and “come from” checking are performed when an authorization ID is received from a remote client.</p> <p>Outbound translation is performed when an authorization ID is sent by DB2 to a remote server.</p> <p>When I, O, or B is specified in this column, rows in the SYSIBM.USERNAMES table are used to perform ID translation.</p> <p><b>I</b> An inbound ID is subject to translation and “come from” checking.</p> <p>No translation is performed on outbound IDs.</p> <p><b>O</b> No translation or “come from” checking is performed on inbound IDs.</p> <p>An outbound ID is subject to translation.</p> <p><b>B</b> An inbound ID is subject to translation and “come from” checking.</p> <p>An outbound ID is subject to translation.</p> <p><b>blank</b> No translation occurs.</p> | G   |
| GENERIC         | CHAR(1)<br>NOT NULL WITH<br>DEFAULT 'N' | <p>Indicates whether DB2 should use its real LU name or generic LU name to identify itself to the partner LU, which is identified by this row.</p> <p><b>N</b> The real VTAM LU name of this DB2 subsystem</p> <p><b>Y</b> The VTAM generic LU name of this DB2 subsystem</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  | G   |
| IBMREQD         | CHAR(1)<br>NOT NULL WITH<br>DEFAULT 'N' | A value of Y indicates that the row came from the basic machine-readable material (MRM) tape. For all other values, see “Release dependency indicators” on page 1005.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          | G   |

**SYSIBM.MODESELECT table**

Associates a mode name with any conversation created to support an outgoing SQL request. Each row represents one or more combinations of LUNAME, authorization ID, and application plan name. Rows in this table can be inserted, updated, and deleted.

| Column name | Data type                               | Description                                                                                                                                                                                                     | Use |
|-------------|-----------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| AUTHID      | CHAR(8)<br>NOT NULL<br>WITH DEFAULT     | Authorization ID of the SQL request. Blank (the default) indicates that the MODENAME specified for the row is to apply to all authorization IDs.                                                                | G   |
| PLANNAME    | CHAR(8)<br>NOT NULL<br>WITH DEFAULT     | Plan name associated with the SQL request. Blank (the default) indicates that the MODENAME specified for the row is to apply to all plan names.                                                                 | G   |
| LUNAME      | CHAR(8)<br>NOT NULL                     | LU name associated with the SQL request.                                                                                                                                                                        | G   |
| MODENAME    | CHAR(8)<br>NOT NULL                     | Name of the logon mode in the VTAM logon mode table to be used in support of the outgoing SQL request. If blank, IBMDB2LM is used for DB2 private protocol connections and IBMRDB is used for DRDA connections. | G   |
| IBMRREQD    | CHAR(1)<br>NOT NULL WITH<br>DEFAULT 'N' | A value of Y indicates that the row came from the basic machine-readable material (MRM) tape. For all other values, see "Release dependency indicators" on page 1005.                                           | G   |

---

## SYSIBM.SYSAUXRELS table

Contains one row for each auxiliary table created for a LOB column. A base table space that is partitioned must have one auxiliary table for each partition of each LOB column.

| Column name | Data type               | Description                                                                                                                                                           | Use |
|-------------|-------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| TBOWNER     | CHAR(8)<br>NOT NULL     | Authorization ID of the owner of the base table.                                                                                                                      | G   |
| TBNAME      | VARCHAR(18)<br>NOT NULL | Name of the base table.                                                                                                                                               | G   |
| COLNAME     | VARCHAR(18)<br>NOT NULL | Name of the LOB column in the base table.                                                                                                                             | G   |
| PARTITION   | SMALLINT<br>NOT NULL    | Partition number if the base table space is partitioned. Otherwise, the value is 0.                                                                                   | G   |
| AUXTBOWNER  | CHAR(8)<br>NOT NULL     | Authorization ID of the owner of the auxiliary table.                                                                                                                 | G   |
| AUXTBNAME   | VARCHAR(18)<br>NOT NULL | Name of the auxiliary table.                                                                                                                                          | G   |
| AUXRELOBID  | INTEGER<br>NOT NULL     | Internal identifier of the relationship between the base table and the auxiliary table.                                                                               | S   |
| IBMREQD     | CHAR(1)<br>NOT NULL     | A value of Y indicates that the row came from the basic machine-readable material (MRM) tape. For all other values, see “Release dependency indicators” on page 1005. | G   |

## SYSIBM.SYSCHECKDEP

### SYSIBM.SYSCHECKDEP table

Contains one row for each reference to a column in a table check constraint.

| Column name | Data type                | Description                                                                                                                                                           | Use |
|-------------|--------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| TBOWNER     | CHAR(8)<br>NOT NULL      | Authorization ID of the owner of the table on which the table check constraint is defined.                                                                            | G   |
| TBNAME      | VARCHAR(18)<br>NOT NULL  | Name of the table on which the check constraint is defined.                                                                                                           | G   |
| CHECKNAME   | VARCHAR(128)<br>NOT NULL | Name of the check constraint.                                                                                                                                         | G   |
| COLNAME     | VARCHAR(18)<br>NOT NULL  | Name of the column that the table check constraint refers to.                                                                                                         | G   |
| IBMREQD     | CHAR(1)<br>NOT NULL      | A value of Y indicates that the row came from the basic machine-readable material (MRM) tape. For all other values, see “Release dependency indicators” on page 1005. | G   |

---

## SYSIBM.SYSCHECKS table

Contains one row for each table check constraint.

| Column name    | Data type                           | Description                                                                                                                                                           | Use |
|----------------|-------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| TBOWNER        | CHAR(8)<br>NOT NULL                 | Authorization ID of the owner of the table on which the table check constraint is defined.                                                                            | G   |
| CREATOR        | CHAR(8)<br>NOT NULL                 | Authorization ID of the creator of the table check constraint.                                                                                                        | G   |
| DBID           | SMALLINT<br>NOT NULL                | Internal identifier of the database for the table check constraint.                                                                                                   | S   |
| OBID           | SMALLINT<br>NOT NULL                | Internal identifier of the table check constraint.                                                                                                                    | S   |
| TIMESTAMP      | TIMESTAMP<br>NOT NULL               | Time when the table check constraint was created.                                                                                                                     | G   |
| RBA            | CHAR(6)<br>FOR BIT DATA<br>NOT NULL | The log RBA when the table check constraint was created.                                                                                                              | G   |
| IBMREQD        | CHAR(1)<br>NOT NULL                 | A value of Y indicates that the row came from the basic machine-readable material (MRM) tape. For all other values, see "Release dependency indicators" on page 1005. | G   |
| TBNAME         | VARCHAR(18)<br>NOT NULL             | Name of the table on which the check constraint is defined.                                                                                                           | G   |
| CHECKNAME      | VARCHAR(128)<br>NOT NULL            | Table check constraint name.                                                                                                                                          | G   |
| CHECKCONDITION | VARCHAR(3800)<br>NOT NULL           | Text of the table check constraint.                                                                                                                                   | G   |

## SYSIBM.SYSCHECKS2

---

### SYSIBM.SYSCHECKS2 table

# Contains one row for each table check constraint for catalog tables created in or  
# after Version 7. Check constraints for catalog tables created before Version 7 are  
# not included in this table.

| Column name | Data type                | Description                                                                                                                                                           | Use |
|-------------|--------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| TBOWNER     | CHAR(8)<br>NOT NULL      | Authorization ID of the owner of the table on which the table check constraint is defined.                                                                            | G   |
| TBNAME      | VARCHAR(18)<br>NOT NULL  | Name of the table on which the check constraint is defined.                                                                                                           | G   |
| CHECKNAME   | VARCHAR(128)<br>NOT NULL | Table check constraint name.                                                                                                                                          | G   |
| PATHSCHEMAS | VARCHAR(254)<br>NOT NULL | SQL path at the time the table check constraint was created. The path is used to resolve unqualified cast function names that are used in the constraint definition.  | G   |
| IBMRREQD    | CHAR(1)<br>NOT NULL      | A value of Y indicates that the row came from the basic machine-readable material (MRM) tape. For all other values, see "Release dependency indicators" on page 1005. | G   |

## SYSIBM.SYSCOLAUTH table

Records the UPDATE or REFERENCES privileges that are held by users on individual columns of a table or view.

| Column name | Data type                             | Description                                                                                                                                                                                                                      | Use |
|-------------|---------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| GRANTOR     | CHAR(8)<br>NOT NULL                   | Authorization ID of the user who granted the privileges. Could also be PUBLIC or PUBLIC followed by an asterisk <sup>45</sup> .                                                                                                  | G   |
| GRANTEE     | CHAR(8)<br>NOT NULL                   | Authorization ID of the user who holds the privilege or the name of an application plan or package that uses the privilege. PUBLIC for a grant to PUBLIC. PUBLIC followed by an asterisk for a grant to PUBLIC AT ALL LOCATIONS. | G   |
| GRANTEETYPE | CHAR(1)<br>NOT NULL                   | Type of grantee:<br><b>blank</b> An authorization ID<br><b>P</b> An application plan or a package. The grantee is a package if COLLID is not blank.                                                                              | G   |
| CREATOR     | CHAR(8)<br>NOT NULL                   | Authorization ID of the owner of the table or view on which the update privilege is held.                                                                                                                                        | G   |
| TNAME       | VARCHAR(18)<br>NOT NULL               | Name of the table or view.                                                                                                                                                                                                       | G   |
|             | CHAR(12)<br>NOT NULL                  | Internal use only                                                                                                                                                                                                                | I   |
|             | CHAR(6)<br>NOT NULL                   | Not used.                                                                                                                                                                                                                        | N   |
|             | CHAR(8)<br>NOT NULL                   | Not used..                                                                                                                                                                                                                       | N   |
| COLNAME     | VARCHAR(18)<br>NOT NULL               | Name of the column to which the UPDATE privilege applies.                                                                                                                                                                        | G   |
| IBMREQD     | CHAR(1)<br>NOT NULL                   | A value of Y indicates that the row came from the basic machine-readable material (MRM) tape. For all other values, see "Release dependency indicators" on page 1005.                                                            | G   |
|             | CHAR(16)<br>NOT NULL WITH<br>DEFAULT  | Not used                                                                                                                                                                                                                         | N   |
| COLLID      | CHAR(18)<br>NOT NULL WITH<br>DEFAULT  | If GRANTEE is a package, its collection name. Otherwise, the value is blank.                                                                                                                                                     | G   |
| CONTOKEN    | CHAR(8)<br>NOT NULL WITH<br>DEFAULT   | If GRANTEE is a package, the consistency token of the DBRM from which the package was derived. Otherwise, the value is blank.                                                                                                    | S   |
| PRIVILEGE   | CHAR(1)<br>NOT NULL WITH<br>DEFAULT   | Indicates which privilege this row describes:<br><b>R</b> Row pertains to the REFERENCES privilege.<br><b>blank</b> Row pertains to the UPDATE privilege.                                                                        | G   |
| GRANTEDTS   | TIMESTAMP<br>NOT NULL WITH<br>DEFAULT | Time when the GRANT statement was executed.                                                                                                                                                                                      | G   |

45. PUBLIC followed by an asterisk (PUBLIC\*) denotes PUBLIC AT ALL LOCATIONS. For the conditions where GRANTOR can be PUBLIC or PUBLIC\*, see Part 3 (Volume 1) of *DB2 Administration Guide*.

## SYSIBM.SYSCOLDIST

### SYSIBM.SYSCOLDIST table

Contains one or more rows for the first key column of an index key. Rows in this table can be inserted, updated, and deleted.

| Column name   | Data type                                | Description                                                                                                                                                                                                                                                                                             | Use |
|---------------|------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
|               | SMALLINT<br>NOT NULL                     | Not used                                                                                                                                                                                                                                                                                                | N   |
| STATSTIME     | TIMESTAMP<br>NOT NULL WITH<br>DEFAULT    | If RUNSTATS updated the statistics, the date and time when the last invocation of RUNSTATS updated the statistics.                                                                                                                                                                                      | G   |
| IBMREQD       | CHAR(1)<br>NOT NULL                      | A value of Y indicates that the row came from the basic machine-readable material (MRM) tape. For all other values, see "Release dependency indicators" on page 1005.                                                                                                                                   | G   |
| TBOWNER       | CHAR(8)<br>NOT NULL                      | Authorization ID of the owner of the table that contains the column.                                                                                                                                                                                                                                    | G   |
| TBNAME        | VARCHAR(18)<br>NOT NULL                  | Name of the table that contains the column.                                                                                                                                                                                                                                                             | G   |
| NAME          | VARCHAR(18)<br>NOT NULL                  | Name of the column. If NUMCOLUMNS is greater than 1, this name identifies the first column name of the set of columns associated with the statistics.                                                                                                                                                   | G   |
| COLVALUE      | VARCHAR(254)<br>NOT NULL<br>FOR BIT DATA | Contains the data of a frequently occurring value. Statistics are not collected for an index on a ROWID column. If the value has a non-character data type, the data might not be printable.                                                                                                            | S   |
| TYPE          | CHAR(1)<br>NOT NULL WITH<br>DEFAULT 'F'  | The type of statistics gathered:<br><b>C</b> Cardinality<br><b>F</b> Frequent value                                                                                                                                                                                                                     | G   |
| CARDF         | FLOAT<br>NOT NULL WITH<br>DEFAULT -1     | Number of distinct values for the column group. This number is valid only for TYPE C statistics.                                                                                                                                                                                                        | S   |
| COLGROUPCOLNO | VARCHAR(254)<br>NOT NULL WITH<br>DEFAULT | Identifies the set of columns associated with the statistics. If the statistics are only associated with a single column, the field contains a zero length. Otherwise, the field is an array of SMALLINT column numbers with a dimension equal to the value in NUMCOLUMNS. This is an updatable column. | S   |
| NUMCOLUMNS    | SMALLINT<br>NOT NULL WITH<br>DEFAULT 1   | Identifies the number of columns associated with the statistics.                                                                                                                                                                                                                                        | G   |
| FREQUENCYF    | FLOAT<br>NOT NULL WITH<br>DEFAULT -1     | Gives the percentage of rows in the table with the value specified in COLVALUE when the number is multiplied by 100. For example, a value of 1 indicates 100%. A value of .153 indicates 15.3%. Statistics are not collected for an index on a ROWID column.                                            | G   |

---

## SYSIBM.SYSCOLDIST\_HIST table

# Contains rows from SYSCOLDIST. Rows are added or changed in this table when  
# RUNSTATS collects history statistics. Rows in this table can also be inserted,  
| updated, and deleted.

| Column name   | Data type                                | Description                                                                                                                                                                                                                                                                                                        | Use |
|---------------|------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| STATSTIME     | TIMESTAMP<br>NOT NULL                    | If RUNSTATS updated the statistics, the date and time when the last invocation of RUNSTATS updated the statistics.                                                                                                                                                                                                 | G   |
| TBOWNER       | CHAR(8)<br>NOT NULL                      | Authorization ID of the owner of the table that contains the column.                                                                                                                                                                                                                                               | G   |
| TBNAME        | VARCHAR(18)<br>NOT NULL                  | Name of the table that contains the column.                                                                                                                                                                                                                                                                        | G   |
| NAME          | VARCHAR(18)<br>NOT NULL                  | Name of the column. If NUMCOLUMNS is greater than 1, this name identifies the first column name of the set of columns associated with the statistics.                                                                                                                                                              | G   |
| COLVALUE      | VARCHAR(255)<br>NOT NULL<br>FOR BIT DATA | Contains the data of a frequently occurring value. Statistics are not collected for an index on a ROWID column. If the value has a non-character data type, the data might not be printable.                                                                                                                       | S   |
| TYPE          | CHAR(1)<br>NOT NULL WITH<br>DEFAULT 'F'  | The type of statistics gathered:<br><b>C</b> Cardinality<br><b>F</b> Frequent value                                                                                                                                                                                                                                | G   |
| CARDF         | FLOAT(8)<br>NOT NULL WITH<br>DEFAULT -1  | Number of distinct values for the column group. This number is valid only for TYPE C statistics. The value is -1 if statistics have not been gathered.                                                                                                                                                             | S   |
| COLGROUPCOLNO | VARCHAR(254)<br>NOT NULL                 | Identifies the set of columns associated with the statistics. If the statistics are only associated with a single column, the field contains a zero length. Otherwise, the field is an array of SMALLINT column numbers with a dimension equal to the value in NUMCOLUMNS.                                         | S   |
| NUMCOLUMNS    | SMALLINT<br>NOT NULL WITH<br>DEFAULT 1   | Identifies the number of columns associated with the statistics.                                                                                                                                                                                                                                                   | G   |
| FREQUENCYF    | FLOAT(8)<br>NOT NULL<br>DEFAULT -1       | Gives the percentage of rows in the table with the value specified in COLVALUE when the number is multiplied by 100. For example, a value of 1 indicates 100%. A value of .153 indicates 15.3%. Statistics are not collected for an index on a ROWID column. The value is -1 if statistics have not been gathered. | G   |
| IBMREQD       | CHAR(1)<br>NOT NULL<br>DEFAULT 'N'       | A value of Y indicates that the row came from the basic machine-readable material (MRM) tape. For all other values, see "Release dependency indicators" on page 1005.                                                                                                                                              | G   |

## SYSIBM.SYSCOLDISTSTATS

### SYSIBM.SYSCOLDISTSTATS table

Contains zero or more rows per partition for the first key column of a partitioning index. Rows are inserted when RUNSTATS scans index partitions of the partitioning index. No row is inserted if the index is a nonpartitioning index. Rows in this table can be inserted, updated, and deleted.

| Column name   | Data type                                | Description                                                                                                                                                                                                                                                                                             | Use |
|---------------|------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
|               | SMALLINT<br>NOT NULL                     | Not used                                                                                                                                                                                                                                                                                                | N   |
| STATSTIME     | TIMESTAMP<br>NOT NULL WITH<br>DEFAULT    | If RUNSTATS updated the statistics, the date and time when the last invocation of RUNSTATS updated the statistics.                                                                                                                                                                                      | G   |
| IBMREQD       | CHAR(1)<br>NOT NULL                      | A value of Y indicates that the row came from the basic machine-readable material (MRM) tape. For all other values, see "Release dependency indicators" on page 1005.                                                                                                                                   | G   |
| PARTITION     | SMALLINT<br>NOT NULL                     | Partition number for the table space that contains the table in which the column is defined.                                                                                                                                                                                                            | G   |
| TBOWNER       | CHAR(8)<br>NOT NULL                      | Authorization ID of the owner of the table that contains the column.                                                                                                                                                                                                                                    | G   |
| TBNAME        | VARCHAR(18)<br>NOT NULL                  | Name of the table that contains the column.                                                                                                                                                                                                                                                             | G   |
| NAME          | VARCHAR(18)<br>NOT NULL                  | Name of the column. If NUMCOLUMNS is greater than 1, this name identifies the first column name of the set of columns associated with the statistics.                                                                                                                                                   | G   |
| COLVALUE      | VARCHAR(254)<br>NOT NULL<br>FOR BIT DATA | Contains the data of a frequently occurring value. Statistics are not collected for an index on a ROWID column. If the value has a non-character data type, the data may not be printable.                                                                                                              | S   |
| TYPE          | CHAR(1)<br>NOT NULL WITH<br>DEFAULT 'F'  | The type of statistics gathered:<br><b>C</b> Cardinality<br><b>F</b> Frequent value                                                                                                                                                                                                                     | G   |
| CARDF         | FLOAT<br>NOT NULL WITH<br>DEFAULT -1     | Number of distinct values for the column group. This number is valid only for TYPE C statistics.                                                                                                                                                                                                        | S   |
| COLGROUPCOLNO | VARCHAR(254)<br>NOT NULL WITH<br>DEFAULT | Identifies the set of columns associated with the statistics. If the statistics are only associated with a single column, the field contains a zero length. Otherwise, the field is an array of SMALLINT column numbers with a dimension equal to the value in NUMCOLUMNS. This is an updatable column. | S   |
| NUMCOLUMNS    | SMALLINT<br>NOT NULL WITH<br>DEFAULT 1   | Identifies the number of columns associated with the statistics.                                                                                                                                                                                                                                        | G   |
| FREQUENCYF    | FLOAT<br>NOT NULL WITH<br>DEFAULT -1     | Gives the percentage of rows in the table with the value specified in COLVALUE when the number is multiplied by 100. For example, a value of 1 indicates 100%. A value of .153 indicates 15.3%. Statistics are not collected for an index on a ROWID column.                                            | G   |

---

## SYSIBM.SYSCOLSTATS table

Contains partition statistics for selected columns. For each column, a row exists for each partition in the table. Rows are inserted when RUNSTATS collects either indexed column statistics or non-indexed column statistics for a partitioned table space. No row is inserted if the table space is nonpartitioned. Rows in this table can be inserted, updated, and deleted.

| Column name | Data type                                 | Description                                                                                                                                                                                                                                                                                                                              | Use |
|-------------|-------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| HIGHKEY     | CHAR(8)<br>NOT NULL<br>FOR BIT DATA       | Highest value of the column within the partition. Blank if statistics have not been gathered or the column is an indicator column. If the column has a non-character data type, the data might not be printable.                                                                                                                         | S   |
| HIGH2KEY    | CHAR(8)<br>NOT NULL<br>FOR BIT DATA       | Second highest value of the column within the partition. Blank if statistics have not been gathered or the column is an indicator column. If the column has a non-character data type, the data might not be printable.                                                                                                                  | S   |
| LOWKEY      | CHAR(8)<br>NOT NULL<br>FOR BIT DATA       | Lowest value of the column within the partition. Blank if statistics have not been gathered or the column is an indicator column. If the column has a non-character data type, the data might not be printable.                                                                                                                          | S   |
| LOW2KEY     | CHAR(8)<br>NOT NULL<br>FOR BIT DATA       | Second lowest value of the column within the partition. Blank if statistics have not been gathered or the column is an indicator column. If the column has a non-character data type, the data might not be printable.                                                                                                                   | S   |
|             | INTEGER<br>NOT NULL                       | Number of distinct column values in the partition.                                                                                                                                                                                                                                                                                       | S   |
| STATSTIME   | TIMESTAMP<br>NOT NULL                     | If RUNSTATS updated the statistics, the date and time when the last invocation of RUNSTATS updated the statistics. If the value is '0001-01-02-00.00.000000', which indicates that an ALTER TABLE statement was executed to change the length of a VARCHAR column, RUNSTATS should be run to update the statistics before they are used. | G   |
| IBMREQD     | CHAR(1)<br>NOT NULL                       | Whether the row came from the basic machine-readable material (MRM) tape:<br><br>N      No<br>Y      Yes                                                                                                                                                                                                                                 | G   |
| PARTITION   | SMALLINT<br>NOT NULL                      | Partition number for the table space that contains the table in which the column is defined.                                                                                                                                                                                                                                             | G   |
| TBOWNER     | CHAR(8)<br>NOT NULL                       | Authorization ID of the owner of the table that contains the column.                                                                                                                                                                                                                                                                     | G   |
| TBNAME      | VARCHAR(18)<br>NOT NULL                   | Name of the table that contains the column.                                                                                                                                                                                                                                                                                              | G   |
| NAME        | VARCHAR(18)<br>NOT NULL                   | Name of the column.                                                                                                                                                                                                                                                                                                                      | G   |
| COLCARDATA  | VARCHAR(1000)<br>NOT NULL<br>FOR BIT DATA | Internal use only                                                                                                                                                                                                                                                                                                                        | I   |

## SYSIBM.SYSCOLUMNS

### SYSIBM.SYSCOLUMNS table

Contains one row for every column of each table and view.

| Column name | Data type               | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   | Use |
|-------------|-------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| NAME        | VARCHAR(18)<br>NOT NULL | Name of the column.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           | G   |
| TBNAME      | VARCHAR(18)<br>NOT NULL | Name of the table or view which contains the column.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          | G   |
| TBCREATOR   | CHAR(8)<br>NOT NULL     | Authorization ID of the owner of the table or view that contains the column.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  | G   |
| COLNO       | SMALLINT<br>NOT NULL    | Numeric place of the column in the table or view; for example 4 (out of 10).                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  | G   |
| COLTYPE     | CHAR(8)<br>NOT NULL     | The type of the column specified in the definition of the column:<br><b>INTEGER</b> Large integer<br><b>SMALLINT</b> Small integer<br><b>FLOAT</b> Floating-point<br><b>CHAR</b> Fixed-length character string<br><b>VARCHAR</b> Varying-length character string<br><b>LONGVAR</b> Varying-length character string<br><b>DECIMAL</b> Decimal<br><b>GRAPHIC</b> Fixed-length graphic string<br><b>VARG</b> Varying-length graphic string<br><b>LONGVARG</b> Varying-length graphic string<br><b>DATE</b> Date<br><b>TIME</b> Time<br><b>TIMESTAMP</b> Timestamp<br><b>BLOB</b> Binary large object<br><b>CLOB</b> Character large object<br><b>DBCLOB</b> Double-byte character large object<br><b>ROWID</b> Row ID data type<br><b>DISTINCT</b> Distinct type | G   |

Whether a column described as VARCHAR, LONGVAR, VARG, or LONGVARG is a long string column depends on its length attribute. A column described as BLOB, CLOB, or DBCLOB is always a long string column.

## SYSIBM.SYSCOLUMNS

| Column name | Data type                           | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    | Use |
|-------------|-------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| LENGTH      | SMALLINT<br>NOT NULL                | Length attribute of the column or, in the case of a decimal column, its precision. The number does not include the internal prefixes that are used to record the actual length and null state, where applicable.<br><br><b>INTEGER</b> 4<br><b>SMALLINT</b> 2<br><b>FLOAT</b> 4 or 8<br><b>CHAR</b> Length of string<br><b>VARCHAR</b> Maximum length of string<br><b>LONGVAR</b> Maximum length of string<br><b>DECIMAL</b> Precision of number<br><b>GRAPHIC</b> Number of DBCS characters<br><b>VARG</b> Maximum number of DBCS characters<br><b>LONGVARG</b> Maximum number of DBCS characters<br><b>DATE</b> 4<br><b>TIME</b> 3<br><b>TIMESTAMP</b> 10<br><b>BLOB</b> 4 - The length of the field that is stored in the base table. The maximum length of the LOB column is found in LENGTH2.<br><br><b>CLOB</b> 4 - The length of the field that is stored in the base table. The maximum length of the CLOB column is found in LENGTH2.<br><br><b>DBCLOB</b> 4 - The length of the field that is stored in the base table. The maximum length of the DBCLOB column is found in LENGTH2.<br><br><b>ROWID</b> 17 - The maximum length of the stored portion of the identifier.<br><br><b>DISTINCT</b> The length of the source data type. | G   |
| SCALE       | SMALLINT<br>NOT NULL                | Scale of decimal data. Zero if not a decimal column.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           | G   |
| NULLS       | CHAR(1)<br>NOT NULL                 | Whether the column can contain null values:<br><br><b>N</b> No<br><b>Y</b> Yes<br><br>The value can be N for a view column that is derived from an expression or a function. Nevertheless, such a column allows nulls when an outer select list refers to it.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  | G   |
|             | INTEGER<br>NOT NULL                 | Not used                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       | N   |
| HIGH2KEY    | CHAR(8)<br>NOT NULL<br>FOR BIT DATA | Second highest value of the column. Blank if statistics have not been gathered, or the column is an indicator column or a column of an auxiliary table. If the column has a non-character data type, the data might not be printable. This is an updatable column.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             | S   |
| LOW2KEY     | CHAR(8)<br>NOT NULL<br>FOR BIT DATA | Second lowest value of the column. Blank if statistics have not been gathered, or the column is an indicator column or a column of an auxiliary table. If the column has a non-character data type, the data might not be printable. This is an updatable column.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              | S   |
| UPDATES     | CHAR(1)<br>NOT NULL                 | Whether the column can be updated:<br><br><b>N</b> No<br><b>Y</b> Yes<br><br>The value is N if the column is:<br><ul style="list-style-type: none"> <li>• Derived from a function or expression</li> <li>• A column with a row ID data type (or a distinct type based on a row ID type)</li> </ul><br>The value is Y if the column is a read-only view or the columns are defined with the AS IDENTITY and GENERATED ALWAYS attributes..                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       | G   |
| IBMREQD     | CHAR(1)<br>NOT NULL                 | A value of Y indicates that the row came from the basic machine-readable material (MRM) tape. For all other values, see "Release dependency indicators" on page 1005.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          | G   |

## SYSIBM.SYSCOLUMNS

| Column name           | Data type                | Description                                                                                                                                                                                                                                                                                                                                                                                                                                       | Use       |               |         |   |                     |        |                       |                      |      |                  |      |                  |           |                       |  |
|-----------------------|--------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------|---------------|---------|---|---------------------|--------|-----------------------|----------------------|------|------------------|------|------------------|-----------|-----------------------|--|
| REMARKS               | VARCHAR(254)<br>NOT NULL | A character string provided by the user with the COMMENT ON statement.                                                                                                                                                                                                                                                                                                                                                                            | G         |               |         |   |                     |        |                       |                      |      |                  |      |                  |           |                       |  |
| DEFAULT               | CHAR(1)<br>NOT NULL      | The contents of this column are meaningful only if the TYPE column for the associated SYSTABLES row indicates that this is for a table (T) or a created temporary table (G).                                                                                                                                                                                                                                                                      | G         |               |         |   |                     |        |                       |                      |      |                  |      |                  |           |                       |  |
|                       |                          | Default indicator:                                                                                                                                                                                                                                                                                                                                                                                                                                |           |               |         |   |                     |        |                       |                      |      |                  |      |                  |           |                       |  |
|                       | <b>A</b>                 | The column has a row ID data type (COLTYPE='ROWID') and the GENERATED ALWAYS attribute.                                                                                                                                                                                                                                                                                                                                                           |           |               |         |   |                     |        |                       |                      |      |                  |      |                  |           |                       |  |
|                       | <b>B</b>                 | The column has a default value that depends on the data type of the column.                                                                                                                                                                                                                                                                                                                                                                       |           |               |         |   |                     |        |                       |                      |      |                  |      |                  |           |                       |  |
|                       |                          | <table> <thead> <tr> <th>Data type</th> <th>Default Value</th> </tr> </thead> <tbody> <tr> <td>Numeric</td> <td>0</td> </tr> <tr> <td>Fixed-length string</td> <td>Blanks</td> </tr> <tr> <td>Varying-length string</td> <td>A string length of 0</td> </tr> <tr> <td>Date</td> <td>The current date</td> </tr> <tr> <td>Time</td> <td>The current time</td> </tr> <tr> <td>Timestamp</td> <td>The current timestamp</td> </tr> </tbody> </table> | Data type | Default Value | Numeric | 0 | Fixed-length string | Blanks | Varying-length string | A string length of 0 | Date | The current date | Time | The current time | Timestamp | The current timestamp |  |
| Data type             | Default Value            |                                                                                                                                                                                                                                                                                                                                                                                                                                                   |           |               |         |   |                     |        |                       |                      |      |                  |      |                  |           |                       |  |
| Numeric               | 0                        |                                                                                                                                                                                                                                                                                                                                                                                                                                                   |           |               |         |   |                     |        |                       |                      |      |                  |      |                  |           |                       |  |
| Fixed-length string   | Blanks                   |                                                                                                                                                                                                                                                                                                                                                                                                                                                   |           |               |         |   |                     |        |                       |                      |      |                  |      |                  |           |                       |  |
| Varying-length string | A string length of 0     |                                                                                                                                                                                                                                                                                                                                                                                                                                                   |           |               |         |   |                     |        |                       |                      |      |                  |      |                  |           |                       |  |
| Date                  | The current date         |                                                                                                                                                                                                                                                                                                                                                                                                                                                   |           |               |         |   |                     |        |                       |                      |      |                  |      |                  |           |                       |  |
| Time                  | The current time         |                                                                                                                                                                                                                                                                                                                                                                                                                                                   |           |               |         |   |                     |        |                       |                      |      |                  |      |                  |           |                       |  |
| Timestamp             | The current timestamp    |                                                                                                                                                                                                                                                                                                                                                                                                                                                   |           |               |         |   |                     |        |                       |                      |      |                  |      |                  |           |                       |  |
|                       | <b>D</b>                 | The column has a row ID data type (COLTYPE='ROWID') and the GENERATED BY DEFAULT attribute.                                                                                                                                                                                                                                                                                                                                                       |           |               |         |   |                     |        |                       |                      |      |                  |      |                  |           |                       |  |
|                       | <b>I</b>                 | The column is defined with the AS IDENTITY and GENERATED ALWAYS attributes.                                                                                                                                                                                                                                                                                                                                                                       |           |               |         |   |                     |        |                       |                      |      |                  |      |                  |           |                       |  |
|                       | <b>J</b>                 | The column is defined with the AS IDENTITY and GENERATED BY DEFAULT attributes.                                                                                                                                                                                                                                                                                                                                                                   |           |               |         |   |                     |        |                       |                      |      |                  |      |                  |           |                       |  |
|                       | <b>N</b>                 | The column has no default value.                                                                                                                                                                                                                                                                                                                                                                                                                  |           |               |         |   |                     |        |                       |                      |      |                  |      |                  |           |                       |  |
|                       | <b>S</b>                 | The column has a default value that is the value of the SQL authorization ID of the process at the time a default value is used.                                                                                                                                                                                                                                                                                                                  |           |               |         |   |                     |        |                       |                      |      |                  |      |                  |           |                       |  |
|                       | <b>U</b>                 | The column has a default value that is the value of the USER special register at the time a default value is used.                                                                                                                                                                                                                                                                                                                                |           |               |         |   |                     |        |                       |                      |      |                  |      |                  |           |                       |  |
|                       | <b>Y</b>                 | If the NULLS column is Y, the column has a default value of null.<br><br>If the NULLS column is N, the default value depends on the data type of the column.                                                                                                                                                                                                                                                                                      |           |               |         |   |                     |        |                       |                      |      |                  |      |                  |           |                       |  |
|                       |                          | <table> <thead> <tr> <th>Data type</th> <th>Default Value</th> </tr> </thead> <tbody> <tr> <td>Numeric</td> <td>0</td> </tr> <tr> <td>Fixed-length string</td> <td>Blanks</td> </tr> <tr> <td>Varying-length string</td> <td>A string length of 0</td> </tr> <tr> <td>Date</td> <td>The current date</td> </tr> <tr> <td>Time</td> <td>The current time</td> </tr> <tr> <td>Timestamp</td> <td>The current timestamp</td> </tr> </tbody> </table> | Data type | Default Value | Numeric | 0 | Fixed-length string | Blanks | Varying-length string | A string length of 0 | Date | The current date | Time | The current time | Timestamp | The current timestamp |  |
| Data type             | Default Value            |                                                                                                                                                                                                                                                                                                                                                                                                                                                   |           |               |         |   |                     |        |                       |                      |      |                  |      |                  |           |                       |  |
| Numeric               | 0                        |                                                                                                                                                                                                                                                                                                                                                                                                                                                   |           |               |         |   |                     |        |                       |                      |      |                  |      |                  |           |                       |  |
| Fixed-length string   | Blanks                   |                                                                                                                                                                                                                                                                                                                                                                                                                                                   |           |               |         |   |                     |        |                       |                      |      |                  |      |                  |           |                       |  |
| Varying-length string | A string length of 0     |                                                                                                                                                                                                                                                                                                                                                                                                                                                   |           |               |         |   |                     |        |                       |                      |      |                  |      |                  |           |                       |  |
| Date                  | The current date         |                                                                                                                                                                                                                                                                                                                                                                                                                                                   |           |               |         |   |                     |        |                       |                      |      |                  |      |                  |           |                       |  |
| Time                  | The current time         |                                                                                                                                                                                                                                                                                                                                                                                                                                                   |           |               |         |   |                     |        |                       |                      |      |                  |      |                  |           |                       |  |
| Timestamp             | The current timestamp    |                                                                                                                                                                                                                                                                                                                                                                                                                                                   |           |               |         |   |                     |        |                       |                      |      |                  |      |                  |           |                       |  |

## SYSIBM.SYSCOLUMNS

| Column name         | Data type                             | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             | Use |
|---------------------|---------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| DEFAULT (continued) | CHAR(1)<br>NOT NULL                   | The contents of this column are meaningful only if the TYPE column for the associated SYSTABLES row indicates that this is for a table (T) or a created temporary table (G).<br><br>Default indicator:<br><br>1 The column has a default value that is the string constant found in the DEFAULTVALUE column of this table row.<br>2 The column has a default value that is the floating-point constant found in the DEFAULTVALUE column of this table row.<br>3 The column has a default value that is the decimal constant found in the DEFAULTVALUE column of this table row.<br>4 The column has a default value that is the integer constant found in the DEFAULTVALUE column of this table row.<br>5 The column has a default value that is the hex character string found in the DEFAULTVALUE column of this table row.<br>7 The column has a graphic data type and has a default value that is the character string constant found in the DEFAULTVALUE column of this table row. | G   |
| #                   |                                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |     |
| #                   |                                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |     |
| #                   |                                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |     |
| KEYSEQ              | SMALLINT<br>NOT NULL                  | The column's numeric position within the table's primary key. The value is 0 if it is not part of a primary key.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        | G   |
| FOREIGNKEY          | CHAR(1)<br>NOT NULL                   | Applies to character columns only, where it indicates the subtype of the data.<br>• B indicates BIT data.<br>• S indicates SBCS data if the encoding scheme is Unicode or if the value of the field MIXED DATA on installation panel DSNTIPF is YES.<br>• Any other value indicates:<br>– MIXED if the encoding scheme is Unicode or the value of the field MIXED DATA on installation panel DSNTIPF is YES.<br>– SBCS if the encoding scheme is not Unicode and the value of the field MIXED DATA on installation panel DSNTIPF is NO.<br><br>For views defined prior to Version 7, subtype information is not available and the default (MIXED or SBCS) is used. This is an updatable column.                                                                                                                                                                                                                                                                                         | G   |
| FLDPROC             | CHAR(1)<br>NOT NULL                   | Whether the column has a field procedure:<br>N No<br>Y Yes<br>blank The column is for a view defined prior to Version 7. Views defined after Version 7 contain Y or N.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  | G   |
| LABEL               | VARCHAR(30)<br>NOT NULL               | The column label provided by the user with a LABEL ON statement; otherwise it is an empty string.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       | G   |
| STATSTIME           | TIMESTAMP<br>NOT NULL WITH<br>DEFAULT | If RUNSTATS updated the statistics, the date and time when the last invocation of RUNSTATS updated the statistics. The default value is '0001-01-01-00.00.000000'. If the value is '0001-01-02-00.00.000000', which indicates that an ALTER TABLE statement was executed to change the length of a VARCHAR column, RUNSTATS should be run to update the statistics before they are used. This is an updatable column.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   | G   |

## SYSIBM.SYSCOLUMNS

| Column name  | Data type                                       | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     | Use |
|--------------|-------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| DEFAULTVALUE | VARCHAR(512)<br>NOT NULL WITH<br>DEFAULT        | This field is meaningful only if the column being described is for a table (the TYPE column of the associated SYSTABLES row is T for table or G for created temporary table).<br><br>When the DEFAULT column is 1, 2, 3, 4, 5, 6, or 7, this field contains the default value of the column.<br><br>If the default value is a string constant or a hexadecimal constant (DEFAULT is 1, 5, 6, or 7, respectively), the value is stored without delimiters, except for a graphic string constant which is enclosed by the shift-out and shift-in characters.<br><br>If the default value is a numeric constant (DEFAULT is 2, 3, or 4), the value is stored as specified by the user, including sign and decimal point representation, as appropriate for the constant.<br>When the default column is S or U and the default value was specified with the definition of a new column on an ALTER TABLE statement, this field contains the value of the CURRENT SQLID or USER special register at the time of the ALTER statement. | G   |
| COLCARDF     | FLOAT<br>NOT NULL WITH<br>DEFAULT               | Estimated number of distinct values in the column. For an indicator column, this is the number of LOBs that are not null and have a length greater than zero. The value is -1 if statistics have not been gathered. The value is -2 for the first column of an index of an auxiliary table. This is an updatable column.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        | S   |
| COLSTATUS    | CHAR(1)<br>NOT NULL WITH<br>DEFAULT             | Indicates the status of the definition of a column:<br><br><b>I</b> The definition is incomplete because a LOB table space, auxiliary table, or index on an auxiliary table has not been created for the column.<br><br><b>blank</b> The definition is complete.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                | G   |
| LENGTH2      | INTEGER<br>NOT NULL WITH<br>DEFAULT             | Maximum length of the data retrieved from the column. Possible values are:<br><br><b>0</b> Not a LOB or ROWID column<br><b>40</b> For a ROWID column, the length of the returned value<br><b>1 to 2 147 483 647 bytes</b><br>For a LOB column, the maximum length                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               | G   |
| DATATYPEID   | INTEGER<br>NOT NULL WITH<br>DEFAULT             | For a built-in data type, the internal ID of the built-in type. For a distinct type, the internal ID of the distinct type.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      | S   |
| SOURCETYPEID | INTEGER<br>NOT NULL WITH<br>DEFAULT             | For a built-in data type, 0. For a distinct type, the internal ID of the built-in data type upon which the distinct type is sourced.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            | S   |
| TYPESCHEMA   | CHAR(8)<br>NOT NULL WITH<br>DEFAULT<br>'SYSIBM' | If COLTYPE is 'DISTINCT', the schema of the distinct type. Otherwise, the value is 'SYSIBM'.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    | G   |
| TYPENAME     | VARCHAR(18)<br>NOT NULL WITH<br>DEFAULT         | If COLTYPE is 'DISTINCT', the name of the distinct type. Otherwise, the value is the same as the value of the COLTYPE column. TYPENAME is set only for columns created in Version 6 or later. The value for columns created earlier is not filled in.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           | G   |
| CREATEDTS    | TIMESTAMP<br>NOT NULL WITH<br>DEFAULT           | Timestamp when the column was created. The value is '0001-01-0100.00.00.000000' if the column was created prior to migration to Version 6.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      | G   |

---

## SYSIBM.SYSCOLUMNS\_HIST table

```
Contains rows from SYSCOLUMNS. Rows are added or changed in this table when
RUNSTATS collects history statistics. Rows in this table can also be inserted,
| updated, and deleted.
```

| Column name      | Data type                                                                                                               | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 | Use            |               |                 |               |              |                |             |                               |                |                                 |                |                                 |                |                     |                |                             |             |                                   |                 |                                   |             |      |             |      |                  |           |             |                                                                                                                         |             |                                                                                              |               |                                    |              |                  |                 |               |   |
|------------------|-------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------|---------------|-----------------|---------------|--------------|----------------|-------------|-------------------------------|----------------|---------------------------------|----------------|---------------------------------|----------------|---------------------|----------------|-----------------------------|-------------|-----------------------------------|-----------------|-----------------------------------|-------------|------|-------------|------|------------------|-----------|-------------|-------------------------------------------------------------------------------------------------------------------------|-------------|----------------------------------------------------------------------------------------------|---------------|------------------------------------|--------------|------------------|-----------------|---------------|---|
| NAME             | VARCHAR(18)<br>NOT NULL                                                                                                 | Name of the column.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         | G              |               |                 |               |              |                |             |                               |                |                                 |                |                                 |                |                     |                |                             |             |                                   |                 |                                   |             |      |             |      |                  |           |             |                                                                                                                         |             |                                                                                              |               |                                    |              |                  |                 |               |   |
| TBNAME           | VARCHAR(18)<br>NOT NULL                                                                                                 | Name of the table or view that contains the column.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         | G              |               |                 |               |              |                |             |                               |                |                                 |                |                                 |                |                     |                |                             |             |                                   |                 |                                   |             |      |             |      |                  |           |             |                                                                                                                         |             |                                                                                              |               |                                    |              |                  |                 |               |   |
| TBCREATOR        | CHAR(8)<br>NOT NULL                                                                                                     | Authorization ID of the owner of the table or view that contains the column.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                | G              |               |                 |               |              |                |             |                               |                |                                 |                |                                 |                |                     |                |                             |             |                                   |                 |                                   |             |      |             |      |                  |           |             |                                                                                                                         |             |                                                                                              |               |                                    |              |                  |                 |               |   |
| COLNO            | SMALLINT<br>NOT NULL                                                                                                    | Numeric place of the column in the table or view. For example 4 (out of 10).                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                | G              |               |                 |               |              |                |             |                               |                |                                 |                |                                 |                |                     |                |                             |             |                                   |                 |                                   |             |      |             |      |                  |           |             |                                                                                                                         |             |                                                                                              |               |                                    |              |                  |                 |               |   |
| COLTYPE          | CHAR(8)<br>NOT NULL                                                                                                     | <p>The type of the column specified in the definition of the column:</p> <table> <tr><td><b>INTEGER</b></td><td>Large integer</td></tr> <tr><td><b>SMALLINT</b></td><td>Small integer</td></tr> <tr><td><b>FLOAT</b></td><td>Floating-point</td></tr> <tr><td><b>CHAR</b></td><td>Fixed-length character string</td></tr> <tr><td><b>VARCHAR</b></td><td>Varying-length character string</td></tr> <tr><td><b>LONGVAR</b></td><td>Varying-length character string</td></tr> <tr><td><b>DECIMAL</b></td><td>Decimal</td></tr> <tr><td><b>GRAPHIC</b></td><td>Fixed-length graphic string</td></tr> <tr><td><b>VARG</b></td><td>Varying-length graphic string</td></tr> <tr><td><b>LONGVARG</b></td><td>Varying-length graphic string</td></tr> <tr><td><b>DATE</b></td><td>Date</td></tr> <tr><td><b>TIME</b></td><td>Time</td></tr> <tr><td><b>TIMESTAMP</b></td><td>Timestamp</td></tr> <tr><td><b>BLOB</b></td><td>Binary large object</td></tr> <tr><td><b>CLOB</b></td><td>Character large object</td></tr> <tr><td><b>DBCLOB</b></td><td>Double-byte character large object</td></tr> <tr><td><b>ROWID</b></td><td>Row ID data type</td></tr> <tr><td><b>DISTINCT</b></td><td>Distinct type</td></tr> </table>                                                                         | <b>INTEGER</b> | Large integer | <b>SMALLINT</b> | Small integer | <b>FLOAT</b> | Floating-point | <b>CHAR</b> | Fixed-length character string | <b>VARCHAR</b> | Varying-length character string | <b>LONGVAR</b> | Varying-length character string | <b>DECIMAL</b> | Decimal             | <b>GRAPHIC</b> | Fixed-length graphic string | <b>VARG</b> | Varying-length graphic string     | <b>LONGVARG</b> | Varying-length graphic string     | <b>DATE</b> | Date | <b>TIME</b> | Time | <b>TIMESTAMP</b> | Timestamp | <b>BLOB</b> | Binary large object                                                                                                     | <b>CLOB</b> | Character large object                                                                       | <b>DBCLOB</b> | Double-byte character large object | <b>ROWID</b> | Row ID data type | <b>DISTINCT</b> | Distinct type | G |
| <b>INTEGER</b>   | Large integer                                                                                                           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |                |               |                 |               |              |                |             |                               |                |                                 |                |                                 |                |                     |                |                             |             |                                   |                 |                                   |             |      |             |      |                  |           |             |                                                                                                                         |             |                                                                                              |               |                                    |              |                  |                 |               |   |
| <b>SMALLINT</b>  | Small integer                                                                                                           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |                |               |                 |               |              |                |             |                               |                |                                 |                |                                 |                |                     |                |                             |             |                                   |                 |                                   |             |      |             |      |                  |           |             |                                                                                                                         |             |                                                                                              |               |                                    |              |                  |                 |               |   |
| <b>FLOAT</b>     | Floating-point                                                                                                          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |                |               |                 |               |              |                |             |                               |                |                                 |                |                                 |                |                     |                |                             |             |                                   |                 |                                   |             |      |             |      |                  |           |             |                                                                                                                         |             |                                                                                              |               |                                    |              |                  |                 |               |   |
| <b>CHAR</b>      | Fixed-length character string                                                                                           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |                |               |                 |               |              |                |             |                               |                |                                 |                |                                 |                |                     |                |                             |             |                                   |                 |                                   |             |      |             |      |                  |           |             |                                                                                                                         |             |                                                                                              |               |                                    |              |                  |                 |               |   |
| <b>VARCHAR</b>   | Varying-length character string                                                                                         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |                |               |                 |               |              |                |             |                               |                |                                 |                |                                 |                |                     |                |                             |             |                                   |                 |                                   |             |      |             |      |                  |           |             |                                                                                                                         |             |                                                                                              |               |                                    |              |                  |                 |               |   |
| <b>LONGVAR</b>   | Varying-length character string                                                                                         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |                |               |                 |               |              |                |             |                               |                |                                 |                |                                 |                |                     |                |                             |             |                                   |                 |                                   |             |      |             |      |                  |           |             |                                                                                                                         |             |                                                                                              |               |                                    |              |                  |                 |               |   |
| <b>DECIMAL</b>   | Decimal                                                                                                                 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |                |               |                 |               |              |                |             |                               |                |                                 |                |                                 |                |                     |                |                             |             |                                   |                 |                                   |             |      |             |      |                  |           |             |                                                                                                                         |             |                                                                                              |               |                                    |              |                  |                 |               |   |
| <b>GRAPHIC</b>   | Fixed-length graphic string                                                                                             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |                |               |                 |               |              |                |             |                               |                |                                 |                |                                 |                |                     |                |                             |             |                                   |                 |                                   |             |      |             |      |                  |           |             |                                                                                                                         |             |                                                                                              |               |                                    |              |                  |                 |               |   |
| <b>VARG</b>      | Varying-length graphic string                                                                                           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |                |               |                 |               |              |                |             |                               |                |                                 |                |                                 |                |                     |                |                             |             |                                   |                 |                                   |             |      |             |      |                  |           |             |                                                                                                                         |             |                                                                                              |               |                                    |              |                  |                 |               |   |
| <b>LONGVARG</b>  | Varying-length graphic string                                                                                           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |                |               |                 |               |              |                |             |                               |                |                                 |                |                                 |                |                     |                |                             |             |                                   |                 |                                   |             |      |             |      |                  |           |             |                                                                                                                         |             |                                                                                              |               |                                    |              |                  |                 |               |   |
| <b>DATE</b>      | Date                                                                                                                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |                |               |                 |               |              |                |             |                               |                |                                 |                |                                 |                |                     |                |                             |             |                                   |                 |                                   |             |      |             |      |                  |           |             |                                                                                                                         |             |                                                                                              |               |                                    |              |                  |                 |               |   |
| <b>TIME</b>      | Time                                                                                                                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |                |               |                 |               |              |                |             |                               |                |                                 |                |                                 |                |                     |                |                             |             |                                   |                 |                                   |             |      |             |      |                  |           |             |                                                                                                                         |             |                                                                                              |               |                                    |              |                  |                 |               |   |
| <b>TIMESTAMP</b> | Timestamp                                                                                                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |                |               |                 |               |              |                |             |                               |                |                                 |                |                                 |                |                     |                |                             |             |                                   |                 |                                   |             |      |             |      |                  |           |             |                                                                                                                         |             |                                                                                              |               |                                    |              |                  |                 |               |   |
| <b>BLOB</b>      | Binary large object                                                                                                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |                |               |                 |               |              |                |             |                               |                |                                 |                |                                 |                |                     |                |                             |             |                                   |                 |                                   |             |      |             |      |                  |           |             |                                                                                                                         |             |                                                                                              |               |                                    |              |                  |                 |               |   |
| <b>CLOB</b>      | Character large object                                                                                                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |                |               |                 |               |              |                |             |                               |                |                                 |                |                                 |                |                     |                |                             |             |                                   |                 |                                   |             |      |             |      |                  |           |             |                                                                                                                         |             |                                                                                              |               |                                    |              |                  |                 |               |   |
| <b>DBCLOB</b>    | Double-byte character large object                                                                                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |                |               |                 |               |              |                |             |                               |                |                                 |                |                                 |                |                     |                |                             |             |                                   |                 |                                   |             |      |             |      |                  |           |             |                                                                                                                         |             |                                                                                              |               |                                    |              |                  |                 |               |   |
| <b>ROWID</b>     | Row ID data type                                                                                                        |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |                |               |                 |               |              |                |             |                               |                |                                 |                |                                 |                |                     |                |                             |             |                                   |                 |                                   |             |      |             |      |                  |           |             |                                                                                                                         |             |                                                                                              |               |                                    |              |                  |                 |               |   |
| <b>DISTINCT</b>  | Distinct type                                                                                                           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |                |               |                 |               |              |                |             |                               |                |                                 |                |                                 |                |                     |                |                             |             |                                   |                 |                                   |             |      |             |      |                  |           |             |                                                                                                                         |             |                                                                                              |               |                                    |              |                  |                 |               |   |
|                  |                                                                                                                         | Whether a column described as VARCHAR, LONGVAR, CLOB, VARG, LONGVARG, DBCLOB, or BLOB is a long string column depends on its length attribute.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |                |               |                 |               |              |                |             |                               |                |                                 |                |                                 |                |                     |                |                             |             |                                   |                 |                                   |             |      |             |      |                  |           |             |                                                                                                                         |             |                                                                                              |               |                                    |              |                  |                 |               |   |
| LENGTH           | SMALLINT<br>NOT NULL                                                                                                    | <p>Length attribute of the column or, in the case of a decimal column, its precision. The number does not include the internal prefixes that are used to record the actual length and null state, where applicable.</p> <table> <tr><td><b>INTEGER</b></td><td>4</td></tr> <tr><td><b>SMALLINT</b></td><td>2</td></tr> <tr><td><b>FLOAT</b></td><td>4 or 8</td></tr> <tr><td><b>CHAR</b></td><td>Length of string</td></tr> <tr><td><b>VARCHAR</b></td><td>Maximum length of string</td></tr> <tr><td><b>LONGVAR</b></td><td>Maximum length of string</td></tr> <tr><td><b>DECIMAL</b></td><td>Precision of number</td></tr> <tr><td><b>GRAPHIC</b></td><td>Number of DBCS characters</td></tr> <tr><td><b>VARG</b></td><td>Maximum number of DBCS characters</td></tr> <tr><td><b>LONGVARG</b></td><td>Maximum number of DBCS characters</td></tr> <tr><td><b>DATE</b></td><td>4</td></tr> <tr><td><b>TIME</b></td><td>3</td></tr> <tr><td><b>TIMESTAMP</b></td><td>10</td></tr> <tr><td><b>BLOB</b></td><td>4 - The length of the field that is stored in the base table. The maximum length of the LOB column is found in LENGTH2.</td></tr> <tr><td><b>CLOB</b></td><td>4 - The length of the field that is stored in the base table. The maximum length of the CLOB</td></tr> </table> | <b>INTEGER</b> | 4             | <b>SMALLINT</b> | 2             | <b>FLOAT</b> | 4 or 8         | <b>CHAR</b> | Length of string              | <b>VARCHAR</b> | Maximum length of string        | <b>LONGVAR</b> | Maximum length of string        | <b>DECIMAL</b> | Precision of number | <b>GRAPHIC</b> | Number of DBCS characters   | <b>VARG</b> | Maximum number of DBCS characters | <b>LONGVARG</b> | Maximum number of DBCS characters | <b>DATE</b> | 4    | <b>TIME</b> | 3    | <b>TIMESTAMP</b> | 10        | <b>BLOB</b> | 4 - The length of the field that is stored in the base table. The maximum length of the LOB column is found in LENGTH2. | <b>CLOB</b> | 4 - The length of the field that is stored in the base table. The maximum length of the CLOB | G             |                                    |              |                  |                 |               |   |
| <b>INTEGER</b>   | 4                                                                                                                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |                |               |                 |               |              |                |             |                               |                |                                 |                |                                 |                |                     |                |                             |             |                                   |                 |                                   |             |      |             |      |                  |           |             |                                                                                                                         |             |                                                                                              |               |                                    |              |                  |                 |               |   |
| <b>SMALLINT</b>  | 2                                                                                                                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |                |               |                 |               |              |                |             |                               |                |                                 |                |                                 |                |                     |                |                             |             |                                   |                 |                                   |             |      |             |      |                  |           |             |                                                                                                                         |             |                                                                                              |               |                                    |              |                  |                 |               |   |
| <b>FLOAT</b>     | 4 or 8                                                                                                                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |                |               |                 |               |              |                |             |                               |                |                                 |                |                                 |                |                     |                |                             |             |                                   |                 |                                   |             |      |             |      |                  |           |             |                                                                                                                         |             |                                                                                              |               |                                    |              |                  |                 |               |   |
| <b>CHAR</b>      | Length of string                                                                                                        |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |                |               |                 |               |              |                |             |                               |                |                                 |                |                                 |                |                     |                |                             |             |                                   |                 |                                   |             |      |             |      |                  |           |             |                                                                                                                         |             |                                                                                              |               |                                    |              |                  |                 |               |   |
| <b>VARCHAR</b>   | Maximum length of string                                                                                                |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |                |               |                 |               |              |                |             |                               |                |                                 |                |                                 |                |                     |                |                             |             |                                   |                 |                                   |             |      |             |      |                  |           |             |                                                                                                                         |             |                                                                                              |               |                                    |              |                  |                 |               |   |
| <b>LONGVAR</b>   | Maximum length of string                                                                                                |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |                |               |                 |               |              |                |             |                               |                |                                 |                |                                 |                |                     |                |                             |             |                                   |                 |                                   |             |      |             |      |                  |           |             |                                                                                                                         |             |                                                                                              |               |                                    |              |                  |                 |               |   |
| <b>DECIMAL</b>   | Precision of number                                                                                                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |                |               |                 |               |              |                |             |                               |                |                                 |                |                                 |                |                     |                |                             |             |                                   |                 |                                   |             |      |             |      |                  |           |             |                                                                                                                         |             |                                                                                              |               |                                    |              |                  |                 |               |   |
| <b>GRAPHIC</b>   | Number of DBCS characters                                                                                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |                |               |                 |               |              |                |             |                               |                |                                 |                |                                 |                |                     |                |                             |             |                                   |                 |                                   |             |      |             |      |                  |           |             |                                                                                                                         |             |                                                                                              |               |                                    |              |                  |                 |               |   |
| <b>VARG</b>      | Maximum number of DBCS characters                                                                                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |                |               |                 |               |              |                |             |                               |                |                                 |                |                                 |                |                     |                |                             |             |                                   |                 |                                   |             |      |             |      |                  |           |             |                                                                                                                         |             |                                                                                              |               |                                    |              |                  |                 |               |   |
| <b>LONGVARG</b>  | Maximum number of DBCS characters                                                                                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |                |               |                 |               |              |                |             |                               |                |                                 |                |                                 |                |                     |                |                             |             |                                   |                 |                                   |             |      |             |      |                  |           |             |                                                                                                                         |             |                                                                                              |               |                                    |              |                  |                 |               |   |
| <b>DATE</b>      | 4                                                                                                                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |                |               |                 |               |              |                |             |                               |                |                                 |                |                                 |                |                     |                |                             |             |                                   |                 |                                   |             |      |             |      |                  |           |             |                                                                                                                         |             |                                                                                              |               |                                    |              |                  |                 |               |   |
| <b>TIME</b>      | 3                                                                                                                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |                |               |                 |               |              |                |             |                               |                |                                 |                |                                 |                |                     |                |                             |             |                                   |                 |                                   |             |      |             |      |                  |           |             |                                                                                                                         |             |                                                                                              |               |                                    |              |                  |                 |               |   |
| <b>TIMESTAMP</b> | 10                                                                                                                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |                |               |                 |               |              |                |             |                               |                |                                 |                |                                 |                |                     |                |                             |             |                                   |                 |                                   |             |      |             |      |                  |           |             |                                                                                                                         |             |                                                                                              |               |                                    |              |                  |                 |               |   |
| <b>BLOB</b>      | 4 - The length of the field that is stored in the base table. The maximum length of the LOB column is found in LENGTH2. |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |                |               |                 |               |              |                |             |                               |                |                                 |                |                                 |                |                     |                |                             |             |                                   |                 |                                   |             |      |             |      |                  |           |             |                                                                                                                         |             |                                                                                              |               |                                    |              |                  |                 |               |   |
| <b>CLOB</b>      | 4 - The length of the field that is stored in the base table. The maximum length of the CLOB                            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |                |               |                 |               |              |                |             |                               |                |                                 |                |                                 |                |                     |                |                             |             |                                   |                 |                                   |             |      |             |      |                  |           |             |                                                                                                                         |             |                                                                                              |               |                                    |              |                  |                 |               |   |

## SYSIBM.SYSCOLUMNS\_HIST

| Column name | Data type                               | Description                                                                                                                                                                                                                                                                                                                                                                                    | Use |
|-------------|-----------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| LENGTH2     | INTEGER<br>NOT NULL                     | Maximum length of the data retrieved from the column. Possible values are:<br><b>0</b> Not a LOB or ROWID column<br><b>40</b> For a ROWID column, the length of the returned value<br><b>1 to 2 147 483 647 bytes</b><br>For a LOB column, the maximum length                                                                                                                                  | G   |
| NULLS       | CHAR(1)<br>NOT NULL                     | Whether the column can contain null values:<br><b>N</b> No<br><b>Y</b> Yes                                                                                                                                                                                                                                                                                                                     | G   |
| HIGH2KEY    | CHAR(8)<br>NOT NULL<br>FOR BIT DATA     | Second highest value of the column. Blank if statistics have not been gathered, or the column is an indicator column or a column of an auxiliary table. If the column has a non-character data type, the data might not be printable.                                                                                                                                                          | S   |
| LOW2KEY     | CHAR(8)<br>NOT NULL<br>FOR BIT DATA     | Second lowest value of the column. Blank if statistics have not been gathered, or the column is an indicator column or a column of an auxiliary table. If the column has a non-character data type, the data might not be printable.                                                                                                                                                           | S   |
| STATSTIME   | TIMESTAMP<br>NOT NULL                   | If RUNSTATS updated the statistics, the date and time when the last invocation of RUNSTATS updated the statistics. The default value is '0001-01-01-00.00.00.000000'. If the value is '0001-01-02-00.00.00.000000', which indicates that an ALTER TABLE statement was executed to change the length of a VARCHAR column, RUNSTATS should be run to update the statistics before they are used. | G   |
| COLCARDF    | FLOAT(8)<br>NOT NULL WITH<br>DEFAULT -1 | Estimated number of distinct values in the column. For an indicator column, this is the number of LOBs that are not null and have a length greater than zero. The value is -1 if statistics have not been gathered. The value is -2 for the first column of an index of an auxiliary table.                                                                                                    | S   |
| IBMRREQD    | CHAR(1)<br>NOT NULL<br>DEFAULT 'N'      | A value of Y indicates that the row came from the basic machine-readable material (MRM) tape. For all other values, see "Release dependency indicators" on page 1005.                                                                                                                                                                                                                          | G   |

---

## SYSIBM.SYSCONSTDEP table

Records dependencies on check constraints or user-defined defaults for a column.

| Column name | Data type                | Description                                                                                                                                                           | Use |
|-------------|--------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| BNAME       | VARCHAR(18)<br>NOT NULL  | Name of the object on which the dependency exists.                                                                                                                    | G   |
| BSHEMA      | CHAR(8)<br>NOT NULL      | Schema of the object on which the dependency exists.                                                                                                                  | G   |
| BTYPE       | CHAR(1)<br>NOT NULL      | Type of object on which the dependency exists:<br><b>F</b> Function instance<br><b>G</b> Temporary table                                                              | G   |
| DTBNAME     | VARCHAR(18)<br>NOT NULL  | Name of the table to which the dependency applies.                                                                                                                    | G   |
| DTBCREATOR  | CHAR(8)<br>NOT NULL      | Authorization ID of the owner of the table to which the dependency applies.                                                                                           | G   |
| DCONSTNAME  | VARCHAR(128)<br>NOT NULL | If DTTYPE = 'C', the unqualified name of the check constraint. If DTTYPE = 'D', a column name.                                                                        | G   |
| DTTYPE      | CHAR(1)<br>NOT NULL      | Type of object:<br><b>C</b> Check constraint<br><b>D</b> User-defined default constant                                                                                | G   |
| IBMREQD     | CHAR(1)<br>NOT NULL      | A value of Y indicates that the row came from the basic machine-readable material (MRM) tape. For all other values, see "Release dependency indicators" on page 1005. | G   |

## SYSIBM.SYSCOPY

### SYSIBM.SYSCOPY table

Contains information needed for recovery.

| Column name | Data type                           | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              | Use |
|-------------|-------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| DBNAME      | CHAR(8)<br>NOT NULL                 | Name of the database.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    | G   |
| TSNAME      | CHAR(8)<br>NOT NULL                 | Name of the target table space or index space.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           | G   |
| DSNUM       | INTEGER<br>NOT NULL                 | Data set number within the table space. For partitioned table spaces, this value corresponds to the partition number for a single partition copy, or 0 for a copy of an entire partitioned table space or index space.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   | G   |
| ICTYPE      | CHAR(1)<br>NOT NULL                 | Type of operation:<br><br>A      ALTER<br>B      REBUILD INDEX<br>D      CHECK DATA LOG(NO) (no log records for the range are available for RECOVER utility)<br>F      COPY FULL YES<br>I      COPY FULL NO<br>P      RECOVER TOCOPY or RECOVER TORBA (partial recovery point)<br>Q      QUIESCE<br>R      LOAD REPLACE LOG(YES)<br>S      LOAD REPLACE LOG(NO)<br>W      REORG LOG(NO)<br>X      REORG LOG(YES)<br>Y      LOAD LOG(NO)<br>Z      LOAD LOG(YES)<br>T      TERM UTILITY command (terminated utility)                                                                                                                                                                                                                                                                                      | G   |
| ICDATE      | CHAR(6)<br>NOT NULL                 | Date of the entry in the form <i>yymmdd</i> . For the COPYTOCOPY utility, this value is the date of the original entry, when the primary local site or primary recovery site copy was made.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              | G   |
| START_RBA   | CHAR(6)<br>NOT NULL<br>FOR BIT DATA | A 48-bit positive integer that contains the LRSN of a point in the DB2 recovery log. (The LRSN is the RBA in a non-data-sharing environment.)<br><br><ul style="list-style-type: none"> <li>• For ICTYPE I or F, the starting point for all updates since the image copy was taken</li> <li>• For ICTYPE P, the point after the log-apply phase of point-in-time recovery</li> <li>• For ICTYPE Q, the point after all data sets have been successfully quiesced</li> <li>• For ICTYPE R or S, the end of the log before the start of the LOAD utility and before any data is changed</li> <li>• For ICTYPE T, the end of the log when the utility is terminated</li> <li>• For other values of ICTYPE, the end of the log before the start of the RELOAD phase of the LOAD or REORG utility.</li> </ul> | G   |
| FILESEQNO   | INTEGER<br>NOT NULL                 | Tape file sequence number of the copy.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   | G   |
| DEVTYPE     | CHAR(8)<br>NOT NULL                 | Device type the copy is on.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              | G   |
| IBMREQD     | CHAR(1)<br>NOT NULL                 | A value of Y indicates that the row came from the basic machine-readable material (MRM) tape. For all other values, see "Release dependency indicators" on page 1005.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    | G   |
| DSNAME      | CHAR(44)<br>NOT NULL                | For ICTYPE='P' (RECOVER TOCOPY only), 'I', or 'F', DSNAME contains the data set name. Otherwise, DSNAME contains the name of the database and table space or index space in the form, <i>database-name.space-name</i> , or DSNAME is blank for any row migrated from a DB2 release prior to Version 4.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   | G   |

| Column name | Data type                             | Description                                                                                                                                                                                                                                                                                                                                                                                          | Use |
|-------------|---------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| ICTIME      | CHAR(6)<br>NOT NULL                   | The time at which this row was inserted, in the form <i>hhmmss</i> . The insertion takes place after the completion of the operation that the row represents. ICTIME is blank for any row which was migrated from Version 1 Release 1 of DB2. For the COPYTOCOPY utility, this value is the time when the row was inserted for the primary local site or primary recovery site copy.                 | G   |
| SHRLEVEL    | CHAR(1)<br>NOT NULL                   | SHRLEVEL parameter on COPY (for ICTYPE F or I only):<br><b>C</b> Change<br><b>R</b> Reference<br><b>blank</b> Does not describe an image copy or was migrated from Version 1 Release 1 of DB2.                                                                                                                                                                                                       | G   |
| DSVOLSER    | VARCHAR(1784)<br>NOT NULL             | The volume serial numbers of the data set. A list of 6-byte numbers separated by commas. Blank if the data set is cataloged.                                                                                                                                                                                                                                                                         | G   |
| TIMESTAMP   | TIMESTAMP<br>NOT NULL WITH<br>DEFAULT | The date and time when the row was inserted. This is the date and time recorded in ICDATE and ICTIME. The use of TIMESTAMP is recommended over that of ICDATE and ICTIME, because the latter two columns may not be supported in later DB2 releases. For the COPYTOCOPY utility, this value is the date and time when the row was inserted for the primary local site or primary recovery site copy. | G   |
| ICBACKUP    | CHAR(2)<br>NOT NULL WITH<br>DEFAULT   | Specifies the type of image copy contained in the data set:<br><b>blank</b> LOCALSITE primary copy (first data set named with COPYDDN)<br><b>LB</b> LOCALSITE backup copy (second data set named with COPYDDN)<br><b>RP</b> RECOVERYSITE primary copy (first data set named with RECOVERYDDN)<br><b>RB</b> RECOVERYSITE backup copy (second data set named with RECOVERYDDN)                         | G   |
| ICUNIT      | CHAR(1)<br>NOT NULL WITH<br>DEFAULT   | Indicates the media that the image copy data set is stored on:<br><b>D</b> DASD<br><b>T</b> Tape<br><b>blank</b> Medium is neither tape nor DASD, the image copy is from a DB2 release prior to Version 2 Release 3, or ICTYPE is not 'I' or 'F'.                                                                                                                                                    | G   |

## SYSIBM.SYSCOPY

| Column name      | Data type                                           | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               | Use |
|------------------|-----------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| STYPE            | CHAR(1)<br>NOT NULL WITH<br>DEFAULT                 | When ICTYPE=A (the length of an indexed VARCHAR column in a table was increased), the value is V.<br><br>When ICTYPE=F, the values are:<br><b>blank</b> DB2 image copy<br><b>C</b> DFSMS concurrent copy ("I" instance of the table space)<br><b>J</b> DFSMS concurrent copy ("J" instance of the table space)<br><b>R</b> LOAD REPLACE(YES)<br><b>S</b> LOAD REPLACE(NO)<br><b>W</b> REORG LOG(NO)<br><b>X</b> REORG LOG(YES)<br><br>The MERGECOPY utility, when used to merge an embedded copy with subsequent incremental copies, also produces a record that contains ICTYPE=F and the STYPE of the original image copy (R, S, W, or X).                                                              | G   |
| #<br>#<br>#<br># |                                                     | When ICTYPE=P and the operation is RECOVER TORBA LOGONLY, the value is L.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |     |
|                  |                                                     | When ICTYPE=Q and option WRITE(YES) is in effect when the quiesce point is taken, the value is W.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |     |
|                  |                                                     | When ICTYPE=R, S, W, or X and the operation is resetting REORG pending status, the value is A.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |     |
|                  |                                                     | When ICTYPE=T, this field indicates which COPY utility was terminated by the TERM UTILITY command or the START DATABASE command with the ACCESS(FORCE) option. The values are:<br><b>F</b> COPY FULL YES<br><b>I</b> COPY FULL NO                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |     |
|                  |                                                     | For other values of ICTYPE, the value is blank.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |     |
| PIT_RBA          | CHAR(6)<br>NOT NULL WITH<br>DEFAULT<br>FOR BIT DATA | When ICTYPE=P, this field contains the LRSN for the point in the DB2 log. (The LRSN is the RBA in a non-data-sharing environment). For other ICTYPES, this field is X'000000000000'.<br><br>When ICTYPE=P, this field indicates the stop location of a point-in-time recovery. If a record contains ICTYPE=P and PIT_RBA=X'000000000000', the copy pending status is active and a full image copy is required. If such a record is encountered during fallback processing of RECOVER, the recover job fails, and a point-in-time recovery is required. PIT_RBA can be zero if the point-in-time recovery is completed by the fall-back processing of RECOVER, or if ICTYPE=P from a prior release of DB2. | G   |
| GROUP_MEMBER     | CHAR(8)<br>NOT NULL WITH<br>DEFAULT                 | The DB2 data sharing member name of the DB2 subsystem that performed the operation. This column is blank if the DB2 subsystem was not in a DB2 data sharing environment at the time the operation was performed.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          | G   |
| OTYPE            | CHAR(1)<br>NOT NULL WITH<br>DEFAULT 'T'             | Type of object that the recovery information is for:<br><b>I</b> Index space<br><b>T</b> Table space                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      | G   |
| LOWDSNUM         | INTEGER<br>NOT NULL WITH<br>DEFAULT                 | Partition number of the lowest partition in the range for SYSCOPY records created for REORG and LOAD REPLACE for resetting a REORG pending status. Version number of an index for SYSCOPY records created for a COPY (ICTYPE=F) of an index space (OTYPE=I). (An index is versioned when a VARCHAR column in the index key is lengthened.) The column is valid only for these uses.                                                                                                                                                                                                                                                                                                                       | G   |
| HIGHDSNUM        | INTEGER<br>NOT NULL WITH<br>DEFAULT                 | Partition number of the highest partition in the range. This column is valid only for SYSCOPY records created for REORG and LOAD REPLACE for resetting REORG pending status.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              | G   |

| Column name | Data type                               | Description                                                                                                                                     | Use |
|-------------|-----------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| COPYPAGESF  | FLOAT(8)<br>NOT NULL WITH<br>DEFAULT -1 | Number of pages written to the copy data set. For inline copies, this number might include pages appearing more than once in the copy data set. | G   |
| NPAGESF     | FLOAT(8)<br>NOT NULL WITH<br>DEFAULT -1 | The number of pages in the tablespace or index at the time of COPY. This number might include preformatted pages that are not actually copied.  | G   |
| CPAGESF     | FLOAT(8)<br>NOT NULL WITH<br>DEFAULT -1 | Total number of changed pages.                                                                                                                  | G   |
| JOBNAME     | CHAR(8)<br>NOT NULL WITH<br>DEFAULT     | Job name of the utility.                                                                                                                        | G   |
| AUTHID      | CHAR(8)<br>NOT NULL WITH<br>DEFAULT     | Authorization ID of the utility.                                                                                                                | G   |

## SYSIBM.SYSDATABASE

### SYSIBM.SYSDATABASE table

Contains one row for each database, except for database DSNDB01.

| Column name     | Data type                               | Description                                                                                                                                                                                                                                                                                                                                                                                             | Use |
|-----------------|-----------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| NAME            | CHAR(8)<br>NOT NULL                     | Database name.                                                                                                                                                                                                                                                                                                                                                                                          | G   |
| CREATOR         | CHAR(8)<br>NOT NULL                     | Authorization ID of the owner of the database.                                                                                                                                                                                                                                                                                                                                                          | G   |
| STGROUP         | CHAR(8)<br>NOT NULL                     | Name of the default storage group of the database; blank for a system database.                                                                                                                                                                                                                                                                                                                         | G   |
| BPOOL           | CHAR(8)<br>NOT NULL                     | Name of the default buffer pool of the table space; blank for a system table space.                                                                                                                                                                                                                                                                                                                     | G   |
| DBID            | SMALLINT<br>NOT NULL                    | Internal identifier of the database. If there were 32511 databases or more when this database was created, the DBID is a negative number.                                                                                                                                                                                                                                                               | S   |
| IBMREQD         | CHAR(1)<br>NOT NULL                     | A value of Y indicates that the row came from the basic machine-readable material (MRM) tape. For all other values, see "Release dependency indicators" on page 1005.                                                                                                                                                                                                                                   | G   |
| CREATEDBY       | CHAR(8)<br>NOT NULL WITH<br>DEFAULT     | Primary authorization ID of the user who created the database.                                                                                                                                                                                                                                                                                                                                          | G   |
|                 | CHAR(1)<br>NOT NULL WITH<br>DEFAULT     | Not used                                                                                                                                                                                                                                                                                                                                                                                                | N   |
| TIMESTAMP       | TIMESTAMP<br>NOT NULL WITH<br>DEFAULT   | The value is '0001-01-01-00.00.000000'.                                                                                                                                                                                                                                                                                                                                                                 | G   |
| TYPE            | CHAR(1)<br>NOT NULL WITH<br>DEFAULT     | Type of database:<br><b>blank</b> Not a work file database or a TEMP database.<br><b>T</b> A TEMP database. The database was created with the AS TEMP clause, which indicates it is used for declared temporary tables.<br><b>W</b> A work file database. The database is DSNDB07, or it was created with the WORKFILE clause and used as a work file database by a member of a DB2 data sharing group. | G   |
| GROUP_MEMBER    | CHAR(8)<br>NOT NULL WITH<br>DEFAULT     | The DB2 data sharing member name of the DB2 subsystem that uses this work file database. This column is blank if the work file database was not created in a DB2 data sharing environment, or if the database is not a work file database as indicated by the TYPE column.                                                                                                                              | G   |
| CREATEDTS       | TIMESTAMP<br>NOT NULL WITH<br>DEFAULT   | Time when the CREATE statement was executed for the database. For DSNDB04 and DSNDB06, the value is '1985-04-01-00.00.00.000000'.                                                                                                                                                                                                                                                                       | G   |
| ALTEREDTS       | TIMESTAMP<br>NOT NULL WITH<br>DEFAULT   | Time when the most recent ALTER DATABASE statement was applied. If no ALTER DATABASE statement has been applied, ALTEREDTS has the value of CREATEDTS.                                                                                                                                                                                                                                                  | G   |
| ENCODING_SCHEME | CHAR(1)<br>NOT NULL WITH<br>DEFAULT 'E' | Default encoding scheme for the database:<br><b>E</b> EBCDIC<br><b>A</b> ASCII<br><b>U</b> UNICODE<br><b>blank</b> For DSNDB04, a work file database, and a TEMP database.                                                                                                                                                                                                                              | G   |
| SBCS_CCSID      | INTEGER<br>NOT NULL WITH<br>DEFAULT     | Default SBCS CCSID for the database. For a TEMP database or a database created in a DB2 release prior to Version 5, the value is 0.                                                                                                                                                                                                                                                                     | G   |
| DBCS_CCSID      | INTEGER<br>NOT NULL WITH<br>DEFAULT     | Default DBCS CCSID for the database. For a TEMP database or a database created in a DB2 release prior to Version 5, the value is 0.                                                                                                                                                                                                                                                                     | G   |

**SYSIBM.SYSDATABASE**

| Column name | Data type                                 | Description                                                                                                                        | Use |
|-------------|-------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------|-----|
| MIXED_CCSID | INTEGER<br>NOT NULL WITH<br>DEFAULT       | Default mixed CCSID for the database. For a TEMP database or database created in a DB2 release prior to Version 5, the value is 0. | G   |
| INDEXBP     | CHAR(8)<br>NOT NULL WITH<br>DEFAULT 'BPO' | Name of the default buffer pool for indexes.                                                                                       | G   |

## SYSIBM.SYSDATATYPES

### SYSIBM.SYSDATATYPES table

Contains one row for each distinct type defined to the system.

| Column name     | Data type                | Description                                                                                                                                                                                                                                                            | Use |
|-----------------|--------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| SCHEMA          | CHAR(8)<br>NOT NULL      | Schema of the distinct type.                                                                                                                                                                                                                                           | G   |
| OWNER           | CHAR(8)<br>NOT NULL      | Owner of the distinct type.                                                                                                                                                                                                                                            | G   |
| NAME            | CHAR(18)<br>NOT NULL     | Name of the distinct type.                                                                                                                                                                                                                                             | G   |
| CREATEDBY       | CHAR(8)<br>NOT NULL      | Authorization ID under which the distinct type was created.                                                                                                                                                                                                            | G   |
| SOURCESCHEMA    | CHAR(8)<br>NOT NULL      | Schema of the source data type.                                                                                                                                                                                                                                        | G   |
| SOURCETYPE      | CHAR(18)<br>NOT NULL     | Name of the source type.                                                                                                                                                                                                                                               | G   |
| METATYPE        | CHAR(1)<br>NOT NULL      | The class of data type:<br><b>T</b> Distinct type                                                                                                                                                                                                                      | G   |
| DATATYPEID      | INTEGER<br>NOT NULL      | Internal identifier of the distinct type.                                                                                                                                                                                                                              | S   |
| SOURCETYPEID    | INTEGER<br>NOT NULL      | Internal ID of the built-in data type upon which the distinct type is sourced.                                                                                                                                                                                         | S   |
| LENGTH          | INTEGER<br>NOT NULL      | Maximum length or precision of a distinct type that is sourced on the IBM-defined DECIMAL data type.                                                                                                                                                                   | G   |
| SCALE           | SMALLINT<br>NOT NULL     | Scale for a distinct type that is sourced on the IBM-defined DECIMAL type. For all other distinct types, the value is 0.                                                                                                                                               | G   |
| SUBTYPE         | CHAR(1)<br>NOT NULL      | Subtype of the distinct type, which is based on the subtype of the source type:<br><b>B</b> The subtype is FOR BIT DATA.<br><b>S</b> The subtype is FOR SBCS DATA.<br><b>M</b> The subtype is FOR MIXED DATA.<br><b>blank</b> The source type is not a character type. | G   |
| CREATEDTS       | TIMESTAMP<br>NOT NULL    | Time when the distinct type was created.                                                                                                                                                                                                                               | G   |
| ENCODING_SCHEME | CHAR(1)<br>NOT NULL      | Encoding scheme of the distinct type:<br><b>A</b> ASCII<br><b>E</b> EBCDIC<br><b>U</b> UNICODE                                                                                                                                                                         | G   |
| IBMREQD         | CHAR(1)<br>NOT NULL      | A value of Y indicates that the row came from the basic machine-readable material (MRM) tape. For all other values, see "Release dependency indicators" on page 1005.                                                                                                  | G   |
| REMARKS         | VARCHAR(254)<br>NOT NULL | A character string provided by the user with the COMMENT ON statement.                                                                                                                                                                                                 | G   |

## SYSIBM.SYSDBAUTH table

Records the privileges that are held by users over databases.

| Column name   | Data type            | Description                                                                                                                                                                                                                                                                                           | Use |
|---------------|----------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| GRANTOR       | CHAR(8)<br>NOT NULL  | Authorization ID of the user who granted the privileges. Could also be PUBLIC or PUBLIC followed by an asterisk. <sup>46</sup>                                                                                                                                                                        | G   |
| GRANTEE       | CHAR(8)<br>NOT NULL  | Application ID of the user who holds the privilege. Could also be PUBLIC for a grant to PUBLIC.                                                                                                                                                                                                       | G   |
| NAME          | CHAR(8)<br>NOT NULL  | Database name.                                                                                                                                                                                                                                                                                        | G   |
|               | CHAR(12)<br>NOT NULL | Internal use only                                                                                                                                                                                                                                                                                     | I   |
|               | CHAR(6)<br>NOT NULL  | Not used.                                                                                                                                                                                                                                                                                             | N   |
|               | CHAR(8)<br>NOT NULL  | Not used.                                                                                                                                                                                                                                                                                             | N   |
|               | CHAR(1)<br>NOT NULL  | Not used                                                                                                                                                                                                                                                                                              | N   |
| AUTHHOWGOT    | CHAR(1)<br>NOT NULL  | Authorization level of the user from whom the privileges were received. This authorization level is not necessarily the highest authorization level of the grantor.<br><br><b>blank</b> Not applicable<br><b>C</b> DBCTL<br><b>D</b> DBADM<br><b>L</b> SYSCTRL<br><b>M</b> DBMAINT<br><b>S</b> SYSADM | G   |
| CREATETABAUTH | CHAR(1)<br>NOT NULL  | Whether the GRANTEE can create tables within the database:<br><br><b>blank</b> Privilege is not held<br><b>G</b> Privilege held with the GRANT option<br><b>Y</b> Privilege is held without the GRANT option                                                                                          | G   |
| CREATETSAUTH  | CHAR(1)<br>NOT NULL  | Whether the GRANTEE can create table spaces within the database:<br><br><b>blank</b> Privilege is not held<br><b>G</b> Privilege held with the GRANT option<br><b>Y</b> Privilege is held without the GRANT option                                                                                    | G   |
| DBADMAUTH     | CHAR(1)<br>NOT NULL  | Whether the GRANTEE has DBADM authority over the database:<br><br><b>blank</b> Privilege is not held<br><b>G</b> Privilege held with the GRANT option<br><b>Y</b> Privilege is held without the GRANT option                                                                                          | G   |
| DBCTRLAUTH    | CHAR(1)<br>NOT NULL  | Whether the GRANTEE has DBCTRL authority over the database:<br><br><b>blank</b> Privilege is not held<br><b>G</b> Privilege held with the GRANT option<br><b>Y</b> Privilege is held without the GRANT option                                                                                         | G   |
| DBMAINTAUTH   | CHAR(1)<br>NOT NULL  | Whether the GRANTEE has DBMAINT authority over the database:<br><br><b>blank</b> Privilege is not held<br><b>G</b> Privilege held with the GRANT option<br><b>Y</b> Privilege is held without the GRANT option                                                                                        | G   |
| DISPLAYDBAUTH | CHAR(1)<br>NOT NULL  | Whether the GRANTEE can issue the DISPLAY command for the database:<br><br><b>blank</b> Privilege is not held<br><b>G</b> Privilege held with the GRANT option<br><b>Y</b> Privilege is held without the GRANT option                                                                                 | G   |

46. PUBLIC followed by an asterisk (PUBLIC\*) denotes PUBLIC AT ALL LOCATIONS. For the conditions where GRANTOR can be PUBLIC or PUBLIC\*, see Part 3 (Volume 1) of *DB2 Administration Guide*.

## SYSIBM.SYSDBAUTH

| Column name   | Data type                             | Description                                                                                                                                                                                                                                              | Use |
|---------------|---------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| DROPAUTH      | CHAR(1)<br>NOT NULL                   | Whether the GRANTEE can issue the ALTER DATABASE and DROP DATABASE statement:<br><b>blank</b> Privilege is not held<br><b>G</b> Privilege held with the GRANT option<br><b>Y</b> Privilege is held without the GRANT option                              | G   |
| IMAGCOPYAUTH  | CHAR(1)<br>NOT NULL                   | Whether the GRANTEE can use the COPY, MERGECOPY, MODIFY, and QUIESCE utilities on the database:<br><b>blank</b> Privilege is not held<br><b>G</b> Privilege held with the GRANT option<br><b>Y</b> Privilege is held without the GRANT option            | G   |
| LOADAUTH      | CHAR(1)<br>NOT NULL                   | Whether the GRANTEE can use the LOAD utility to load tables in the database:<br><b>blank</b> Privilege is not held<br><b>G</b> Privilege held with the GRANT option<br><b>Y</b> Privilege is held without the GRANT option                               | G   |
| REORGAUTH     | CHAR(1)<br>NOT NULL                   | Whether the GRANTEE can use the REORG utility to reorganize table spaces and indexes in the database:<br><b>blank</b> Privilege is not held<br><b>G</b> Privilege held with the GRANT option<br><b>Y</b> Privilege is held without the GRANT option      | G   |
| RECOVERDBAUTH | CHAR(1)<br>NOT NULL                   | Whether the GRANTEE can use the RECOVER and REPORT utilities on table spaces in the database:<br><b>blank</b> Privilege is not held<br><b>G</b> Privilege held with the GRANT option<br><b>Y</b> Privilege is held without the GRANT option              | G   |
| REPAIRAUTH    | CHAR(1)<br>NOT NULL                   | Whether the GRANTEE can use the DIAGNOSE and REPAIR utilities on table spaces and indexes in the database:<br><b>blank</b> Privilege is not held<br><b>G</b> Privilege held with the GRANT option<br><b>Y</b> Privilege is held without the GRANT option | G   |
| STARTDBAUTH   | CHAR(1)<br>NOT NULL                   | Whether the GRANTEE can use the START command against the database:<br><b>blank</b> Privilege is not held<br><b>G</b> Privilege held with the GRANT option<br><b>Y</b> Privilege is held without the GRANT option                                        | G   |
| STATSAUTH     | CHAR(1)<br>NOT NULL                   | Whether the GRANTEE can use the CHECK and RUNSTATS utilities against the database:<br><b>blank</b> Privilege is not held<br><b>G</b> Privilege held with the GRANT option<br><b>Y</b> Privilege is held without the GRANT option                         | G   |
| STOPAUTH      | CHAR(1)<br>NOT NULL                   | Whether the GRANTEE can issue the STOP command against the database:<br><b>blank</b> Privilege is not held<br><b>G</b> Privilege held with the GRANT option<br><b>Y</b> Privilege is held without the GRANT option                                       | G   |
| IBMREQD       | CHAR(1)<br>NOT NULL                   | A value of Y indicates that the row came from the basic machine-readable material (MRM) tape. For all other values, see "Release dependency indicators" on page 1005.                                                                                    | G   |
| GRANTEDTS     | TIMESTAMP<br>NOT NULL WITH<br>DEFAULT | Time when the GRANT statement was executed.                                                                                                                                                                                                              | G   |

**SYSIBM.SYSDBRM table**

Contains one row for each DBRM of each application plan.

| Column name | Data type                           | Description                                                                                                                                                                                                                                                                                                                                                                                                                    | Use |
|-------------|-------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| NAME        | CHAR(8)<br>NOT NULL                 | Name of the DBRM.                                                                                                                                                                                                                                                                                                                                                                                                              | G   |
| TIMESTAMP   | CHAR(8)<br>NOT NULL<br>FOR BIT DATA | Consistency token.                                                                                                                                                                                                                                                                                                                                                                                                             | S   |
| PDSNAME     | CHAR(44)<br>NOT NULL                | Name of the partitioned data set of which the DBRM is a member.                                                                                                                                                                                                                                                                                                                                                                | G   |
| PLNAME      | CHAR(8)<br>NOT NULL                 | Name of the application plan of which this DBRM is a part.                                                                                                                                                                                                                                                                                                                                                                     | G   |
| PLCREATOR   | CHAR(8)<br>NOT NULL                 | Authorization ID of the owner of the application plan.                                                                                                                                                                                                                                                                                                                                                                         | G   |
| PRECOMPTIME | CHAR(8)<br>NOT NULL                 | Time of precompilation in the form <i>hhmmssth</i> .<br><br>If the LEVEL precompiler option is used, then this value does not represent the precompile time.                                                                                                                                                                                                                                                                   | G   |
| PRECOMPDATE | CHAR(6)<br>NOT NULL                 | Date of precompilation in the form <i>yymmdd</i> .<br><br>If the LEVEL precompiler option is used, then this value does not represent the precompile date.                                                                                                                                                                                                                                                                     | G   |
| QUOTE       | CHAR(1)<br>NOT NULL                 | SQL string delimiter for the SQL statements in the DBRM:<br><b>N</b> Apostrophe<br><b>Y</b> Quotation mark                                                                                                                                                                                                                                                                                                                     | G   |
| COMMA       | CHAR(1)<br>NOT NULL                 | Decimal point representation for SQL statements in the DBRM:<br><b>N</b> Period<br><b>Y</b> Comma                                                                                                                                                                                                                                                                                                                              | G   |
| HOSTLANG    | CHAR(1)<br>NOT NULL                 | The host language used:<br><b>B</b> Assembler language<br><b>C</b> OS/VS COBOL<br><b>D</b> C<br><b>F</b> Fortran<br><b>P</b> PL/I<br><b>2</b> VS COBOL II or IBM COBOL Release 1 (formerly called COBOL/370)<br><b>3</b> IBM COBOL (Release 2 or subsequent releases)<br><b>4</b> C++                                                                                                                                          | G   |
| IBMREQD     | CHAR(1)<br>NOT NULL                 | A value of Y indicates that the row came from the basic machine-readable material (MRM) tape. For all other values, see "Release dependency indicators" on page 1005.                                                                                                                                                                                                                                                          | G   |
| CHARSET     | CHAR(1)<br>NOT NULL WITH<br>DEFAULT | Indicates whether the system CCSID for SBCS data was 290 (Katakana) when the program was precompiled:<br><b>A</b> No<br><b>K</b> Yes                                                                                                                                                                                                                                                                                           | G   |
| MIXED       | CHAR(1)<br>NOT NULL WITH<br>DEFAULT | Indicates if mixed data was in effect when the application program was precompiled (for more on when mixed data is in effect, see "Character strings" on page 49):<br><b>N</b> No<br><b>Y</b> Yes                                                                                                                                                                                                                              | G   |
| # DEC31     | CHAR(1)<br>NOT NULL WITH<br>DEFAULT | Indicates whether DEC31 was in effect when the program was precompiled (for more on when DEC31 is in effect, see "Arithmetic with two decimal operands" on page 114). If this option is specified at precompile time in the form D31.s, where s is a scale from 1 to 9, then DEC31 is in effect. If this option is specified at precompile time in the form D15.s, then DEC15 is in effect.<br><b>blank</b> No<br><b>Y</b> Yes | G   |

## SYSIBM.SYSDBRM

| Column name | Data type                               | Description                         | Use |
|-------------|-----------------------------------------|-------------------------------------|-----|
| VERSION     | VARCHAR(64)<br>NOT NULL WITH<br>DEFAULT | Version identifier for the DBRM.    | G   |
| PRECOMPTS   | TIMESTAMP<br>NOT NULL WITH<br>DEFAULT   | Time when the DBRM was precompiled. | G   |

---

**SYSIBM.SYSDUMMY1 table**

Contains one row. The table is used for SQL statements in which a table reference is required, but the contents of the table are not important.

| Column name | Data type           | Description                                                                                                                                                           | Use |
|-------------|---------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| IBMREQD     | CHAR(1)<br>NOT NULL | A value of Y indicates that the row came from the basic machine-readable material (MRM) tape. For all other values, see "Release dependency indicators" on page 1005. | G   |

## SYSIBM.SYSFIELDS

### SYSIBM.SYSFIELDS table

Contains one row for every column that has a field procedure.

| Column name | Data type                                 | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                   | Use |
|-------------|-------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| TBCREATOR   | CHAR(8)<br>NOT NULL                       | Authorization ID of the owner of the table that contains the column.                                                                                                                                                                                                                                                                                                                                                                                                          | G   |
| TBNAME      | VARCHAR(18)<br>NOT NULL                   | Name of the table that contains the column.                                                                                                                                                                                                                                                                                                                                                                                                                                   | G   |
| COLNO       | SMALLINT<br>NOT NULL                      | Numeric place of this column in the table.                                                                                                                                                                                                                                                                                                                                                                                                                                    | G   |
| NAME        | VARCHAR(18)<br>NOT NULL                   | Name of the column.                                                                                                                                                                                                                                                                                                                                                                                                                                                           | G   |
| FLDTYPE     | CHAR(8)<br>NOT NULL                       | Data type of the encoded values in the field <sup>47</sup> :<br><b>INTEGER</b> Large integer<br><b>SMALLINT</b> Small integer<br><b>FLOAT</b> Floating-point<br><b>CHAR</b> Fixed-length character string<br><b>VARCHAR</b> Varying-length character string<br><b>DECIMAL</b> Decimal<br><b>GRAPHIC</b> Fixed-length graphic string<br><b>VARG</b> Varying-length graphic string                                                                                              | G   |
| LENGTH      | SMALLINT<br>NOT NULL                      | The length attribute of the field; or, for a decimal field, its precision <sup>47</sup> . The number does not include the internal prefixes that can be used to record actual length and null state.<br><b>INTEGER</b> 4<br><b>SMALLINT</b> 2<br><b>FLOAT</b> 8<br><b>CHAR</b> Length of string<br><b>VARCHAR</b> Maximum length of string<br><b>DECIMAL</b> Precision of number<br><b>GRAPHIC</b> Number of DBCS characters<br><b>VARG</b> Maximum number of DBCS characters | G   |
| SCALE       | SMALLINT<br>NOT NULL                      | Scale if FLDTYPE is DECIMAL; otherwise, the value is 0.                                                                                                                                                                                                                                                                                                                                                                                                                       | G   |
| FLDPROC     | CHAR(8)<br>NOT NULL                       | For a row describing a field procedure, the name of the procedure <sup>47</sup> .                                                                                                                                                                                                                                                                                                                                                                                             | G   |
| WORKAREA    | SMALLINT<br>NOT NULL                      | For a row describing a field procedure, the size, in bytes, of the work area required for the encoding and decoding of the procedure <sup>47</sup> .                                                                                                                                                                                                                                                                                                                          | G   |
| IBMREQD     | CHAR(1)<br>NOT NULL                       | A value of Y indicates that the row came from the basic machine-readable material (MRM) tape. For all other values, see "Release dependency indicators" on page 1005.                                                                                                                                                                                                                                                                                                         | G   |
| EXTPARML    | SMALLINT<br>NOT NULL                      | For a row describing a field procedure, the length of the field procedure parameter value block <sup>47</sup> .                                                                                                                                                                                                                                                                                                                                                               | G   |
| PARMLIST    | VARCHAR(254)<br>NOT NULL                  | For a row describing a field procedure, the parameter list following FIELDPROC in the statement that created the column, with insignificant blanks removed <sup>47</sup> .                                                                                                                                                                                                                                                                                                    | G   |
| EXTPARM     | VARCHAR(1530)<br>NOT NULL<br>FOR BIT DATA | For a row describing a field procedure, the parameter value block of the field procedure (the control block passed to the field procedure when it is invoked) <sup>47</sup> .                                                                                                                                                                                                                                                                                                 | G   |

47. Some columns might contain statistical values from a prior release.

---

## SYSIBM.SYSFOREIGNKEYS table

Contains one row for every column of every foreign key.

| Column name | Data type               | Description                                                                                                                                                           | Use |
|-------------|-------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| CREATOR     | CHAR(8)<br>NOT NULL     | Authorization ID of the owner of the table that contains the column.                                                                                                  | G   |
| TBNAME      | VARCHAR(18)<br>NOT NULL | Name of the table that contains the column.                                                                                                                           | G   |
| RELNAME     | CHAR(8)<br>NOT NULL     | Constraint name for the constraint for which the column is part of the foreign key.                                                                                   | G   |
| COLNAME     | VARCHAR(18)<br>NOT NULL | Name of the column.                                                                                                                                                   | G   |
| COLNO       | SMALLINT<br>NOT NULL    | Numeric place of the column in its table.                                                                                                                             | G   |
| COLSEQ      | SMALLINT<br>NOT NULL    | Numeric place of the column in the foreign key.                                                                                                                       | G   |
| IBMREQD     | CHAR(1)<br>NOT NULL     | A value of Y indicates that the row came from the basic machine-readable material (MRM) tape. For all other values, see "Release dependency indicators" on page 1005. | G   |

## SYSIBM.SYSINDEXES

### SYSIBM.SYSINDEXES table

Contains one row for every index.

| Column name | Data type               | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   | Use |
|-------------|-------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| NAME        | VARCHAR(18)<br>NOT NULL | Name of the index.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            | G   |
| CREATOR     | CHAR(8)<br>NOT NULL     | Authorization ID of the owner of the index.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   | G   |
| TBNAME      | VARCHAR(18)<br>NOT NULL | Name of the table on which the index is defined.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              | G   |
| TBCREATOR   | CHAR(8)<br>NOT NULL     | Authorization ID of the owner of the table.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   | G   |
| UNIQUERULE  | CHAR(1)<br>NOT NULL     | Whether the index is unique:<br><br>D No (duplicates are allowed)<br>U Yes<br>P Yes, and it is a primary index (As in prior releases of DB2, a value of P is used for primary keys that are used to enforce a referential constraint.)<br>C Yes, and it is an index used to enforce UNIQUE constraint<br>N Yes, and it is defined with UNIQUE WHERE NOT NULL<br>R Yes, and it is an index used to enforce the uniqueness of a non-primary parent key<br>G Yes, and it is an index used to enforce the uniqueness of values in a column defined as ROWID GENERATED BY DEFAULT. | G   |
| COLCOUNT    | SMALLINT<br>NOT NULL    | The number of columns in the key.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             | G   |
| CLUSTERING  | CHAR(1)<br>NOT NULL     | Whether CLUSTER was specified when the index was created:<br><br>N No<br>Y Yes                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                | G   |
| CLUSTERED   | CHAR(1)<br>NOT NULL     | Whether the table is actually clustered by the index:<br><br>N A significant number of rows are not in clustering order, or statistics have not been gathered.<br>Y Most of the rows are in clustering order.<br>blank Not applicable.<br><br>This is an updatable column that can also be changed by the RUNSTATS utility.                                                                                                                                                                                                                                                   | G   |
| DBID        | SMALLINT<br>NOT NULL    | Internal identifier of the database.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          | S   |
| OBID        | SMALLINT<br>NOT NULL    | Internal identifier of the index fan set descriptor.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          | S   |
| ISOBID      | SMALLINT<br>NOT NULL    | Internal identifier of the index page set descriptor.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         | S   |
| DBNAME      | CHAR(8)<br>NOT NULL     | Name of the database that contains the index.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 | G   |
| INDEXSPACE  | CHAR(8)<br>NOT NULL     | Name of the index space.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      | G   |
|             | INTEGER<br>NOT NULL     | Not used                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      | N   |
|             | INTEGER<br>NOT NULL     | Not used                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      | N   |
| NLEAF       | INTEGER<br>NOT NULL     | Number of active leaf pages in the index. The value is -1 if statistics have not been gathered. This is an updatable column.                                                                                                                                                                                                                                                                                                                                                                                                                                                  | S   |
| NLEVELS     | SMALLINT<br>NOT NULL    | Number of levels in the index tree. If the index is partitioned, it is the maximum of the number of levels in the index tree for all the partitions. The value is -1 if statistics have not been gathered. This is an updatable column.                                                                                                                                                                                                                                                                                                                                       | S   |

## SYSIBM.SYSINDEXES

| Column name   | Data type                             | Description                                                                                                                                                                                                                                                                                                                                                                   | Use |
|---------------|---------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| BPOOL         | CHAR(8)<br>NOT NULL                   | Name of the buffer pool used for the index.                                                                                                                                                                                                                                                                                                                                   | G   |
| PGSIZE        | SMALLINT<br>NOT NULL                  | Size, in bytes, of the leaf pages in the index: 256, 512, 1024, 2048, or 4096                                                                                                                                                                                                                                                                                                 | G   |
| ERASERULE     | CHAR(1)<br>NOT NULL                   | Whether the data sets are erased when dropped. The value is meaningless if the index is partitioned:<br><br>N No<br>Y Yes                                                                                                                                                                                                                                                     | G   |
|               |                                       | Not used                                                                                                                                                                                                                                                                                                                                                                      | N   |
| CLOSERULE     | CHAR(1)<br>NOT NULL                   | Whether the data sets are candidates for closure when the limit on the number of open data sets is reached:<br><br>N No<br>Y Yes                                                                                                                                                                                                                                              | G   |
| SPACE         | INTEGER<br>NOT NULL                   | Number of kilobytes of DASD storage allocated to the index, as determined by the last execution of the STOSPACE utility. The value is 0 if the index is not related to a storage group, or if STOSPACE has not been run. If the index space is partitioned, the value is the total kilobytes of DASD storage allocated to all partitions that are defined in a storage group. | G   |
| IBMREQD       | CHAR(1)<br>NOT NULL                   | A value of Y indicates that the row came from the basic machine-readable material (MRM) tape. For all other values, see "Release dependency indicators" on page 1005.                                                                                                                                                                                                         | G   |
| CLUSTERRATIO  | SMALLINT<br>NOT NULL WITH<br>DEFAULT  | Percentage of rows that are in clustering order. For a partitioning index, it is the weighted average of all index partitions in terms of the number of rows in the partition. The value is 0 if statistics have not been gathered. The value is -2 if the index is for an auxiliary table. This is an updatable column.                                                      | S   |
| CREATEDBY     | CHAR(8)<br>NOT NULL WITH<br>DEFAULT   | Primary authorization ID of the user who created the index.                                                                                                                                                                                                                                                                                                                   | G   |
|               | SMALLINT<br>NOT NULL                  | Internal use only                                                                                                                                                                                                                                                                                                                                                             | I   |
|               | SMALLINT<br>NOT NULL                  | Not used                                                                                                                                                                                                                                                                                                                                                                      | N   |
| STATSTIME     | TIMESTAMP<br>NOT NULL WITH<br>DEFAULT | If RUNSTATS updated the statistics, the date and time when the last invocation of RUNSTATS updated the statistics. The default value is '0001-01-01-00.00.00.000000'. This is an updatable column.                                                                                                                                                                            | G   |
| INDEXTYPE     | CHAR(1)<br>NOT NULL WITH<br>DEFAULT   | The index type:<br><br>2 Type 2 index<br>blank Type 1 index                                                                                                                                                                                                                                                                                                                   | G   |
| FIRSTKEYCARDF | FLOAT<br>NOT NULL WITH<br>DEFAULT -1  | Number of distinct values of the first key column. This number is an estimate if updated while collecting statistics on a single partition. The value is -1 if statistics have not been gathered. This is an updatable column.                                                                                                                                                | S   |
| FULLKEYCARDF  | FLOAT<br>NOT NULL WITH<br>DEFAULT -1  | Number of distinct values of the key. The value is -1 if statistics have not been gathered. This is an updatable column.                                                                                                                                                                                                                                                      | S   |
| CREATEDTS     | TIMESTAMP<br>NOT NULL WITH<br>DEFAULT | Time when the CREATE statement was executed for the index. If the index was created in a DB2 release prior to Version 5, the value is '0001-01-01-00.00.00.000000'.                                                                                                                                                                                                           | G   |
| ALTEREDTS     | TIMESTAMP<br>NOT NULL WITH<br>DEFAULT | Time when the most recent ALTER INDEX statement was executed for the index. If no ALTER INDEX statement has been applied, ALTEREDTS has the value of CREATEDTS. If the index was created in a DB2 release prior to Version 5, the value is '0001-01-01-00.00.00.000000'.                                                                                                      | G   |

## SYSIBM.SYSINDEXES

| Column name   | Data type                                                              | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            | Use |
|---------------|------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| PIECESIZE     | INTEGER<br>NOT NULL<br>WITH DEFAULT                                    | Maximum size of a data set in kilobytes for nonpartitioning indexes.                                                                                                                                                                                                                                                                                                                                                                                                                                   | G   |
| COPY          | CHAR(1)<br>NOT NULL WITH<br>DEFAULT 'N'                                | A value of zero (0) indicates that the index is a partitioning index or that the index was created in a DB2 release prior to Version 5.<br><br>Whether COPY YES was specified for the index, which indicates if the index can be copied and if SYSIBM.SYSLGRNX recording is enabled for the index.<br><br>N No<br>Y Yes                                                                                                                                                                                | G   |
| COPYLRSN      | CHAR(6)<br>NOT NULL WITH<br>DEFAULT<br>X'000000000000'<br>FOR BIT DATA | The value can be either an RBA or LRSN. (LRSN is only for data sharing.) If the index is currently defined as COPY YES, the value is the RBA or LRSN when the index was created with COPY YES or altered to COPY YES, not the current RBA or LRSN. If the index is currently defined as COPY NO, the value is set to X'000000000000' if the index was created with COPY NO; otherwise, if the index was altered to COPY NO, the value in COPYLRSN is not changed when the index is altered to COPY NO. | G   |
| CLUSTERRATIOF | FLOAT<br>NOT NULL WITH<br>DEFAULT                                      | When multiplied by 100, the value of the column is the percentage of rows that are in clustering order. For example, a value of .9125 indicates 91.25%. For a partitioning index, it is the weighted average of all index partitions in terms of the number of rows in the partition. The value is 0 if statistics have not been gathered. The value is -2 if the index is for an auxiliary table. This is an updatable column.                                                                        | G   |
| SPACEF        | FLOAT(8)<br>NOT NULL WITH<br>DEFAULT -1                                | Kilobytes of DASD storage. The value is -1 if statistics have not been gathered. This is an updatable column.                                                                                                                                                                                                                                                                                                                                                                                          | G   |
| REMARKS       | VARCHAR(254)<br>NOT NULL WITH<br>DEFAULT                               | A character field string provided by the user with the COMMENT ON statement..                                                                                                                                                                                                                                                                                                                                                                                                                          | G   |

---

## SYSIBM.SYSINDEXES\_HIST table

```
Contains rows from SYSINDEXES. Rows are added or changed in this table when
RUNSTATS collects history statistics. Rows in this table can also be inserted,
| updated, and deleted.
```

| Column name   | Data type                               | Description                                                                                                                                                                                                                                                                                 | Use |
|---------------|-----------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| NAME          | VARCHAR(18)<br>NOT NULL                 | Name of the index.                                                                                                                                                                                                                                                                          | G   |
| CREATOR       | CHAR(8)<br>NOT NULL                     | Authorization ID of the owner of the index.                                                                                                                                                                                                                                                 | G   |
| TBNAME        | VARCHAR(18)<br>NOT NULL                 | Name of the table on which the index is defined.                                                                                                                                                                                                                                            | G   |
| TBCREATOR     | VARCHAR(18)<br>NOT NULL                 | Authorization ID of the owner of the table.                                                                                                                                                                                                                                                 | G   |
| CLUSTERING    | CHAR(1)<br>NOT NULL                     | Whether CLUSTER was specified when the index was created:<br><br>N No<br>Y Yes                                                                                                                                                                                                              | G   |
| NLEAF         | INTEGER<br>NOT NULL WITH<br>DEFAULT -1  | Number of active leaf pages in the index. The value is -1 if statistics have not been gathered.                                                                                                                                                                                             | S   |
| NLEVELS       | SMALLINT<br>NOT NULL WITH<br>DEFAULT -1 | Number of levels in the index tree. If the index is partitioned, it is the maximum of the number of levels in the index tree for all the partitions. The value is -1 if statistics have not been gathered.                                                                                  | S   |
| STATSTIME     | TIMESTAMP<br>NOT NULL                   | If RUNSTATS updated the statistics, the date and time when the last invocation of RUNSTATS updated the statistics. The default value is '0001-01-01-00.00.00.000000'.                                                                                                                       | G   |
| FIRSTKEYCARDF | FLOAT(8)<br>NOT NULL WITH<br>DEFAULT -1 | Number of distinct values of the first key column. This number is an estimate if updated while collecting statistics on a single partition. The value is -1 if statistics have not been gathered.                                                                                           | S   |
| FULLKEYCARDF  | FLOAT(8)<br>NOT NULL WITH<br>DEFAULT -1 | Number of distinct values of the key. The value is -1 if statistics have not been gathered.                                                                                                                                                                                                 | S   |
| CLUSTERRATIOF | FLOAT(8)<br>NOT NULL                    | Percentage of rows that are in clustering order. For a partitioning index, it is the weighted average of all index partitions in terms of the number of rows in the partition. The value is 0 if statistics have not been gathered. The value is -2 if the index is for an auxiliary table. | G   |
| SPACEF        | FLOAT(8)<br>NOT NULL WITH<br>DEFAULT -1 | Number of kilobytes of DASD storage allocated to the index space partition. The value is -1 if statistics have not been gathered.                                                                                                                                                           | G   |
| IBMREQD       | CHAR(1)<br>NOT NULL WITH<br>DEFAULT 'N' | A value of Y indicates that the row came from the basic machine-readable material (MRM) tape. For all other values, see "Release dependency indicators" on page 1005.                                                                                                                       | G   |

## SYSIBM.SYSINDEXPART

### SYSIBM.SYSINDEXPART table

Contains one row for each nonpartitioning index and one row for each partition of a partitioning index.

| Column name | Data type                                | Description                                                                                                                                                                                                                                   | Use |
|-------------|------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| PARTITION   | SMALLINT<br>NOT NULL                     | Partition number; Zero if index is not partitioned.                                                                                                                                                                                           | G   |
| IXNAME      | VARCHAR(18)<br>NOT NULL                  | Name of the index.                                                                                                                                                                                                                            | G   |
| IXCREATOR   | CHAR(8)<br>NOT NULL                      | Authorization ID of the owner of the index.                                                                                                                                                                                                   | G   |
| PQTY        | INTEGER<br>NOT NULL                      | Primary space allocation in units of 4KB storage blocks. For user-managed data sets, the value is set to the primary space allocation only if RUNSTATS INDEX with UPDATE(ALL) or UPDATE(SPACE) is executed; otherwise, the value is zero.     | G   |
| SQTY        | SMALLINT<br>NOT NULL                     | Secondary space allocation in units of 4KB storage blocks. For user-managed data sets, the value is set to the secondary space allocation only if RUNSTATS INDEX with UPDATE(ALL) or UPDATE(SPACE) is executed; otherwise, the value is zero. | G   |
|             |                                          | If the value does not fit into the column, the value of the column is 32767. See the description of column SECQTYI.                                                                                                                           |     |
| STORTYPE    | CHAR(1)<br>NOT NULL                      | Type of storage allocation:<br><b>E</b> Explicit, and STORNAME names an integrated catalog facility catalog<br><b>I</b> Implicit, and STORNAME names a storage group                                                                          | G   |
| STORNAME    | CHAR(8)<br>NOT NULL                      | Name of storage group or integrated catalog facility catalog used for space allocation.                                                                                                                                                       | G   |
| VCATNAME    | CHAR(8)<br>NOT NULL                      | Name of integrated catalog facility catalog used for space allocation.                                                                                                                                                                        | G   |
|             | INTEGER<br>NOT NULL                      | Not used                                                                                                                                                                                                                                      | N   |
|             | INTEGER<br>NOT NULL                      | Not used                                                                                                                                                                                                                                      | N   |
| LEAFDIST    | INTEGER<br>NOT NULL                      | 100 times the average number of leaf pages between successive active leaf pages of the index. The value is -1 if statistics have not been gathered.                                                                                           | S   |
|             | INTEGER<br>NOT NULL                      | Not used                                                                                                                                                                                                                                      | S   |
| IBMREQD     | CHAR(1)<br>NOT NULL                      | A value of Y indicates that the row came from the basic machine-readable material (MRM) tape. For all other values, see "Release dependency indicators" on page 1005.                                                                         | G   |
| LIMITKEY    | VARCHAR(512)<br>NOT NULL<br>FOR BIT DATA | The high value of the limit key of the partition in an internal format. Zero if the index is not partitioned.<br><br>If any column of the key has a field procedure, the internal format is the encoded form of the value.                    | S   |
| FREEPAGE    | SMALLINT<br>NOT NULL                     | Number of pages that are loaded before a page is left as free space.                                                                                                                                                                          | G   |
| PCTFREE     | SMALLINT<br>NOT NULL                     | Percentage of each leaf or nonleaf page that is left as free space.                                                                                                                                                                           | G   |

| Column name        | Data type                               | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          | Use |
|--------------------|-----------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| SPACE              | INTEGER<br>NOT NULL WITH<br>DEFAULT     | Number of kilobytes of DASD storage allocated to the index space partition, as determined by the last execution of the STOSPACE utility. The value is 0 if STOSPACE or RUNSTATS has not been run. The value is updated by STOSPACE if the index is related to a storage group. The value is updated by RUNSTATS if the utility is executed as RUNSTATS INDEX with UPDATE(ALL) or UPDATE(SPACE). The value is -1 if the index was defined with the DEFINE NO clause, which defers the physical creation of the data sets until data is first inserted into the index, and data has yet to be inserted into the index. | G   |
| STATSTIME          | TIMESTAMP<br>NOT NULL WITH<br>DEFAULT   | If RUNSTATS updated the statistics, the date and time when the last invocation of RUNSTATS updated the statistics. The default value is '0001-01-01-00.00.00.000000'.                                                                                                                                                                                                                                                                                                                                                                                                                                                | G   |
|                    | CHAR(1)<br>NOT NULL                     | Not used                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             | N   |
| GBPCACHE           | CHAR(1)<br>NOT NULL WITH<br>DEFAULT     | Group buffer pool cache option specified for this index or index partition.<br><b>blank</b> Only changed pages are cached in the group buffer pool.<br><b>A</b> Changed and unchanged pages are cached in the group buffer pool.<br><b>N</b> No data is cached in the group buffer pool.                                                                                                                                                                                                                                                                                                                             | G   |
| FAROFFPOSF         | FLOAT<br>NOT NULL WITH<br>DEFAULT -1    | Number of referred to rows far from optimal position because of an insert into a full page. The value is -1 if statistics have not been gathered. The column is not applicable for an index on an auxiliary table.                                                                                                                                                                                                                                                                                                                                                                                                   | S   |
| NEAROFFPOSF        | FLOAT<br>NOT NULL WITH<br>DEFAULT -1    | Number of referred to rows near, but not at optimal position, because of an insert into a full page. Not applicable for an index on an auxiliary table.                                                                                                                                                                                                                                                                                                                                                                                                                                                              | S   |
| CARDF              | FLOAT<br>NOT NULL WITH<br>DEFAULT -1    | Number of keys in the index that refer to data rows or LOBs. The value is -1 if statistics have not been gathered.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   | S   |
| SECQTYI            | INTEGER<br>NOT NULL WITH<br>DEFAULT     | Secondary space allocation in units of 4KB storage. If a storage group is not used, the value is zero.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               | G   |
| IPREFIX            | CHAR(1)<br>NOT NULL WITH<br>DEFAULT 'I' | The first character of the instance qualifier for this index's data set name. 'I' or 'J' are the only valid characters for this field. The default is 'I'.                                                                                                                                                                                                                                                                                                                                                                                                                                                           | G   |
| ALTEREDTS          | TIMESTAMP<br>NOT NULL WITH<br>DEFAULT   | Time when the most recent ALTER INDEX statement was executed for the index. If no ALTER INDEX statement has been applied, the value is '0001-01-01-00.00.00.000000'.                                                                                                                                                                                                                                                                                                                                                                                                                                                 | G   |
| SPACEF             | FLOAT(8)<br>NOT NULL WITH<br>DEFAULT -1 | Kilobytes of DASD storage. The value is -1 if statistics have not been gathered. This is an updatable column.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        | G   |
| DSNUM              | INTEGER<br>NOT NULL WITH<br>DEFAULT -1  | Number of data sets. The value is -1 if statistics have not been gathered. This is an updatable column.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              | G   |
| # EXTENTS          | INTEGER<br>NOT NULL WITH<br>DEFAULT -1  | Number of data set extents. The value is -1 if statistics have not been gathered. This is an updatable column. This value is only for the last DSNUM for the object.                                                                                                                                                                                                                                                                                                                                                                                                                                                 | G   |
| PSEUDO_DEL_ENTRIES | INTEGER<br>NOT NULL WITH<br>DEFAULT -1  | Number of psuedo deleted entries (entries that are logically deleted but still physically present in the index). For a non-unique index, value is the number of RIDs that are pseudo deleted. For a unique index, the value is the number of keys and RIDs that are pseudo deleted. The value is -1 if statistics have not been gathered. This is an updatable column.                                                                                                                                                                                                                                               | G   |
| LEAFNEAR           | INTEGER<br>NOT NULL WITH<br>DEFAULT -1  | Number of leaf pages physically near previous leaf page for successive active leaf pages. The value is -1 if statistics have not been gathered. This is an updatable column.                                                                                                                                                                                                                                                                                                                                                                                                                                         | S   |

## SYSIBM.SYSINDEXPART

| Column name | Data type                              | Description                                                                                                                                                                                                                | Use |
|-------------|----------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| LEAFFAR     | INTEGER<br>NOT NULL WITH<br>DEFAULT -1 | Number of leaf pages located physically far away from previous leaf pages for successive (active leaf) pages accessed in an index scan. The value is -1 if statistics have not been gathered. This is an updatable column. | S   |

---

## SYSIBM.SYSINDEXPART\_HIST table

# Contains rows from SYSINDEXPART. Rows are added or changed in this table  
# when RUNSTATS collects history statistics.. Rows in this table can be inserted,  
# updated, and deleted.

| Column name        | Data type                               | Description                                                                                                                                                                                                                                                   | Use |
|--------------------|-----------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| PARTITION          | SMALLINT<br>NOT NULL                    | Partition number. Zero if index is not partitioned.                                                                                                                                                                                                           | G   |
| IXNAME             | VARCHAR(18)<br>NOT NULL                 | Name of the index.                                                                                                                                                                                                                                            | G   |
| IXCREATOR          | CHAR(8)<br>NOT NULL                     | Authorization ID of the owner of the index.                                                                                                                                                                                                                   | G   |
| PQTY               | INTEGER<br>NOT NULL                     | Primary space allocation in units of 4KB storage blocks. Zero if a storage group is not used.                                                                                                                                                                 | G   |
| SECQTYI            | INTEGER<br>NOT NULL                     | Secondary space allocation in units of 4KB storage. If a storage group is not used, the value is 0.                                                                                                                                                           | G   |
| LEAFDIST           | INTEGER<br>NOT NULL WITH<br>DEFAULT -1  | 100 times the average number of leaf pages between successive active leaf pages of the index. The value is -1 if statistics have not been gathered.                                                                                                           | S   |
| SPACEF             | FLOAT(8)<br>NOT NULL WITH<br>DEFAULT -1 | Number of kilobytes of DASD storage allocated to the index space partition. The value is -1 if statistics have not been gathered.                                                                                                                             | G   |
| STATSTIME          | TIMESTAMP<br>NOT NULL                   | If RUNSTATS updated the statistics, the date and time when the last invocation of RUNSTATS updated the statistics. The default value is '0001-01-01-00.00.00.000000'.                                                                                         | G   |
| FAROFFPOSF         | FLOAT(8)<br>NOT NULL WITH<br>DEFAULT -1 | Number of referred to rows far from optimal position because of an insert into a full page. The value is -1 if statistics have not been gathered. The column is not applicable for an index on an auxiliary table.                                            | S   |
| NEAROFFPOSF        | FLOAT(8)<br>NOT NULL WITH<br>DEFAULT -1 | Number of referred to rows near, but not at optimal position, because of an insert into a full page. Not applicable for an index on an auxiliary table. The value is -1 if statistics have not been gathered.                                                 | S   |
| CARDF              | FLOAT(8)<br>NOT NULL WITH<br>DEFAULT -1 | Number of keys in the index that refer to data rows or LOBs. The value is -1 if statistics have not been gathered.                                                                                                                                            | S   |
| # EXTENTS          | INTEGER<br>NOT NULL WITH<br>DEFAULT -1  | Number of data set extents. The value is -1 if statistics have not been gathered. This value is only for the last DSNUM for the object.                                                                                                                       | G   |
| PSEUDO_DEL_ENTRIES | INTEGER<br>NOT NULL WITH<br>DEFAULT -1  | Number of psuedo deleted entries. The value is -1 if statistics have not been gathered.                                                                                                                                                                       | G   |
| DSNUM              | INTEGER<br>NOT NULL WITH<br>DEFAULT -1  | Data set number within the table space. For partitioned index spaces, this value corresponds to the partition number for a single partition copy, or 0 for a copy of an entire partitioned index space. The value is -1 if statistics have not been gathered. | G   |
| IBMREQD            | CHAR(1)<br>NOT NULL WITH<br>DEFAULT 'N' | A value of Y indicates that the row came from the basic machine-readable material (MRM) tape. For all other values, see "Release dependency indicators" on page 1005.                                                                                         | G   |
| LEAFNEAR           | INTEGER<br>NOT NULL WITH<br>DEFAULT -1  | Number of leaf pages physically near previous leaf page for successive active leaf pages. The value is -1 if statistics have not been gathered. This is an updatable column.                                                                                  | S   |
| LEAFFAR            | INTEGER<br>NOT NULL WITH<br>DEFAULT -1  | Number of leaf pages located physically far away from previous leaf pages for successive (active leaf) pages accessed in an index scan. The value is -1 if statistics have not been gathered. This is an updatable column.                                    | S   |

---

## SYSIBM.SYSINDEXSTATS

### SYSIBM.SYSINDEXSTATS table

Contains one row for each partition of a partitioning index. Rows in this table can be inserted, updated, and deleted.

| Column name   | Data type                            | Description                                                                                                                                                                                                              | Use |
|---------------|--------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| FIRSTKEYCARD  | INTEGER<br>NOT NULL                  | For the index partition, number of distinct values of the first key column.                                                                                                                                              | S   |
| FULLKEYCARD   | INTEGER<br>NOT NULL                  | For the index partition, number of distinct values of the key.                                                                                                                                                           | S   |
| NLEAF         | INTEGER<br>NOT NULL                  | Number of active leaf pages in the index partition.                                                                                                                                                                      | S   |
| NLEVELS       | SMALLINT<br>NOT NULL                 | Number of levels in the partition index tree.                                                                                                                                                                            | S   |
|               | SMALLINT<br>NOT NULL                 | Not used                                                                                                                                                                                                                 | N   |
|               | SMALLINT<br>NOT NULL                 | Not used                                                                                                                                                                                                                 | N   |
| CLUSTERRATIO  | SMALLINT<br>NOT NULL                 | For the index partition, the percentage of rows that are in clustering order. The value is 0 if statistics have not been gathered.                                                                                       | N   |
| STATSTIME     | TIMESTAMP<br>NOT NULL                | If RUNSTATS updated the statistics, the date and time when the last invocation of RUNSTATS updated the statistics. The default value is '0001-01-01-00.00.00.000000'.                                                    | G   |
| IBMREQD       | CHAR(1)<br>NOT NULL                  | A value of Y indicates that the row came from the basic machine-readable material (MRM) tape. For all other values, see "Release dependency indicators" on page 1005.                                                    | G   |
| PARTITION     | SMALLINT<br>NOT NULL                 | Partition number of the index.                                                                                                                                                                                           | G   |
| OWNER         | CHAR(8)<br>NOT NULL                  | Authorization ID of the owner of the index.                                                                                                                                                                              | G   |
| NAME          | VARCHAR(18)<br>NOT NULL              | Name of the index.                                                                                                                                                                                                       | G   |
| KEYCOUNT      | INTEGER<br>NOT NULL                  | Total number of rows in the partition.                                                                                                                                                                                   | S   |
| FIRSTKEYCARDF | FLOAT<br>NOT NULL WITH<br>DEFAULT -1 | For the index partition, number of distinct values of the first key column.                                                                                                                                              | S   |
| FULLKEYCARDF  | FLOAT<br>NOT NULL WITH<br>DEFAULT -1 | For the index partition, number of distinct values of the key.                                                                                                                                                           | S   |
| KEYCOUNTF     | FLOAT<br>WITH<br>DEFAULT -1          | Total number of rows in the partition.                                                                                                                                                                                   | S   |
| CLUSTERRATIOF | FLOAT<br>NOT NULL WITH<br>DEFAULT    | For the index partition, the value, when multiplied by 100, is the percentage of rows that are in clustering order. For example, a value of .9125 indicates 91.25%. The value is 0 if statistics have not been gathered. | G   |

---

## SYSIBM.SYSINDEXSTATS\_HIST table

```
Contains rows from SYSINDEXSTATS. Rows are added or changed in this table
when RUNSTATS collects history statistics. Rows in this table can also be inserted,
| updated, and deleted.
```

| Column name   | Data type                               | Description                                                                                                                                                                                                        | Use |
|---------------|-----------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| NLEAF         | INTEGER<br>NOT NULL WITH<br>DEFAULT -1  | Number of active leaf pages in the index partition. The value is -1 if statistics have not been gathered.                                                                                                          | S   |
| NLEVELS       | SMALLINT<br>NOT NULL WITH<br>DEFAULT -1 | Number of levels in the partition index tree. The value is -1 if statistics have not been gathered.                                                                                                                | S   |
| STATSTIME     | TIMESTAMP<br>NOT NULL                   | If RUNSTATS updated the statistics, the date and time when the last invocation of RUNSTATS updated the statistics. The default value is '0001-01-01-00.00.00.000000'.                                              | G   |
| PARTITION     | SMALLINT<br>NOT NULL                    | Partition number of the index.                                                                                                                                                                                     | G   |
| OWNER         | CHAR(8)<br>NOT NULL                     | Authorization ID of the owner of the index.                                                                                                                                                                        | G   |
| NAME          | VARCHAR(18)<br>NOT NULL                 | Name of the index.                                                                                                                                                                                                 | G   |
| FIRSTKEYCARDF | FLOAT(8)<br>NOT NULL WITH<br>DEFAULT -1 | For the index partition, number of distinct values of the first key column. The value is -1 if statistics have not been gathered.                                                                                  | S   |
| FULLKEYCARDF  | FLOAT(8)<br>NOT NULL WITH<br>DEFAULT -1 | For the index partition, number of distinct values of the key. The value is -1 if statistics have not been gathered.                                                                                               | S   |
| KEYCOUNTF     | FLOAT(8)<br>NOT NULL WITH<br>DEFAULT -1 | Total number of rows in the partition. The value is -1 if statistics have not been gathered.                                                                                                                       | S   |
| CLUSTERRATIOF | FLOAT(8)<br>NOT NULL                    | For the index partition, the value, when multiplied by 100, is the percentage of rows that are in clustering order. For example, a value of indicates 91.25%. The value is 0 if statistics have not been gathered. | G   |
| IBMREQD       | CHAR(1)<br>NOT NULL WITH<br>DEFAULT 'N' | A value of Y indicates that the row came from the basic machine-readable material (MRM) tape. For all other values, see "Release dependency indicators" on page 1005.                                              | G   |

---

## SYSIBM.SYSJARCLASS\_SOURCE

---

### SYSIBM.SYSJARCLASS\_SOURCE table

Auxiliary table for SYSIBM.SYSJARCONTENTS.

| Column name  | Data type             | Description                                | Use |
|--------------|-----------------------|--------------------------------------------|-----|
| CLASS_SOURCE | CLOB(10M)<br>NOT NULL | The contents of the class in the JAR file. | G   |

---

## SYSIBM.SYSJARCONTENTS table

Contains Java class source for installed JAR.

| Column name        | Data type                                | Description                                                                                                                                                           | Use |
|--------------------|------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| JARSCHEMA          | CHAR(8)<br>NOT NULL                      | The schema of the JAR file.                                                                                                                                           | G   |
| JAR_ID             | CHAR(18)<br>NOT NULL                     | The name of the JAR file.                                                                                                                                             | G   |
| CLASS              | VARCHAR(128)<br>NOT NULL                 | The class name contained in the JAR file.                                                                                                                             | G   |
| CLASS_SOURCE_ROWID | ROWID<br>NOT NULL<br>GENERATED<br>ALWAYS | ID used to support CLOB datatype.                                                                                                                                     | G   |
| CLASS_SOURCE       | CLOB(10M)<br>NOT NULL                    | The contents of the class in the JAR file.                                                                                                                            | G   |
| IBMRREQD           | CHAR(1)<br>NOT NULL WITH<br>DEFAULT 'N'  | A value of Y indicates that the row came from the basic machine-readable material (MRM) tape. For all other values, see "Release dependency indicators" on page 1005. | G   |

---

## SYSIBM.SYSJARDATA

---

### SYSIBM.SYSJARDATA table

Auxiliary table for SYSIBM.SYSJAROBJECTS.

| Column name | Data type              | Description                   | Use |
|-------------|------------------------|-------------------------------|-----|
| JAR_DATA    | BLOB(100M)<br>NOT NULL | The contents of the JAR file. | G   |

---

## SYSIBM.SYSJAROBJECTS table

Contains binary large object representing the installed JAR.

| Column name    | Data type                                | Description                                                                                                                                                           | Use |
|----------------|------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| JARSCHEMA      | CHAR(8)<br>NOT NULL                      | The schema of the JAR file.                                                                                                                                           | G   |
| JAR_ID         | CHAR(18)<br>NOT NULL                     | The name of the JAR file.                                                                                                                                             | G   |
| OWNER          | CHAR(8)<br>NOT NULL                      | Authorization ID of the owner of the JAR object.                                                                                                                      | G   |
| JAR_DATA_ROWID | ROWID<br>NOT NULL<br>GENERATED<br>ALWAYS | ID used to support BLOB datatype.                                                                                                                                     | G   |
| JAR_DATA       | BLOB(100M)<br>NOT NULL                   | The contents of the JAR file. This is an updatable column.                                                                                                            | G   |
| # PATH         | VARCHAR(1024)<br>NOT NULL                | The URL path of the source JAR file.                                                                                                                                  | G   |
| CREATEDTS      | TIMESTAMP<br>NOT NULL                    | Time when the JAR object was created.                                                                                                                                 | G   |
| ALTEREDTS      | TIMESTAMP<br>NOT NULL                    | Time when the JAR object was altered.                                                                                                                                 | G   |
| IBMREQD        | CHAR(1)<br>NOT NULL WITH<br>DEFAULT 'N'  | A value of Y indicates that the row came from the basic machine-readable material (MRM) tape. For all other values, see "Release dependency indicators" on page 1005. | G   |

---

## SYSIBM.SYSJAVAOPTS

### SYSIBM.SYSJAVAOPTS table

Contains build options used during INSTALL\_JAR.

| Column name     | Data type                               | Description                                                                                                                                                           | Use |
|-----------------|-----------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| JARSCHEMA       | CHAR(8)<br>NOT NULL                     | The schema of the JAR file.                                                                                                                                           | G   |
| JAR_ID          | CHAR(18)<br>NOT NULL                    | The name of the JAR file.                                                                                                                                             | G   |
| BUILDSchema     | CHAR(8)<br>NOT NULL                     | Schema name for BUILDNAME.                                                                                                                                            | G   |
| BUILDNAME       | CHAR(18)<br>NOT NULL                    | Procedure used to create the routine.                                                                                                                                 | G   |
| BUILDOwner      | CHAR(8)<br>NOT NULL                     | Authorization ID used to create the routine.                                                                                                                          | G   |
| DBMLIB          | VARCHAR(128)<br>NOT NULL                | PDS name where DBRM is located.                                                                                                                                       | G   |
| HPJCOMPILE_OPTS | VARCHAR(256)<br>NOT NULL                | HPJ compile options used to install the routine.                                                                                                                      | G   |
| BIND_OPTS       | VARCHAR(1024)<br>NOT NULL               | Bind options used to install the routine.                                                                                                                             | G   |
| POBJECT_LIB     | VARCHAR(128)<br>NOT NULL                | PDSE name where program object is located.                                                                                                                            | G   |
| IBMREQD         | CHAR(1)<br>NOT NULL WITH<br>DEFAULT 'N' | A value of Y indicates that the row came from the basic machine-readable material (MRM) tape. For all other values, see "Release dependency indicators" on page 1005. | G   |

---

## SYSIBM.SYSKEYCOLUSE table

Contains a row for every column in a unique constraint (primary key or unique key) from the SYSIBM.SYSTABCONST table.

| Column name | Data type                               | Description                                                                                                                                                           | Use |
|-------------|-----------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| CONSTNAME   | VARCHAR(128)<br>NOT NULL                | Name of the constraint.                                                                                                                                               | G   |
| TBCREATOR   | CHAR(8)<br>NOT NULL                     | Authorization ID of the owner of the table on which the constraint is defined.                                                                                        | G   |
| TBNAME      | VARCHAR(18)<br>NOT NULL                 | Name of the table on which the constraint is defined.                                                                                                                 | G   |
| COLNAME     | VARCHAR(18)<br>NOT NULL                 | Name of the column                                                                                                                                                    | G   |
| COLSEQ      | SMALLINT<br>NOT NULL                    | Numeric position of the column in the key (the first position in the key is 1).                                                                                       | G   |
| COLNO       | SMALLINT<br>NOT NULL                    | Numeric position of the column in the table on which the constraint is defined.                                                                                       | G   |
| IBMREQD     | CHAR(1)<br>NOT NULL WITH<br>DEFAULT 'N' | A value of Y indicates that the row came from the basic machine-readable material (MRM) tape. For all other values, see "Release dependency indicators" on page 1005. | G   |

---

## SYSIBM.SYSKEYS

### SYSIBM.SYSKEYS table

Contains one row for each column of an index key.

| Column name | Data type               | Description                                                                                                                                                           | Use |
|-------------|-------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| IXNAME      | VARCHAR(18)<br>NOT NULL | Name of the index.                                                                                                                                                    | G   |
| IXCREATOR   | CHAR(8)<br>NOT NULL     | Authorization ID of the owner of the index.                                                                                                                           | G   |
| COLNAME     | VARCHAR(18)<br>NOT NULL | Name of the column of the key.                                                                                                                                        | G   |
| COLNO       | SMALLINT<br>NOT NULL    | Numeric position of the column in the table; for example, 4 (out of 10).                                                                                              | G   |
| COLSEQ      | SMALLINT<br>NOT NULL    | Numeric position of the column in the key; for example, 4 (out of 4).                                                                                                 | G   |
| ORDERING    | CHAR(1)<br>NOT NULL     | Order of the column in the key:<br><b>A</b> Ascending<br><b>D</b> Descending                                                                                          | G   |
| IBMREQD     | CHAR(1)<br>NOT NULL     | A value of Y indicates that the row came from the basic machine-readable material (MRM) tape. For all other values, see “Release dependency indicators” on page 1005. | G   |

---

## SYSIBM.SYSLOBSTATS table

Contains one row for each LOB table space.

| Column name | Data type                | Description                                                                                                                                                                                                                                                                                                                            | Use |
|-------------|--------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| STATSTIME   | TIMESTAMP<br>NOT NULL    | Timestamp of RUNSTATS statistics update.                                                                                                                                                                                                                                                                                               | G   |
| AVGSIZE     | INTEGER<br>NOT NULL      | Average size of a LOB, measured in bytes, in the LOB table space.                                                                                                                                                                                                                                                                      | S   |
| FREESPACE   | INTEGER<br>NOT NULL      | Number of kilobytes of available space in the LOB table space.                                                                                                                                                                                                                                                                         | S   |
| ORGRATIO    | DECIMAL(5,2)<br>NOT NULL | The percentage of organization in the LOB table space. A value of 100 indicates perfect organization of the LOB table space. A value of 1 indicates that the LOB table space is disorganized.<br><br>A value of 0.00 indicates that the LOB table space is totally disorganized. An empty LOB table space will have a value of 100.00. | S   |
| DBNAME      | CHAR(8)<br>NOT NULL      | Name of the database that contains the LOB table space named in NAME.                                                                                                                                                                                                                                                                  | G   |
| NAME        | CHAR(8)<br>NOT NULL      | Name of the LOB table space.                                                                                                                                                                                                                                                                                                           | G   |
| IBMREQD     | CHAR(1)<br>NOT NULL      | A value of Y indicates that the row came from the basic machine-readable material (MRM) tape. For all other values, see "Release dependency indicators" on page 1005.                                                                                                                                                                  | G   |

## SYSIBM.SYSLOBSTATS\_HIST

### SYSIBM.SYSLOBSTATS\_HIST table

# Contains rows from SYSLOBSTATS. Rows are added or changed in this table when  
# RUNSTATS collects history statistics. Rows in this table can also be inserted,  
| updated, and deleted.

| Column name | Data type                               | Description                                                                                                                                                                                                                                                                                                                           | Use |
|-------------|-----------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| STATSTIME   | TIMESTAMP<br>NOT NULL                   | Timestamp of RUNSTATS statistics update.                                                                                                                                                                                                                                                                                              | G   |
| FREESPACE   | INTEGER<br>NOT NULL                     | Number of pages of free space in the LOB table space.                                                                                                                                                                                                                                                                                 | S   |
| ORGRATIO    | DECIMAL(5,2)<br>NOT NULL                | The percentage of organization in the LOB table space. A value of 100 indicates perfect organization of the LOB table space. A value of 1 indicates that the LOB table space is disorganized.<br><br>A value of 0.00 indicates that the LOB table space is totally disorganized. An empty LOB table space will have a value of 100.00 | S   |
| DBNAME      | CHAR(8)<br>NOT NULL                     | Name of the database that contains the LOB table space named in NAME.                                                                                                                                                                                                                                                                 | G   |
| NAME        | CHAR(8)<br>NOT NULL                     | Name of the LOB table space.                                                                                                                                                                                                                                                                                                          | G   |
| IBMREQD     | CHAR(1)<br>NOT NULL WITH<br>DEFAULT 'N' | A value of Y indicates that the row came from the basic machine-readable material (MRM) tape. For all other values, see "Release dependency indicators" on page 1005.                                                                                                                                                                 | G   |

## SYSIBM.SYSPACKAGE table

Contains a row for every package.

| Column name | Data type             | Description                                                                                                                                                                                                                                                                                                                                                                                                                          | Use |
|-------------|-----------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| LOCATION    | CHAR(16)<br>NOT NULL  | Always contains blanks                                                                                                                                                                                                                                                                                                                                                                                                               | S   |
| COLLID      | CHAR(18)<br>NOT NULL  | Name of the package collection. For a trigger package, it is the schema name of the trigger.                                                                                                                                                                                                                                                                                                                                         | G   |
| NAME        | CHAR(8)<br>NOT NULL   | Name of the package.                                                                                                                                                                                                                                                                                                                                                                                                                 | G   |
| CONTOKEN    | CHAR(8)<br>NOT NULL   | Consistency token for the package. For a package derived from a DB2 DBRM, this is either: <ul style="list-style-type: none"><li>• The “level” as specified by the LEVEL option when the package’s program was precompiled</li><li>• The timestamp indicating when the package’s program was precompiled, in an internal format.</li></ul>                                                                                            | S   |
| OWNER       | CHAR(8)<br>NOT NULL   | Authorization ID of the package owner. For a trigger package, the value is the authorization ID of the owner of the trigger, which is set to the current authorization ID (the plan or package owner for static CREATE TRIGGER statement; the current SQLID for a dynamic CREATE TRIGGER statement).                                                                                                                                 | G   |
| CREATOR     | CHAR(8)<br>NOT NULL   | Authorization ID of the owner of the creator of the package version. For a trigger package, the value is determined differently. For dynamic SQL, it is the primary authorization ID of the user who issued the CREATE TRIGGER statement. For static SQL, it is the authorization ID of the plan or package owner.                                                                                                                   | G   |
| TIMESTAMP   | TIMESTAMP<br>NOT NULL | Timestamp indicating when the package was created.                                                                                                                                                                                                                                                                                                                                                                                   | G   |
| BINDTIME    | TIMESTAMP<br>NOT NULL | Timestamp indicating when the package was last bound.                                                                                                                                                                                                                                                                                                                                                                                | G   |
| QUALIFIER   | CHAR(8)<br>NOT NULL   | Implicit qualifier for the unqualified table, view, index, and alias names in the static SQL statements of the package.                                                                                                                                                                                                                                                                                                              | G   |
| PKSIZE      | INTEGER<br>NOT NULL   | Size of the base section <sup>48</sup> of the package, in bytes.                                                                                                                                                                                                                                                                                                                                                                     | G   |
| AVGSIZE     | INTEGER<br>NOT NULL   | Average size, in bytes, of those sections <sup>48</sup> of the plan that contain SQL statements processed at bind time.                                                                                                                                                                                                                                                                                                              | G   |
| SYSENTRIES  | SMALLINT<br>NOT NULL  | Number of enabled or disabled entries for this package in SYSIBM.SYSPKSYSTEM. A value of 0 if all types of connections are enabled.                                                                                                                                                                                                                                                                                                  | G   |
| VALID       | CHAR(1)<br>NOT NULL   | Whether the package is valid:<br><b>A</b> An ALTER statement changed the description of the table or base table of a view referred to by the package. The changes do not invalidate the package.<br><b>H</b> An ALTER TABLE statement changed the description of the table or base table of a view referred to by the package. For releases of DB2 prior to V5R1, the change invalidates the package.<br><b>N</b> No<br><b>Y</b> Yes | G   |
| OPERATIVE   | CHAR(1)<br>NOT NULL   | Whether the package can be allocated:<br><b>N</b> No; an explicit BIND or REBIND is required before the package can be allocated.<br><b>Y</b> Yes                                                                                                                                                                                                                                                                                    | G   |

48. Packages are divided into *sections*. The base section of the package must be in the EDM pool during the entire time the package is executing. Other sections of the package, corresponding roughly to sets of related SQL statements, are brought into the pool as needed.

## SYSIBM.SYSPACKAGE

| Column name | Data type           | Description                                                                                                                                                                                                                                                                                                                                                                                                                          | Use |
|-------------|---------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| VALIDATE    | CHAR(1)<br>NOT NULL | Whether validity checking can be deferred until run time:<br><b>B</b> All checking must be performed at bind time.<br><b>R</b> Validation is done at run time for tables, views, and privileges that do not exist at bind time.                                                                                                                                                                                                      | G   |
| ISOLATION   | CHAR(1)<br>NOT NULL | Isolation level when the package was last bound or rebound<br><b>R</b> RR (repeatable read)<br><b>S</b> CS (cursor stability)<br><b>T</b> RS (read stability)<br><b>U</b> UR (uncommitted read)<br><b>blank</b> Not specified, and therefore at the level specified for the plan executing the package                                                                                                                               | G   |
| RELEASE     | CHAR(1)<br>NOT NULL | The value used for RELEASE when the package was last bound or rebound:<br><b>C</b> Value used was COMMIT.<br><b>D</b> Value used was DEALLOCATE.<br><b>blank</b> Not specified, and therefore the value specified for the plan executing the package.                                                                                                                                                                                | G   |
| EXPLAIN     | CHAR(1)<br>NOT NULL | EXPLAIN option specified for the package; that is, whether information on the package's statements was added to the owner of the PLAN_TABLE table:<br><b>N</b> No<br><b>Y</b> Yes                                                                                                                                                                                                                                                    | G   |
| QUOTE       | CHAR(1)<br>NOT NULL | SQL string delimiter for SQL statements in the package:<br><b>N</b> Apostrophe<br><b>Y</b> Quotation mark                                                                                                                                                                                                                                                                                                                            | G   |
| COMMA       | CHAR(1)<br>NOT NULL | Decimal point representation for SQL statements in package:<br><b>N</b> Period<br><b>Y</b> Comma                                                                                                                                                                                                                                                                                                                                     | G   |
| HOSTLANG    | CHAR(1)<br>NOT NULL | Host language for the package's DBRM:<br><b>B</b> Assembler language<br><b>C</b> OS/VS COBOL<br><b>D</b> C<br><b>F</b> Fortran<br><b>P</b> PL/I<br><b>2</b> VS COBOL II or IBM COBOL Release 1 (formerly called COBOL/370™)<br><b>3</b> IBM COBOL (Release 2 or subsequent releases)<br><b>4</b> C++<br><b>blank</b> For remotely bound packages, or trigger packages (TYPE='T')                                                     | G   |
| CHARSET     | CHAR(1)<br>NOT NULL | Indicates whether the system CCSID for SBCS data was 290 (Katakana) when the program was precompiled:<br><b>K</b> Yes<br><b>A</b> No                                                                                                                                                                                                                                                                                                 | G   |
| MIXED       | CHAR(1)<br>NOT NULL | Indicates if mixed data was in effect when the package's program was precompiled (for more on when mixed data is in effect, see "Character strings" on page 49):<br><b>N</b> No<br><b>Y</b> Yes                                                                                                                                                                                                                                      | G   |
| # DEC31     | CHAR(1)<br>NOT NULL | Indicates whether DEC31 was in effect when the package's program was precompiled (for more on when DEC31 is in effect, see "Arithmetic with two decimal operands" on page 114). If this option is specified at precompile time in the form D31.s, where s is a scale from 1 to 9, then DEC31 is in effect. If this option is specified at precompile time in the form D15.s, then DEC15 is in effect.<br><b>N</b> No<br><b>Y</b> Yes | G   |

| Column name  | Data type                           | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          | Use |
|--------------|-------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| DEFERPREP    | CHAR(1)<br>NOT NULL                 | Indicates the CURRENTDATA option when the package was bound or rebound:<br><br><b>A</b> Data currency is required for all cursors. Inhibit blocking for all cursors.<br><b>B</b> Data currency is not required for ambiguous cursors.<br><b>C</b> Data currency is required for ambiguous cursors.<br><b>blank</b> The package was created before the CURRENTDATA option was available.                                                                                                                                              | G   |
| SQLERROR     | CHAR(1)<br>NOT NULL                 | Indicates the SQLERROR option on the most recent subcommand that bound or rebound the package:<br><br><b>C</b> CONTINUE<br><b>N</b> NOPACKAGE                                                                                                                                                                                                                                                                                                                                                                                        | G   |
| REMOTE       | CHAR(1)<br>NOT NULL                 | Source of the package:<br><br><b>C</b> Package was created by BIND COPY.<br><b>D</b> Package was created by BIND COPY with the OPTIONS(COMMAND) option.<br><b>K</b> The package was copied from a package that was originally bound on behalf of a remote requester.<br><b>L</b> The package was copied with the OPTIONS(COMMAND) option from a package that was originally bound on behalf of a remote requester.<br><b>N</b> Package was locally bound from a DBRM.<br><b>Y</b> Package was bound on behalf of a remote requester. | G   |
| PCTIMESTAMP  | TIMESTAMP<br>NOT NULL               | Date and time the application program was precompiled, or 0001-01-01-00.00.00000 if the LEVEL precompiler option was used, or if the package came from a non-DB2 location.                                                                                                                                                                                                                                                                                                                                                           | G   |
| IBMREQD      | CHAR(1)<br>NOT NULL                 | A value of Y indicates that the row came from the basic machine-readable material (MRM) tape. For all other values, see "Release dependency indicators" on page 1005.                                                                                                                                                                                                                                                                                                                                                                | G   |
| VERSION      | VARCHAR(64)<br>NOT NULL             | Version identifier for the package. The value is blank for a trigger package (TYPE='T').                                                                                                                                                                                                                                                                                                                                                                                                                                             | G   |
| PDSNAME      | VARCHAR(44)<br>NOT NULL             | For a locally bound package, the name of the PDS (library) in which the package's DBRM is a member. For a locally copied package, the value in SYSPACKAGE.PDSNAME for the source package. Otherwise, the product signature of the bind requester followed by one of the following:<br><ul style="list-style-type: none"> <li>• The requester's location name if the product is DB2</li> <li>• Otherwise, the requester's LU name enclosed in angle brackets; for example, "&lt;LUSQLDS&gt;".</li> </ul>                              | G   |
| DEGREE       | CHAR(3)<br>NOT NULL WITH<br>DEFAULT | The DEGREE option used when the package was last bound:<br><br><b>ANY</b> DEGREE(ANY)<br><b>1 or blank</b> DEGREE(1). Blank if the package was migrated.                                                                                                                                                                                                                                                                                                                                                                             | G   |
| GROUP_MEMBER | CHAR(8)<br>NOT NULL WITH<br>DEFAULT | The DB2 data sharing member name of the DB2 subsystem that performed the most recent bind. This column is blank if the DB2 subsystem was not in a DB2 data sharing environment when the bind was performed.                                                                                                                                                                                                                                                                                                                          | G   |

## SYSIBM.SYSPACKAGE

| Column name  | Data type                               | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      | Use |
|--------------|-----------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| DYNAMICRULES | CHAR(1)<br>NOT NULL WITH<br>DEFAULT     | <p>The DYNAMICRULES option used when the package was last bound:</p> <ul style="list-style-type: none"> <li><b>B</b> BIND. Dynamic SQL statements are executed with DYNAMICRULES bind behavior.</li> <li><b>D</b> DEFINEBIND. When the package is run under an active stored procedure or user-defined function, dynamic SQL statements in the package are executed with DYNAMICRULES define behavior.</li> <li><b>E</b> DEFINERUN. When the package is run under an active stored procedure or user-defined function, dynamic SQL statements in the package are executed with DYNAMICRULES define behavior.</li> <li><b>F</b> When the package is not run under an active stored procedure or user-defined function, dynamic SQL statements in the package are executed with DYNAMICRULES run behavior.</li> <li><b>G</b> When the package is not run under an active stored procedure or user-defined function, dynamic SQL statements in the package are executed with DYNAMICRULES invoke behavior.</li> <li><b>H</b> INVOKEBIND. When the package is run under an active stored procedure or user-defined function, dynamic SQL statements in the package are executed with DYNAMICRULES invoke behavior.</li> <li><b>I</b> INVOKERUN. When the package is run under an active stored procedure or user-defined function, dynamic SQL statements in the package are executed with DYNAMICRULES invoke behavior.</li> <li><b>R</b> RUN. Dynamic SQL statements are executed with DYNAMICRULES run behavior.</li> <li><b>blank</b> DYNAMICRULES is not specified for the package. The package uses the DYNAMICRULES value of the plan to which the package is appended at execution time.<br/>For a description of the DYNAMICRULES behaviors, see “Authorization IDs and dynamic SQL” on page 43.</li> </ul> | G   |
| REOPTVAR     | CHAR(1)<br>NOT NULL WITH<br>DEFAULT 'N' | Whether the access path is determined again at execution time using input variable values:                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       | G   |

| Column name    | Data type                                | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        | Use |
|----------------|------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| DEFERPREPARE   | CHAR(1)<br>NOT NULL WITH<br>DEFAULT      | Whether PREPARE processing is deferred until OPEN is executed:<br><br><b>N</b> Bind option NODEFER(PREPARE) indicates that PREPARE processing is not deferred until OPEN is executed.<br><b>Y</b> Bind option DEFER(PREPARE) indicates that PREPARE processing is deferred until OPEN is executed.<br><b>blank</b> Bind option not specified for the package. It is inherited from the plan.                                                                                                                                                       | G   |
| KEEPDYNAMIC    | CHAR(1)<br>NOT NULL WITH<br>DEFAULT 'N'  | Whether prepared dynamic statements are to be purged at each commit point:<br><br><b>N</b> The bind option is KEEPDYNAMIC(NO). Prepared dynamic SQL statements are destroyed at commit.<br><b>Y</b> The bind option is KEEPDYNAMIC(YES). Prepared dynamic SQL statements are kept past commit.                                                                                                                                                                                                                                                     | G   |
| PATHSCHEMAS    | VARCHAR(254)<br>NOT NULL WITH<br>DEFAULT | SQL path specified on the BIND or REBIND command that bound the package. The path is used to resolve unqualified data type, function, and stored procedure names used in certain contexts. If the PATH bind option was not specified, the value in the column is a zero length string; however, DB2 uses a default SQL path of: SYSIBM, SYSFUN, SYSPROC, <i>package qualifier</i> .                                                                                                                                                                | G   |
| TYPE           | CHAR(1)<br>NOT NULL WITH<br>DEFAULT      | Type of package. Identifies how the package was created:<br><br><b>blank</b> BIND PACKAGE command created the package.<br><b>T</b> CREATE TRIGGER statement created the package, and the package is a trigger package.                                                                                                                                                                                                                                                                                                                             | G   |
| DBPROTOCOL     | CHAR(1)<br>NOT NULL WITH<br>DEFAULT 'P'  | Whether remote access for SQL with three-part names is implemented with DRDA or DB2 private protocol access:<br><br><b>D</b> DRDA<br><b>P</b> DB2 private protocol                                                                                                                                                                                                                                                                                                                                                                                 | G   |
| FUNCTIONTS     | TIMESTAMP<br>NOT NULL WITH<br>DEFAULT    | Timestamp when the function was resolved. Set by the BIND and REBIND commands, but not by AUTOBIND.                                                                                                                                                                                                                                                                                                                                                                                                                                                | G   |
| OPTHINT        | CHAR(8)<br>NOT NULL WITH<br>DEFAULT      | Value of the OPTHINT bind option. Identifies rows in the authid.PLAN_TABLE to be used as input to the optimizer. Contains blanks if no rows in the authid.PLAN_TABLE are to be used as input.                                                                                                                                                                                                                                                                                                                                                      | G   |
| ENCODING_CCSID | INTEGER<br>NOT NULL WITH<br>DEFAULT      | The CCSID corresponding to the encoding scheme or CCSID as specified for the bind option ENCODING. The Encoding Scheme specified on the bind command:<br><br><b>ccsid</b> The specified or derived CCSID.<br><b>0</b> The default CCSID as specified on panel DSNTIPF at installation time. Used when the package was bound prior to Version 7.                                                                                                                                                                                                    | G   |
| IMMEDWRITE     | CHAR(1)<br>NOT NULL WITH<br>DEFAULT      | Indicates when writes of updated group buffer pool dependent pages are to be done. This option is only applicable for data sharing environments.<br><br><b>N</b> Bind option IMMEDWRITE(NO) indicates normal write activity is done.<br><b>Y</b> Bind option IMMEDWRITE(YES) indicates that immediate writes are done for updated group buffer pool dependent pages.<br><b>1</b> Bind option IMMEDWRITE(PH1) indicates that updated group buffer pool dependent pages are written at or before phase 1 commit.<br><b>blank</b> A migrated package. | G   |
| REBOUND        | CHAR(1)<br>NOT NULL WITH<br>DEFAULT      | The release when the package was bound or rebound.<br><br><b>blank</b> Bound prior to Version 7<br><b>K</b> Bound on Version 7                                                                                                                                                                                                                                                                                                                                                                                                                     | G   |

## SYSIBM.SYSPACKAUTH

### SYSIBM.SYSPACKAUTH table

Records the privileges that are held by users over packages.

| Column name | Data type             | Description                                                                                                                                                                                                                                                                                                                                                                            | Use |
|-------------|-----------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| GRANTOR     | CHAR(8)<br>NOT NULL   | Authorization ID of the user who granted the privilege. Could also be PUBLIC or PUBLIC followed by an asterisk <sup>49</sup> .                                                                                                                                                                                                                                                         | G   |
| GRANTEE     | CHAR(8)<br>NOT NULL   | Authorization ID of the user who holds the privileges, the name of a plan that uses the privileges or PUBLIC for a grant to PUBLIC.                                                                                                                                                                                                                                                    | G   |
| LOCATION    | CHAR(16)<br>NOT NULL  | Always contains blanks                                                                                                                                                                                                                                                                                                                                                                 | S   |
| COLLID      | CHAR(18)<br>NOT NULL  | Collection name for the package or packages on which the privilege was granted.                                                                                                                                                                                                                                                                                                        | G   |
| NAME        | CHAR(8)<br>NOT NULL   | Name of the package on which the privileges are held. An asterisk (*) if the privileges are held on all packages in a collection.                                                                                                                                                                                                                                                      | G   |
|             | CHAR(8)<br>NOT NULL   | Not used                                                                                                                                                                                                                                                                                                                                                                               | N   |
| TIMESTAMP   | TIMESTAMP<br>NOT NULL | Timestamp indicating when the privilege was granted.                                                                                                                                                                                                                                                                                                                                   | G   |
| GRANTEETYPE | CHAR(1)<br>NOT NULL   | Type of grantee:<br><b>blank</b> An authorization ID<br><b>P</b> An application plan                                                                                                                                                                                                                                                                                                   | G   |
| AUTHHOWGOT  | CHAR(1)<br>NOT NULL   | Authorization level of the user from whom the privileges were received. This authorization level is not necessarily the highest authorization level of the grantor.<br><b>blank</b> Not applicable<br><b>A</b> PACKADM (on collection *)<br><b>C</b> DBCTL<br><b>D</b> DBADM<br><b>L</b> SYSCTRL<br><b>M</b> DBMAINT<br><b>P</b> PACKADM (on a specific collection)<br><b>S</b> SYSADM | G   |
| BINDAUTH    | CHAR(1)<br>NOT NULL   | Whether GRANTEE can use the BIND and REBIND subcommands against the package:<br><b>blank</b> Privilege is not held<br><b>G</b> Privilege is held with the GRANT option<br><b>Y</b> Privilege is held without the GRANT option                                                                                                                                                          | G   |
| COPYAUTH    | CHAR(1)<br>NOT NULL   | Whether GRANTEE can COPY the package:<br><b>blank</b> Privilege is not held<br><b>G</b> Privilege is held with the GRANT option<br><b>Y</b> Privilege is held without the GRANT option                                                                                                                                                                                                 | G   |
| EXECUTEAUTH | CHAR(1)<br>NOT NULL   | Whether GRANTEE can execute the package:<br><b>blank</b> Privilege is not held<br><b>G</b> Privilege is held with the GRANT option<br><b>Y</b> Privilege is held without the GRANT option                                                                                                                                                                                              | G   |
| IBMREQD     | CHAR(1)<br>NOT NULL   | A value of Y indicates that the row came from the basic machine-readable material (MRM) tape. For all other values, see "Release dependency indicators" on page 1005.                                                                                                                                                                                                                  | G   |

49. PUBLIC followed by an asterisk (PUBLIC\*) denotes PUBLIC AT ALL LOCATIONS. For the conditions where GRANTOR can be PUBLIC or PUBLIC\*, see Part 3 (Volume 1) of *DB2 Administration Guide*.

---

## SYSIBM.SYSPACKDEP table

Records the dependencies of packages on local tables, views, synonyms, table spaces, indexes, aliases, functions, and stored procedures.

| Column name | Data type                                                                 | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      | Use |                 |       |                                        |   |                 |   |       |   |                  |   |                                                                           |   |             |   |         |   |       |   |      |   |
|-------------|---------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|-----------------|-------|----------------------------------------|---|-----------------|---|-------|---|------------------|---|---------------------------------------------------------------------------|---|-------------|---|---------|---|-------|---|------|---|
| BNAME       | VARCHAR(18)<br>NOT NULL                                                   | The name of an object that a package depends on.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 | G   |                 |       |                                        |   |                 |   |       |   |                  |   |                                                                           |   |             |   |         |   |       |   |      |   |
| BQUALIFIER  | CHAR(8)<br>NOT NULL                                                       | The value of the column depends on the type of object:<br><ul style="list-style-type: none"><li>• If BNAME identifies a table space (BTYPE is R), the value is the name of its database.</li><li>• If BNAME identifies user-defined function, a cast function, a stored procedure, or a sequence (BTYPE is F, O, or Q), the value is the schema name.</li><li>• Otherwise, the value is the authorization ID of the owner of BNAME.</li></ul>                                                                                                    | G   |                 |       |                                        |   |                 |   |       |   |                  |   |                                                                           |   |             |   |         |   |       |   |      |   |
| #<br>#<br># |                                                                           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |     |                 |       |                                        |   |                 |   |       |   |                  |   |                                                                           |   |             |   |         |   |       |   |      |   |
| BTYPE       | CHAR(1)<br>NOT NULL                                                       | Type of object identified by BNAME and BQUALIFIER:<br><table><tr><td>A</td><td>Alias</td></tr><tr><td>F</td><td>User-defined function or cast function</td></tr><tr><td>G</td><td>Temporary table</td></tr><tr><td>I</td><td>Index</td></tr><tr><td>O</td><td>Stored procedure</td></tr><tr><td>P</td><td>Partitioned table space if it is defined as LARGE or with the DSSIZE parm</td></tr><tr><td>R</td><td>Table space</td></tr><tr><td>S</td><td>Synonym</td></tr><tr><td>T</td><td>Table</td></tr><tr><td>V</td><td>View</td></tr></table> | A   | Alias           | F     | User-defined function or cast function | G | Temporary table | I | Index | O | Stored procedure | P | Partitioned table space if it is defined as LARGE or with the DSSIZE parm | R | Table space | S | Synonym | T | Table | V | View | G |
| A           | Alias                                                                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |     |                 |       |                                        |   |                 |   |       |   |                  |   |                                                                           |   |             |   |         |   |       |   |      |   |
| F           | User-defined function or cast function                                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |     |                 |       |                                        |   |                 |   |       |   |                  |   |                                                                           |   |             |   |         |   |       |   |      |   |
| G           | Temporary table                                                           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |     |                 |       |                                        |   |                 |   |       |   |                  |   |                                                                           |   |             |   |         |   |       |   |      |   |
| I           | Index                                                                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |     |                 |       |                                        |   |                 |   |       |   |                  |   |                                                                           |   |             |   |         |   |       |   |      |   |
| O           | Stored procedure                                                          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |     |                 |       |                                        |   |                 |   |       |   |                  |   |                                                                           |   |             |   |         |   |       |   |      |   |
| P           | Partitioned table space if it is defined as LARGE or with the DSSIZE parm |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |     |                 |       |                                        |   |                 |   |       |   |                  |   |                                                                           |   |             |   |         |   |       |   |      |   |
| R           | Table space                                                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |     |                 |       |                                        |   |                 |   |       |   |                  |   |                                                                           |   |             |   |         |   |       |   |      |   |
| S           | Synonym                                                                   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |     |                 |       |                                        |   |                 |   |       |   |                  |   |                                                                           |   |             |   |         |   |       |   |      |   |
| T           | Table                                                                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |     |                 |       |                                        |   |                 |   |       |   |                  |   |                                                                           |   |             |   |         |   |       |   |      |   |
| V           | View                                                                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |     |                 |       |                                        |   |                 |   |       |   |                  |   |                                                                           |   |             |   |         |   |       |   |      |   |
| DLOCATION   | CHAR(16)<br>NOT NULL                                                      | Always contains blanks                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           | S   |                 |       |                                        |   |                 |   |       |   |                  |   |                                                                           |   |             |   |         |   |       |   |      |   |
| DCOLLID     | CHAR(18)<br>NOT NULL                                                      | Name of the package collection.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  | G   |                 |       |                                        |   |                 |   |       |   |                  |   |                                                                           |   |             |   |         |   |       |   |      |   |
| DNAME       | CHAR(8)<br>NOT NULL                                                       | Name of the package.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             | G   |                 |       |                                        |   |                 |   |       |   |                  |   |                                                                           |   |             |   |         |   |       |   |      |   |
| DCONTOKEN   | CHAR(8)<br>NOT NULL                                                       | Consistency token for the package. This is either:<br><ul style="list-style-type: none"><li>• The “level” as specified by the LEVEL option when the package’s program was precompiled</li><li>• The timestamp indicating when the package’s program was precompiled, in an internal format.</li></ul>                                                                                                                                                                                                                                            | S   |                 |       |                                        |   |                 |   |       |   |                  |   |                                                                           |   |             |   |         |   |       |   |      |   |
| IBMREQD     | CHAR(1)<br>NOT NULL                                                       | A value of Y indicates that the row came from the basic machine-readable material (MRM) tape. For all other values, see “Release dependency indicators” on page 1005.                                                                                                                                                                                                                                                                                                                                                                            | G   |                 |       |                                        |   |                 |   |       |   |                  |   |                                                                           |   |             |   |         |   |       |   |      |   |
| DOWNER      | CHAR(8)<br>NOT NULL WITH<br>DEFAULT                                       | Owner of the package.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            | G   |                 |       |                                        |   |                 |   |       |   |                  |   |                                                                           |   |             |   |         |   |       |   |      |   |
| DTYPE       | CHAR(1)<br>NOT NULL WITH<br>DEFAULT                                       | Type of package:<br><table><tr><td>T</td><td>Trigger package</td></tr><tr><td>blank</td><td>Not a trigger package</td></tr></table>                                                                                                                                                                                                                                                                                                                                                                                                              | T   | Trigger package | blank | Not a trigger package                  | G |                 |   |       |   |                  |   |                                                                           |   |             |   |         |   |       |   |      |   |
| T           | Trigger package                                                           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |     |                 |       |                                        |   |                 |   |       |   |                  |   |                                                                           |   |             |   |         |   |       |   |      |   |
| blank       | Not a trigger package                                                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |     |                 |       |                                        |   |                 |   |       |   |                  |   |                                                                           |   |             |   |         |   |       |   |      |   |

**SYSIBM.SYSPACKLIST table**

Contains one or more rows for every local application plan bound with a package list. Each row represents a unique entry in the plan's package list.

| Column name | Data type             | Description                                                                                                                                                           | Use |
|-------------|-----------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| PLANNAME    | CHAR(8)<br>NOT NULL   | Name of the application plan.                                                                                                                                         | G   |
| SEQNO       | SMALLINT<br>NOT NULL  | Sequence number of the entry in the package list.                                                                                                                     | G   |
| LOCATION    | CHAR(16)<br>NOT NULL  | Location of the package. Blank if this is local. An asterisk (*) indicates location to be determined at run time.                                                     | G   |
| COLLID      | CHAR(18)<br>NOT NULL  | Collection name for the package. An asterisk (*) indicates that the collection name is determined at run time.                                                        | G   |
| NAME        | CHAR(8)<br>NOT NULL   | Name of the package. An asterisk (*) indicates an entire collection.                                                                                                  | G   |
| TIMESTAMP   | TIMESTAMP<br>NOT NULL | Timestamp indicating when the row was created.                                                                                                                        | G   |
| IBMREQD     | CHAR(1)<br>NOT NULL   | A value of Y indicates that the row came from the basic machine-readable material (MRM) tape. For all other values, see "Release dependency indicators" on page 1005. | G   |

## SYSIBM.SYSPACKSTMT table

Contains one or more rows for each statement in a package.

| Column name | Data type                                                                                                                                 | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    | Use |                      |   |                     |   |                       |   |                       |   |                                       |   |                                       |       |                                                                                                                                           |   |
|-------------|-------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|----------------------|---|---------------------|---|-----------------------|---|-----------------------|---|---------------------------------------|---|---------------------------------------|-------|-------------------------------------------------------------------------------------------------------------------------------------------|---|
| LOCATION    | CHAR(16)<br>NOT NULL                                                                                                                      | Always contains blanks                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         | S   |                      |   |                     |   |                       |   |                       |   |                                       |   |                                       |       |                                                                                                                                           |   |
| COLLID      | CHAR(18)<br>NOT NULL                                                                                                                      | Name of the package collection.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                | G   |                      |   |                     |   |                       |   |                       |   |                                       |   |                                       |       |                                                                                                                                           |   |
| NAME        | CHAR(8)<br>NOT NULL                                                                                                                       | Name of the package.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           | G   |                      |   |                     |   |                       |   |                       |   |                                       |   |                                       |       |                                                                                                                                           |   |
| CONTOKEN    | CHAR(8)<br>NOT NULL                                                                                                                       | Consistency token for the package. This is either:<br><ul style="list-style-type: none"> <li>• The “level” as specified by the LEVEL option when the package’s program was precompiled</li> <li>• The timestamp indicating when the package’s program was precompiled, in an internal format</li> </ul>                                                                                                                                                                                                                                                                                                                        | S   |                      |   |                     |   |                       |   |                       |   |                                       |   |                                       |       |                                                                                                                                           |   |
| SEQNO       | SMALLINT<br>NOT NULL                                                                                                                      | Sequence number of the row with respect to a statement in the package <sup>50</sup> . The numbering starts with 0.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             | G   |                      |   |                     |   |                       |   |                       |   |                                       |   |                                       |       |                                                                                                                                           |   |
| STMTNO      | SMALLINT<br>NOT NULL                                                                                                                      | The statement number of the statement in the source program. A statement number greater than 32767 is stored as zero <sup>50</sup> or as a negative number <sup>51</sup> . If the value is zero, see STMTNOI for the statement number.                                                                                                                                                                                                                                                                                                                                                                                         | G   |                      |   |                     |   |                       |   |                       |   |                                       |   |                                       |       |                                                                                                                                           |   |
| #<br>#      |                                                                                                                                           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |     |                      |   |                     |   |                       |   |                       |   |                                       |   |                                       |       |                                                                                                                                           |   |
| SECTNO      | SMALLINT<br>NOT NULL                                                                                                                      | The section number of the statement. <sup>51</sup>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             | G   |                      |   |                     |   |                       |   |                       |   |                                       |   |                                       |       |                                                                                                                                           |   |
| BINDERROR   | CHAR(1)<br>NOT NULL                                                                                                                       | Whether an SQL error was detected at bind time:<br><table style="margin-left: 20px;"> <tr> <td>N</td> <td>No</td> </tr> <tr> <td>Y</td> <td>Yes</td> </tr> </table>                                                                                                                                                                                                                                                                                                                                                                                                                                                            | N   | No                   | Y | Yes                 | G |                       |   |                       |   |                                       |   |                                       |       |                                                                                                                                           |   |
| N           | No                                                                                                                                        |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |     |                      |   |                     |   |                       |   |                       |   |                                       |   |                                       |       |                                                                                                                                           |   |
| Y           | Yes                                                                                                                                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |     |                      |   |                     |   |                       |   |                       |   |                                       |   |                                       |       |                                                                                                                                           |   |
| IBMREQD     | CHAR(1)<br>NOT NULL                                                                                                                       | A value of Y indicates that the row came from the basic machine-readable material (MRM) tape. For all other values, see “Release dependency indicators” on page 1005.                                                                                                                                                                                                                                                                                                                                                                                                                                                          | G   |                      |   |                     |   |                       |   |                       |   |                                       |   |                                       |       |                                                                                                                                           |   |
| VERSION     | VARCHAR(64)<br>NOT NULL                                                                                                                   | Version identifier for the package.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            | G   |                      |   |                     |   |                       |   |                       |   |                                       |   |                                       |       |                                                                                                                                           |   |
| STMT        | VARCHAR(254)<br>NOT NULL                                                                                                                  | All or a portion of the text for the SQL statement that the row represents.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    | S   |                      |   |                     |   |                       |   |                       |   |                                       |   |                                       |       |                                                                                                                                           |   |
| ISOLATION   | CHAR(1)<br>NOT NULL WITH<br>DEFAULT                                                                                                       | Isolation level for the SQL statement:<br><table style="margin-left: 20px;"> <tr> <td>R</td> <td>RR (repeatable read)</td> </tr> <tr> <td>T</td> <td>RS (read stability)</td> </tr> <tr> <td>S</td> <td>CS (cursor stability)</td> </tr> <tr> <td>U</td> <td>UR (uncommitted read)</td> </tr> <tr> <td>L</td> <td>KEEP UPDATE LOCKS for an RS isolation</td> </tr> <tr> <td>X</td> <td>KEEP UPDATE LOCKS for an RR isolation</td> </tr> <tr> <td>blank</td> <td>The WITH clause was not specified on this statement.<br/>The isolation level is recorded in SYSPACKAGE.ISOLATION and in SYSPLAN.ISOLATION.</td> </tr> </table> | R   | RR (repeatable read) | T | RS (read stability) | S | CS (cursor stability) | U | UR (uncommitted read) | L | KEEP UPDATE LOCKS for an RS isolation | X | KEEP UPDATE LOCKS for an RR isolation | blank | The WITH clause was not specified on this statement.<br>The isolation level is recorded in SYSPACKAGE.ISOLATION and in SYSPLAN.ISOLATION. | G |
| R           | RR (repeatable read)                                                                                                                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |     |                      |   |                     |   |                       |   |                       |   |                                       |   |                                       |       |                                                                                                                                           |   |
| T           | RS (read stability)                                                                                                                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |     |                      |   |                     |   |                       |   |                       |   |                                       |   |                                       |       |                                                                                                                                           |   |
| S           | CS (cursor stability)                                                                                                                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |     |                      |   |                     |   |                       |   |                       |   |                                       |   |                                       |       |                                                                                                                                           |   |
| U           | UR (uncommitted read)                                                                                                                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |     |                      |   |                     |   |                       |   |                       |   |                                       |   |                                       |       |                                                                                                                                           |   |
| L           | KEEP UPDATE LOCKS for an RS isolation                                                                                                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |     |                      |   |                     |   |                       |   |                       |   |                                       |   |                                       |       |                                                                                                                                           |   |
| X           | KEEP UPDATE LOCKS for an RR isolation                                                                                                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |     |                      |   |                     |   |                       |   |                       |   |                                       |   |                                       |       |                                                                                                                                           |   |
| blank       | The WITH clause was not specified on this statement.<br>The isolation level is recorded in SYSPACKAGE.ISOLATION and in SYSPLAN.ISOLATION. |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |     |                      |   |                     |   |                       |   |                       |   |                                       |   |                                       |       |                                                                                                                                           |   |

50. Rows in which the value of SEQNO, STMTNO, and SECTNO are zero are for internal use.

51. To convert a negative STMTNO to a meaningful statement number that corresponds to your precompile output, add 65536 to it. For example, -26472 is equivalent to +39064 (-26472 + 65536).

## SYSIBM.SYSPACKSTMT

| Column name | Data type                           | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    | Use |
|-------------|-------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| STATUS      | CHAR(1)<br>NOT NULL WITH<br>DEFAULT | Status of binding the statement:<br><br><b>A</b> Distributed - statement uses DB2 private protocol access. The statement will be parsed and executed at the server using defaults for input variables during access path selection.<br><b>B</b> Distributed - statement uses DB2 private protocol access. The statement will be parsed and executed at the server using values for input variables during access path selection.<br><b>C</b> Compiled - statement was bound successfully using defaults for input variables during access path selection.<br><b>D</b> Distributed - statement references a remote object using DB2 private protocol access (a three-part name), but DB2 will implicitly use DRDA access instead because the statement was bound with bind option DBPROTOCOL(DRDA). This option allows the use of three-part names with DRDA access, but it requires that the package be bound at the target remote site.<br><b>E</b> Explain - statement is an SQL EXPLAIN statement. The explain is done at bind time using defaults for input variables during access path selection.<br><b>F</b> Parsed - statement did not bind successfully and VALIDATE(RUN) was used. The statement will be rebound at execution time using values for input variables during access path selection.<br><b>G</b> Compiled - statement bound successfully, but REOPT is specified. The statement will be rebound at execution time using values for input variables during access path selection.<br><b>H</b> Parsed - statement is either a data definition statement or a statement that did not bind successfully and VALIDATE(RUN) was used. The statement will be rebound at execution time using defaults for input variables during access path selection. Data manipulation statements use defaults for input variables during access path selection.<br><b>I</b> Indefinite - statement is dynamic. The statement will be bound at execution time using defaults for input variables during access path selection.<br><b>J</b> Indefinite - statement is dynamic. The statement will be bound at execution time using values for input variables during access path selection.<br><b>K</b> Control - CALL statement.<br><b>L</b> Bad - the statement has some allowable error. The bind continues but the statement cannot be executed.<br><b>M</b> Parsed - statement references a table that is qualified with SESSION and was not bound because the table reference could be for a declared temporary table that will not be defined until the package or plan is run. The SQL statement will be rebound at execution time using values for input variables during access path selection.<br><b>blank</b> The statement is non-executable, or was bound in a DB2 release prior to Version 5. | S   |
| ACCESSPATH  | CHAR(1)<br>NOT NULL WITH<br>DEFAULT | For static statements, indicates if the access path for the statement is based on user-specified optimization hints. A value of 'H' indicates that optimization hints were used. A blank value indicates that the access path was determined without the use of optimization hints, or that there is no access path associated with the statement.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             | G   |
|             |                                     | For dynamic statements, the value is blank.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |     |
| # STMTNOI   | INTEGER<br>NOT NULL WITH<br>DEFAULT | If the value of STMTNO is 0, the column contains the statement number of the statement in the source program. If both STMTNO and STMTNOI are 0, the statement number is greater than 32767.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    | G   |

| Column name | Data type                              | Description                                                                                                                                                                                                                                                                                                                                                                                                 | Use |
|-------------|----------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| SECTNOI     | INTEGER<br>NOT NULL WITH<br>DEFAULT    | The section number of the statement.                                                                                                                                                                                                                                                                                                                                                                        | G   |
| EXPLAINABLE | CHAR(1)<br>NOT NULL WITH<br>DEFAULT    | Contains one of the following values:<br><br>Y      Indicates that the SQL statement can be used with the EXPLAIN function and may have rows describing its access path in the userid.PLAN_TABLE.<br><br>N      Indicates that the SQL statement does not have any rows describing its access path in the userid.PLAN_TABLE.<br><br>blank    Indicates that the SQL statement was bound prior to Version 7. | G   |
| QUERYNO     | INTEGER<br>NOT NULL WITH<br>DEFAULT -1 | The query number of the SQL statement in the source program. SQL statements bound prior to Version 7 have a default value of -1. Statements bound in Version 7 or later use the value specified on the QUERYNO clause on SELECT, UPDATE, INSERT, DELETE, EXPLAIN, and DECLARE CURSOR statements. If the QUERYNO clause is not specified, the query number is set to the statement number.                   | G   |

## SYSIBM.SYSPARMS

### SYSIBM.SYSPARMS table

Contains a row for each parameter of a routine or multiple rows for table parameters (one for each column of the table).

| Column name   | Data type            | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     | Use |
|---------------|----------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| SCHEMA        | CHAR(8)<br>NOT NULL  | Schema of the routine.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          | G   |
| OWNER         | CHAR(8)<br>NOT NULL  | Owner of the routine.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           | G   |
| NAME          | CHAR(18)<br>NOT NULL | Name of the routine.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            | G   |
| SPECIFICNAME  | CHAR(18)<br>NOT NULL | Specific name of the routine.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   | G   |
| ROUTINETYPE   | CHAR(1)<br>NOT NULL  | Type of routine:<br><b>F</b> User-defined function or cast function<br><b>P</b> Stored procedure                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                | G   |
| CAST_FUNCTION | CHAR(1)<br>NOT NULL  | Whether the routine is a cast function:<br><b>N</b> Not a cast function<br><b>Y</b> A cast function                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             | G   |
|               |                      | The only way to get a value of Y is if a user creates a distinct type when DB2 implicitly generates cast functions for the distinct type.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |     |
| PARMNAME      | CHAR(18)<br>NOT NULL | Name of the parameter.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          | G   |
| ROUTINEID     | INTEGER<br>NOT NULL  | Internal identifier of the routine.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             | S   |
| ROWTYPE       | CHAR(1)<br>NOT NULL  | Type of parameter:<br><b>P</b> Input parameter.<br><b>O</b> Output parameter; not applicable for functions<br><b>B</b> Both an input and an output parameter; not applicable for functions<br><b>R</b> Result before casting; not applicable for stored procedures<br><b>C</b> Result after casting; not applicable for stored procedures<br><b>S</b> Input parameter of the underlying built-in source function. For a sourced function and a given ORDINAL value: <ul style="list-style-type: none"><li>• The row with ROWTYPE = P describes the input parameter of the user-defined function (identified by ROUTINEID).</li><li>• The row with ROWTYPE = S describes the corresponding input parameter of the built-in function that is the underlying source function (identified by the SOURCESCHEMA and SOURCESPECIFIC values).</li></ul> | G   |
| #             | #                    | A value of 'X' indicates that the row is not used to describe a particular parameter of the routine. Instead, the row is used to record a CCSID for the encoding scheme specified in a PARAMETER CCSID clause, or a DATATYPEID for the representation of the variable length character string parameter of a LANGUAGE C routine, as specified in a PARAMETER VARCHAR clause.                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |     |
| #             | #                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |     |
| #             | #                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |     |

| Column name                         | Data type            | Description                                                                                                                                                                                                                                                                                                                                                | Use |
|-------------------------------------|----------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| ORDINAL                             | SMALLINT<br>NOT NULL | If ROWTYPE is B, O, P, or S, the ordinal number of the parameter within the routine signature.                                                                                                                                                                                                                                                             | G   |
| #<br>#<br>#<br>#<br>#               |                      | If ROWTYPE is C or R, the value depends on the type of function: <ul style="list-style-type: none"><li>• For a scalar function, the value is 0.</li><li>• For a table function, the value is the ordinal number of the column of the output table.</li></ul>                                                                                               |     |
|                                     |                      | If ROWTYPE is X, the value is 0.                                                                                                                                                                                                                                                                                                                           |     |
| TYPESCHEMA                          | CHAR(8)<br>NOT NULL  | Schema of the data type of the parameter.                                                                                                                                                                                                                                                                                                                  | G   |
| TYPENAME                            | CHAR(18)<br>NOT NULL | Name of the data type of the parameter.                                                                                                                                                                                                                                                                                                                    | G   |
| DATATYPEID<br>#<br>#<br>#<br>#<br># | INTEGER<br>NOT NULL  | For a built-in data type, the internal ID of the built-in type. For a distinct type, the internal ID of the distinct type. When ROWTYPE is X and ORDINAL is 0, a non-zero DATATYPEID specifies the actual representation, for a LANGUAGE C routine, of any varying length string parameters that appear in the routine's parameter list or RETURNS clause. | S   |
| SOURCETYPEID                        | INTEGER<br>NOT NULL  | For a built-in data type, 0. For a distinct type, the internal ID of the built-in data type upon which the distinct type is sourced.                                                                                                                                                                                                                       | S   |
| LOCATOR                             | CHAR(1)<br>NOT NULL  | Indicates whether a locator to a value, instead of the actual value, is to be passed or returned when the routine is called:<br><b>N</b> The actual value is to be passed.<br><b>Y</b> A locator to a value is to be passed                                                                                                                                | G   |
| TABLE                               | CHAR(1)<br>NOT NULL  | The data type of a column for a table parameter:<br><b>N</b> This is not a table parameter.<br><b>Y</b> This is a table parameter.                                                                                                                                                                                                                         | G   |
| TABLE_COLNO                         | SMALLINT<br>NOT NULL | For table parameters, the column number of the table.<br>Otherwise, the value is 0.                                                                                                                                                                                                                                                                        | G   |
| LENGTH                              | INTEGER<br>NOT NULL  | Length attribute of the parameter, or in the case of a decimal parameter, its precision.                                                                                                                                                                                                                                                                   | G   |
| SCALE                               | SMALLINT<br>NOT NULL | Scale of the data type of the parameter.                                                                                                                                                                                                                                                                                                                   | G   |
| SUBTYPE                             | CHAR(1)<br>NOT NULL  | If the data type is a distinct type, the subtype of the distinct type, which is based on the subtype of its source type:<br><b>B</b> The subtype is FOR BIT DATA.<br><b>S</b> The subtype is FOR SBCS DATA.<br><b>M</b> The subtype is FOR MIXED DATA.<br><b>blank</b> The source type is not a character type.                                            | G   |
| CCSID<br>#<br>#<br>#                | INTEGER<br>NOT NULL  | CCSID of the data type for character, graphic, date, time, and timestamp data types. When ROWTYPE is X and ORDINAL is 0, if the CCSID column is non-zero, it indicates the CCSID for all string parameters.                                                                                                                                                | G   |
| CAST_FUNCTION_ID                    | INTEGER<br>NOT NULL  | Internal function ID of the function used to cast the argument, if this function is sourced on another function, or result.<br>Otherwise, the value is 0. Not applicable for stored procedures.                                                                                                                                                            | S   |
| ENCODING_SCHEME                     | CHAR(1)<br>NOT NULL  | Encoding scheme of the parameter:<br><b>A</b> ASCII<br><b>E</b> EBCDIC<br><b>U</b> UNICODE<br><b>blank</b> The source type is not a character, graphic, or datetime type.                                                                                                                                                                                  | G   |
|                                     |                      |                                                                                                                                                                                                                                                                                                                                                            |     |
| IBMREQD                             | CHAR(1)<br>NOT NULL  | A value of Y indicates that the row came from the basic machine-readable material (MRM) tape. For all other values, see "Release dependency indicators" on page 1005.                                                                                                                                                                                      | G   |

## SYSIBM.SYSPKSYSTEM

### SYSIBM.SYSPKSYSTEM table

Contains zero or more rows for every package. Each row for a given package represents one or more connections to an environment in which the package could be executed.

| Column name | Data type            | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   | Use |
|-------------|----------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| LOCATION    | CHAR(16)<br>NOT NULL | Always contains blanks                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        | S   |
| COLLID      | CHAR(18)<br>NOT NULL | Name of the package collection.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               | G   |
| NAME        | CHAR(8)<br>NOT NULL  | Name of the package.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          | G   |
| CONTOKEN    | CHAR(8)<br>NOT NULL  | Consistency token for the package. This is either:<br><ul style="list-style-type: none"> <li>• The “level” as specified by the LEVEL option when the package’s program was precompiled</li> <li>• The timestamp indicating when the package’s program was precompiled, in an internal format.</li> </ul>                                                                                                                                                                                                                                                                                                                                                                                                      | S   |
| SYSTEM      | CHAR(8)<br>NOT NULL  | Environment. Values can be:<br><b>BATCH</b> TSO batch<br><b>CICS</b> Customer Information Control System<br><b>DB2CALL</b> DB2 call attachment facility<br><b>DLIBATCH</b> DLI batch support facility<br><b>IMSBMP</b> IMS BMP region<br><b>IMSMPP</b> IMS MPP and IFP region<br><b>REMOTE</b> remote server                                                                                                                                                                                                                                                                                                                                                                                                  | G   |
| ENABLE      | CHAR(1)<br>NOT NULL  | Indicates whether the connections represented by the row are enabled or disabled:<br><b>N</b> Disabled<br><b>Y</b> Enabled                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    | G   |
| CNAME       | CHAR(20)<br>NOT NULL | Identifies the connection or connections to which the row applies. Interpretation depends on the environment specified by SYSTEM. Values can be:<br><ul style="list-style-type: none"> <li>• Blank if SYSTEM=BATCH or SYSTEM=DB2CALL</li> <li>• The LU name for a database server if SYSTEM=REMOTE</li> <li>• Either the requester’s location (if the product is DB2) or the requester’s LU name enclosed in angle brackets if SYSTEM=REMOTE.</li> <li>• The name of a single connection if SYSTEM has any other value.</li> </ul> <p>CNAME can also be blank when SYSTEM is not equal to BATCH or DB2CALL. When this is so, the row applies to all servers or connections for the indicated environment.</p> | G   |
| IBMREQD     | CHAR(1)<br>NOT NULL  | A value of Y indicates that the row came from the basic machine-readable material (MRM) tape. For all other values, see “Release dependency indicators” on page 1005.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         | G   |

## SYSIBM.SYSPLAN table

Contains one row for each application plan.

| Column name | Data type           | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      | Use |
|-------------|---------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| NAME        | CHAR(8)<br>NOT NULL | Name of the application plan.                                                                                                                                                                                                                                                                                                                                                                                                                                                                    | G   |
| CREATOR     | CHAR(8)<br>NOT NULL | Authorization ID of the owner of the application plan.                                                                                                                                                                                                                                                                                                                                                                                                                                           | G   |
|             | CHAR(6)<br>NOT NULL | Not used.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        | N   |
| VALIDATE    | CHAR(1)<br>NOT NULL | Whether validity checking can be deferred until run time:<br><b>B</b> All checking must be performed during BIND.<br><b>R</b> Validation is done at run time for tables, views, and privileges that do not exist at bind time.                                                                                                                                                                                                                                                                   | G   |
| ISOLATION   | CHAR(1)<br>NOT NULL | Isolation level for the plan:<br><b>R</b> RR (repeatable read)<br><b>T</b> RS (read stability)<br><b>S</b> CS (cursor stability)<br><b>U</b> UR (uncommitted read)                                                                                                                                                                                                                                                                                                                               | G   |
| VALID       | CHAR(1)<br>NOT NULL | Whether the application plan is valid:<br><b>A</b> An ALTER TABLE statement changed the description of the table or base table of a view that is referred to by the application plan. The changes do not invalidate the plan.<br><b>H</b> An ALTER TABLE statement changed the description of the table or base table of a view that is referred to by the application plan. For releases of DB2 prior to Version 5, the change invalidates the application plan.<br><b>N</b> No<br><b>Y</b> Yes | G   |
| OPERATIVE   | CHAR(1)<br>NOT NULL | Whether the application plan can be allocated:<br><b>N</b> No; an explicit BIND or REBIND is required before the plan can be allocated<br><b>Y</b> Yes                                                                                                                                                                                                                                                                                                                                           | G   |
|             | CHAR(8)<br>NOT NULL | Not used.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        | N   |
| PLSIZE      | INTEGER<br>NOT NULL | Size of the base section <sup>52</sup> of the plan, in bytes.                                                                                                                                                                                                                                                                                                                                                                                                                                    | G   |
| IBMREQD     | CHAR(1)<br>NOT NULL | A value of Y indicates that the row came from the basic machine-readable material (MRM) tape. For all other values, see "Release dependency indicators" on page 1005.                                                                                                                                                                                                                                                                                                                            | G   |
| AVGSIZE     | INTEGER<br>NOT NULL | Average size, in bytes, of those sections <sup>52</sup> of the plan that contain SQL statements processed at bind time.                                                                                                                                                                                                                                                                                                                                                                          | G   |
| ACQUIRE     | CHAR(1)<br>NOT NULL | When resources are acquired:<br><b>A</b> At allocation<br><b>U</b> At first use                                                                                                                                                                                                                                                                                                                                                                                                                  | G   |
| RELEASE     | CHAR(1)<br>NOT NULL | When resources are released:<br><b>C</b> At commit<br><b>D</b> At deallocation                                                                                                                                                                                                                                                                                                                                                                                                                   | G   |
|             | CHAR(1)<br>NOT NULL | Not used                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         | N   |
|             | CHAR(1)<br>NOT NULL | Not used                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         | N   |

52. Plans are divided into *sections*. The base section of the plan must be in the EDM pool during the entire time the application program is executing. Other sections of the plan, corresponding roughly to sets of related SQL statements, are brought into the pool as needed.

## SYSIBM.SYSPLAN

| Column name   | Data type                            | Description                                                                                                                                                                                                                                                                                                                                                                                       | Use |
|---------------|--------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
|               | CHAR(1)<br>NOT NULL                  | Not used                                                                                                                                                                                                                                                                                                                                                                                          | N   |
| EXPLAN        | CHAR(1)<br>NOT NULL                  | EXPLAIN option specified for the plan; that is, whether information on the plan's statements was added to the owner's PLAN_TABLE table:<br><br>N No<br>Y Yes                                                                                                                                                                                                                                      | G   |
| EXPREDICATE   | CHAR(1)<br>NOT NULL                  | Indicates the CURRENTDATA option when the plan was bound or rebound:<br><br>B Data currency is not required for ambiguous cursors.<br>Allow blocking for ambiguous cursors.<br>C Data currency is required for ambiguous cursors. Inhibit blocking for ambiguous cursors.<br>N Blocking is inhibited for ambiguous cursors, but the plan was created before the CURRENTDATA option was available. | G   |
| BOUNDBY       | CHAR(8)<br>NOT NULL WITH<br>DEFAULT  | Primary authorization ID of the binder of the plan.                                                                                                                                                                                                                                                                                                                                               | G   |
| QUALIFIER     | CHAR(8)<br>NOT NULL WITH<br>DEFAULT  | Implicit qualifier for the unqualified table, view, index, and alias names in the static SQL statements of the plan.                                                                                                                                                                                                                                                                              | G   |
| CACHESIZE     | SMALLINT<br>NOT NULL WITH<br>DEFAULT | Size, in bytes, of the cache to be acquired for the plan. A value of zero indicates that no cache is used.                                                                                                                                                                                                                                                                                        | G   |
| PLENTRIES     | SMALLINT<br>NOT NULL WITH<br>DEFAULT | Number of package list entries for the plan. The negative of that number if there are rows for the plan in SYSIBM.SYPACKLIST but the plan was bound in a prior release after fall back.                                                                                                                                                                                                           | G   |
| DEFERPREP     | CHAR(1)<br>NOT NULL WITH<br>DEFAULT  | Whether the package was last bound with the DEFER(PREPARE) option:<br><br>N No<br>Y Yes                                                                                                                                                                                                                                                                                                           | G   |
| CURRENTSERVER | CHAR(16)<br>NOT NULL WITH<br>DEFAULT | Location name specified with the CURRENTSERVER option when the plan was last bound. Blank if none was specified, implying that the first server is the local DB2 subsystem.                                                                                                                                                                                                                       | G   |
| SYSENTRIES    | SMALLINT<br>NOT NULL WITH<br>DEFAULT | Number of rows associated with the plan in SYSIBM.SYSPLSYSTEM. The negative of that number if such rows exist but the plan was bound in a prior release after fall back. A negative value or zero means that all connections are enabled.                                                                                                                                                         | G   |
| DEGREE        | CHAR(3)<br>NOT NULL WITH<br>DEFAULT  | The DEGREE option used when the plan was last bound:<br><br>ANY DEGREE(ANY)<br>1 or blank DEGREE(1). Blank if the plan was migrated.                                                                                                                                                                                                                                                              | G   |
| SQLRULES      | CHAR(1)<br>NOT NULL WITH<br>DEFAULT  | The SQLRULES option used when the plan was last bound:<br><br>D or blank SQLRULES(DB2)<br>S SQLRULES(STD)<br>blank A migrated plan                                                                                                                                                                                                                                                                | G   |
| DISCONNECT    | CHAR(1)<br>NOT NULL WITH<br>DEFAULT  | The DISCONNECT option used when the plan was last bound:<br><br>E or blank DISCONNECT(EXPLICIT) (EXPLICIT)<br>A DISCONNECT(AUTOMATIC) (AUTOMATIC)<br>C DISCONNECT(CONDITIONAL)<br>blank A migrated plan                                                                                                                                                                                           | G   |
| GROUP_MEMBER  | CHAR(8)<br>NOT NULL WITH<br>DEFAULT  | The DB2 data sharing member name of the DB2 subsystem that performed the most recent bind. This column is blank if the DB2 subsystem was not in a DB2 data sharing environment when the bind was performed.                                                                                                                                                                                       | G   |

| Column name    | Data type                                | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    | Use |
|----------------|------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| DYNAMICRULES   | CHAR(1)<br>NOT NULL WITH<br>DEFAULT      | The DYNAMICRULES option used when the plan was last bound:<br><b>B</b> BIND. Dynamic SQL statements are executed with DYNAMICRULES bind behavior.<br><b>blank</b> RUN. Dynamic SQL statements in the plan are executed with DYNAMICRULES run behavior.                                                                                                                                                                                                                                                                                         | G   |
| BOUNDS         | TIMESTAMP<br>NOT NULL WITH<br>DEFAULT    | Time when the plan was bound.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  | G   |
| REOPTVAR       | CHAR(1)<br>NOT NULL WITH<br>DEFAULT 'N'  | Whether the access path is determined again at execution time using input variable values:<br><b>N</b> Bind option NOREOPT(VARS) indicates that the access path is determined at bind time.<br><b>Y</b> Bind option REOPT(VARS) indicates that the access path is determined at execution time for SQL statements with variable values.                                                                                                                                                                                                        | G   |
| KEEPDYNAMIC    | CHAR(1)<br>NOT NULL WITH<br>DEFAULT 'N'  | Whether prepared dynamic statements are to be purged at each commit point:<br><b>N</b> The bind option is KEEPDYNAMIC(NO). Prepared dynamic SQL statements are destroyed at commit or rollback.<br><b>Y</b> The bind option is KEEPDYNAMIC(YES). Prepared dynamic SQL statements are kept past commit or rollback.                                                                                                                                                                                                                             | G   |
| PATHSCHEMAS    | VARCHAR(254)<br>NOT NULL WITH<br>DEFAULT | SQL path specified on the BIND or REBIND command that bound the plan. The path is used to resolve unqualified data type, function, and stored procedure names used in certain contexts. If the PATH bind option was not specified, the value in the column is a zero length string; however, DB2 uses a default SQL path of: SYSIBM, SYSFUN, SYSPROC, <i>plan qualifier</i> .                                                                                                                                                                  | G   |
| DBPROTOCOL     | CHAR(1)<br>NOT NULL WITH<br>DEFAULT 'P'  | Whether remote access for SQL with three-part names is implemented with DRDA or DB2 private protocol access:<br><b>D</b> DRDA<br><b>P</b> DB2 private protocol                                                                                                                                                                                                                                                                                                                                                                                 | G   |
| FUNCTIONTS     | TIMESTAMP<br>NOT NULL WITH<br>DEFAULT    | Timestamp when the function was resolved. Set by the BIND and REBIND commands, but not by AUTOBIND.                                                                                                                                                                                                                                                                                                                                                                                                                                            | G   |
| OPTHINT        | CHAR(8)<br>NOT NULL WITH<br>DEFAULT      | Value of the OPTHINT bind option. Identifies rows in the authid.PLAN_TABLE to be used as input to the optimizer. Contains blanks if no rows in the authid.PLAN_TABLE are to be used as input.                                                                                                                                                                                                                                                                                                                                                  | G   |
| ENCODING_CCSID | INTEGER<br>NOT NULL WITH<br>DEFAULT      | The CCSID corresponding to the encoding scheme or CCSID as specified for the bind option ENCODING. The Encoding Scheme specified on the bind command:<br><b>ccsid</b> The specified or derived CCSID.<br><b>0</b> The default CCSID as specified on panel DSNTIPF at installation time. Used when the plan was bound prior to Version 7                                                                                                                                                                                                        | G   |
| IMMEDWRITE     | CHAR(1)<br>NOT NULL WITH<br>DEFAULT      | Indicates when writes of updated group buffer pool dependent pages are to be done. This option is only applicable for data sharing environments.<br><b>N</b> Bind option IMMEDWRITE(NO) indicates normal write activity is done.<br><b>Y</b> Bind option IMMEDWRITE(YES) indicates that immediate writes are done for updated group buffer pool dependent pages.<br><b>1</b> Bind option IMMEDWRITE(PH1) indicates that updated group buffer pool dependent pages are written at or before phase 1 commit.<br><b>blank</b> A migrated package. | G   |

## SYSIBM.SYSPLAN

| Column name | Data type                           | Description                                                                                                                | Use |
|-------------|-------------------------------------|----------------------------------------------------------------------------------------------------------------------------|-----|
| RELBOUND    | CHAR(1)<br>NOT NULL WITH<br>DEFAULT | The release when the package was bound or rebound.<br><b>blank</b> Bound prior to Version 7<br><b>K</b> Bound on Version 7 | G   |

## SYSIBM.SYSPLANAUTH table

Records the privileges that are held by users over application plans.

| Column name | Data type                             | Description                                                                                                                                                                                                                                                                                           | Use |
|-------------|---------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| GRANTOR     | CHAR(8)<br>NOT NULL                   | Authorization ID of the user who granted the privileges.                                                                                                                                                                                                                                              | G   |
| GRANTEE     | CHAR(8)<br>NOT NULL                   | Authorization ID of the user who holds the privileges. Could also be PUBLIC for a grant to PUBLIC.                                                                                                                                                                                                    | G   |
| NAME        | CHAR(8)<br>NOT NULL                   | Name of the application plan on which the privileges are held.                                                                                                                                                                                                                                        | G   |
|             | CHAR(12)<br>NOT NULL                  | Internal use only                                                                                                                                                                                                                                                                                     | I   |
|             | CHAR(6)<br>NOT NULL                   | Not used.                                                                                                                                                                                                                                                                                             | N   |
|             | CHAR(8)<br>NOT NULL                   | Not used.                                                                                                                                                                                                                                                                                             | N   |
|             | CHAR(1)<br>NOT NULL                   | Not used                                                                                                                                                                                                                                                                                              | N   |
| AUTHHOWGOT  | CHAR(1)<br>NOT NULL                   | Authorization level of the user from whom the privileges were received. This authorization level is not necessarily the highest authorization level of the grantor.<br><br><b>blank</b> Not applicable<br><b>C</b> DBCTL<br><b>D</b> DBADM<br><b>L</b> SYSCTRL<br><b>M</b> DBMAINT<br><b>S</b> SYSADM | G   |
| BINDAUTH    | CHAR(1)<br>NOT NULL                   | Whether the GRANTEE can use the BIND, REBIND, or FREE subcommands against the plan:<br><br><b>blank</b> Privilege is not held<br><b>G</b> Privilege is held with the GRANT option<br><b>Y</b> Privilege is held without the GRANT option                                                              | G   |
| EXECUTEAUTH | CHAR(1)<br>NOT NULL                   | Whether the GRANTEE can run application programs that use the application plan:<br><br><b>blank</b> Privilege is not held<br><b>G</b> Privilege is held with the GRANT option<br><b>Y</b> Privilege is held without the GRANT option                                                                  | G   |
| IBMREQD     | CHAR(1)<br>NOT NULL                   | A value of Y indicates that the row came from the basic machine-readable material (MRM) tape. For all other values, see "Release dependency indicators" on page 1005.                                                                                                                                 | G   |
| GRANTEDTS   | TIMESTAMP<br>NOT NULL WITH<br>DEFAULT | Time when the GRANT statement was executed.                                                                                                                                                                                                                                                           | G   |

**SYSIBM.SYSPLANDEP table**

Records the dependencies of plans on tables, views, aliases, synonyms, table spaces, indexes, functions, and stored procedures.

| Column name | Data type               | Description                                                                                                                                                                                                                                                                            | Use |
|-------------|-------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| BNAME       | VARCHAR(18)<br>NOT NULL | The name of an object the plan depends on.                                                                                                                                                                                                                                             | G   |
| BCREATOR    | CHAR(8)<br>NOT NULL     | If BNAME is a table space, its database. Otherwise, the authorization ID of the owner of BNAME.                                                                                                                                                                                        | G   |
| BTYPE       | CHAR(1)<br>NOT NULL     | Type of object identified by BNAME:<br><br>A Alias<br>F User-defined function or cast function<br>G Temporary table<br>I Index<br>O Stored procedure<br>P Partitioned table space if it is defined as LARGE or with the DSSIZE parm<br>R Table space<br>S Synonym<br>T Table<br>V View | G   |
| #           |                         |                                                                                                                                                                                                                                                                                        |     |
| DNAME       | CHAR(8)<br>NOT NULL     | Name of the plan.                                                                                                                                                                                                                                                                      | G   |
| IBMREQD     | CHAR(1)<br>NOT NULL     | A value of Y indicates that the row came from the basic machine-readable material (MRM) tape. For all other values, see "Release dependency indicators" on page 1005.                                                                                                                  | G   |

---

## SYSIBM.SYSPLSYSTEM table

Contains zero or more rows for every plan. Each row for a given plan represents one or more connections to an environment in which the plan could be used.

| Column name | Data type           | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                 | Use |
|-------------|---------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| NAME        | CHAR(8)<br>NOT NULL | Name of the plan.                                                                                                                                                                                                                                                                                                                                                                                                                                                           | G   |
| SYSTEM      | CHAR(8)<br>NOT NULL | Environment. Values can be:<br><b>BATCH</b> TSO batch<br><b>DB2CALL</b> DB2 call attachment facility<br><b>CICS</b> Customer Information Control System<br><b>DLIBATCH</b> DLI batch support facility<br><b>IMSBMP</b> IMS BMP region<br><b>IMSMPP</b> IMS MPP or IFP region                                                                                                                                                                                                | G   |
| ENABLE      | CHAR(1)<br>NOT NULL | Indicates whether the connections represented by the row are enabled or disabled:<br><b>N</b> Disabled<br><b>Y</b> Enabled                                                                                                                                                                                                                                                                                                                                                  | G   |
| CNAME       | CHAR(8)<br>NOT NULL | Identifies the connection or connections to which the row applies. Interpretation depends on the environment specified by SYSTEM. Values can be: <ul style="list-style-type: none"><li>• Blank if SYSTEM=BATCH or SYSTEM=DB2CALL</li><li>• The name of a single connection if SYSTEM has any other value</li></ul> CNAME can also be blank when SYSTEM is not equal to BATCH or DB2CALL. When this is so, the row applies to all connections for the indicated environment. | G   |
| IBMREQD     | CHAR(1)<br>NOT NULL | A value of Y indicates that the row came from the basic machine-readable material (MRM) tape. For all other values, see "Release dependency indicators" on page 1005.                                                                                                                                                                                                                                                                                                       | G   |

**SYSIBM.SYSPROCEDURES table**

In releases of DB2 for OS/390 and z/OS prior to Version 6, users were required to use the SYSPROCEDURES catalog table to define stored procedures to DB2. In Version 6 and Version 7, the SYSROUTINES catalog table contains information about stored procedures. When the versions were installed, the rows in SYSPROCEDURES that had non-blank values for AUTHID and LUNAME were copied, with appropriate formatting, to SYSROUTINES.

Although this version of DB2 for OS/390 and z/OS does not use SYSPROCEDURES, SYSPROCEDURES is available for fallback to Version 5. For information about falling back and remigrating, see *DB2 Installation Guide*. However, any procedures that are defined with this version will not be available for fallback to Version 5. Likewise, any procedure definitions that are altered for this version with the ALTER PROCEDURE statement will not be changed in SYSPROCEDURES and thus will not be available in Version 5.

| Column name | Data type                           | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    | Use |
|-------------|-------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| PROCEDURE   | CHAR(18)<br>NOT NULL                | Name of the stored procedure specified on the SQL CALL statement.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              | G   |
| AUTHID      | CHAR(8)<br>NOT NULL WITH<br>DEFAULT | SQL authorization ID of the user running the SQL application that issued the SQL CALL statement. When the SQL CALL statement is received from a remote location, this column is compared to the value of the authorization ID after outbound and inbound name translation operations have been performed.<br><br>If AUTHID is blank, values in this row apply to all authorization IDs.                                                                                                                                                                                                                                                        | G   |
| LUNAME      | CHAR(8)<br>NOT NULL WITH<br>DEFAULT | LUNAME of the system that issued the SQL CALL statement.<br><ul style="list-style-type: none"> <li>• If the LUNAME column contains the local DB2 system's LUNAME, this row applies to local applications that issue the SQL CALL statement.</li> <li>• If the LUNAME column contains the LUNAME of a remote client, this row applies to SQL CALL statements received from that remote client.</li> <li>• If LUNAME is blank, the values in this row apply to all systems, including the local DB2 system and clients connected through TCP/IP or SNA.</li> </ul><br>To ease migration to future releases of DB2, specify blanks in this field. | G   |
| LOADMOD     | CHAR(8)<br>NOT NULL                 | Member name of the MVS load module that DB2 should load to satisfy the request for the stored procedure.<br><br>When the value of LANGUAGE is COMPJAVA, this column value is not used.                                                                                                                                                                                                                                                                                                                                                                                                                                                         | G   |
| LINKAGE     | CHAR(1)<br>NOT NULL WITH<br>DEFAULT | Linkage convention used to pass parameters to the stored procedure:<br><b>N</b> The SIMPLE WITH NULLS convention is used where an indicator array is passed to the stored procedure. Null input parameters are allowed.<br><b>blank</b> The SIMPLE linkage convention is used where input parameters cannot be null.<br><br>Conventions for passing parameters to stored procedures are described in Part 6 of <i>DB2 Application Programming and SQL Guide</i> .                                                                                                                                                                              | G   |

## SYSIBM.SYSPROCEDURES

| Column name       | Data type                               | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    | Use |
|-------------------|-----------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| COLLID            | CHAR(18)<br>NOT NULL                    | Name of the package collection to use when the stored procedure is executed.<br><br>A blank value indicates that the package collection is the same as the package collection of the program that issued the SQL CALL statement.                                                                                                                                                                                                                                                               | G   |
| LANGUAGE          | CHAR(8)<br>NOT NULL                     | Programming language used to create the stored procedure. Possible values are 'ASSEMBLE', 'PLI', 'COBOL', 'C', 'REXX', or 'COMPJAVA'.                                                                                                                                                                                                                                                                                                                                                          | G   |
| ASUTIME           | INTEGER<br>NOT NULL WITH<br>DEFAULT     | Number of service units permitted for any single invocation of this stored procedure. If ASUTIME is zero, there is no limit on the service units.<br><br>If a stored procedure uses more service units than allowed by the ASUTIME value, DB2 cancels the stored procedure.                                                                                                                                                                                                                    | G   |
| STAYRESIDENT      | CHAR(1)<br>NOT NULL WITH<br>DEFAULT     | Determines whether the stored procedure load module is deleted from memory when the stored procedure ends.<br><br><b>Y</b> The load module remains resident in memory after the stored procedure ends.<br><br><b>blank</b> The load module is deleted from memory after the stored procedure ends.                                                                                                                                                                                             | G   |
| IBMREQD           | CHAR(1)<br>NOT NULL                     | A value of Y indicates that the row came from the basic machine-readable material (MRM) tape. For all other values, see "Release dependency indicators" on page 1005.                                                                                                                                                                                                                                                                                                                          | G   |
| RUNOPTS           | VARCHAR(254)<br>NOT NULL                | The Language Environment (Language Environment for MVS & VM) run-time options to use for this stored procedure. If this column contains an empty string, the installation default Language Environment run-time options are used.<br><br>When the value of LANGUAGE is COMPJAVA, this column value is the stored procedure program name, in the format <i>class.method</i> .<br><br>An example Language Environment run-time option list follows:<br>'MSGFILE(OUTFILE),RPTSTG(ON),RPTOPTS(ON)' | G   |
| PARMLIST          | VARCHAR(3000)<br>NOT NULL               | Defines the parameter list expected by the stored procedure. For syntax and a description of the information contained in the PARMLIST string, see Part 6 of <i>DB2 Application Programming and SQL Guide</i> .                                                                                                                                                                                                                                                                                | G   |
| RESULT_SETS       | SMALLINT<br>NOT NULL WITH<br>DEFAULT    | Maximum number of query result sets that can be returned by this stored procedure.<br><br>Zero indicates there are no query result sets.                                                                                                                                                                                                                                                                                                                                                       | G   |
| WLM_ENV           | CHAR(18)<br>NOT NULL WITH<br>DEFAULT    | Name of the WLM environment to be used to run this stored procedure.<br><br>A blank value results in the stored procedure being run in the DB2-established stored procedures address space.                                                                                                                                                                                                                                                                                                    | G   |
| PGM_TYPE          | CHAR(1)<br>NOT NULL WITH<br>DEFAULT 'M' | Whether the stored procedure runs as a main routine or a subroutine:<br><br><b>M</b> The stored procedure runs as a main routine.<br><b>S</b> The stored procedure runs as a subroutine.                                                                                                                                                                                                                                                                                                       | G   |
| EXTERNAL_SECURITY | CHAR(1)<br>NOT NULL WITH<br>DEFAULT 'N' | Whether a special RACF environment is required to control access to non-SQL resources:<br><br><b>N</b> RACF access to non-SQL resources is not required for the stored procedure. This option is sufficient when the stored procedure only accesses SQL objects.<br><br><b>Y</b> A RACF environment should be automatically created by DB2 each time the stored procedure is invoked so that RACF can manage access to non-SQL resources.                                                      | G   |

## SYSIBM.SYSPROCEDURES

| Column name      | Data type                   | Description                                                                                                                                                                                                                                                                           | Use |
|------------------|-----------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| COMMIT_ON_RETURN | CHAR(1)<br>WITH DEFAULT 'N' | Whether the unit of work is always to be committed immediately upon successful return (non-negative SQLCODE) from this stored procedure:<br><b>N</b> The unit of work is to continue.<br><b>Y</b> The unit of work is to be committed.<br>A null value means the same as the value N. | G   |

**SYSIBM.SYSRELS table**

Contains one row for every referential constraint.

| Column name  | Data type                               | Description                                                                                                                                                           | Use |
|--------------|-----------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| CREATOR      | CHAR(8)<br>NOT NULL                     | Authorization ID of the owner of the dependent table of the referential constraint.                                                                                   | G   |
| TBNAME       | VARCHAR(18)<br>NOT NULL                 | Name of the dependent table of the referential constraint.                                                                                                            | G   |
| RELNAME      | CHAR(8)<br>NOT NULL                     | Constraint name.                                                                                                                                                      | G   |
| REFTBNAME    | VARCHAR(18)<br>NOT NULL                 | Name of the parent table of the referential constraint.                                                                                                               | G   |
| REFTBCREATOR | CHAR(8)<br>NOT NULL                     | Authorization ID of the owner of the parent table.                                                                                                                    | G   |
| COLCOUNT     | SMALLINT<br>NOT NULL                    | Number of columns in the foreign key.                                                                                                                                 | G   |
| DELETERULE   | CHAR(1)<br>NOT NULL                     | Type of delete rule for the referential constraint:<br><br>A NO ACTION<br>C CASCADE<br>N SET NULL<br>R RESTRICT                                                       | G   |
| IBMREQD      | CHAR(1)<br>NOT NULL                     | A value of Y indicates that the row came from the basic machine-readable material (MRM) tape. For all other values, see "Release dependency indicators" on page 1005. | G   |
| RELOBID1     | SMALLINT<br>NOT NULL WITH<br>DEFAULT    | Internal identifier of the constraint with respect to the database that contains the parent table.                                                                    | S   |
| RELOBID2     | SMALLINT<br>NOT NULL WITH<br>DEFAULT    | Internal identifier of the constraint with respect to the database that contains the dependent table.                                                                 | S   |
| TIMESTAMP    | TIMESTAMP<br>NOT NULL WITH<br>DEFAULT   | Date and time the constraint was defined. If the constraint is between catalog tables prior to DB2 Version 2 Release 3, the value is '1985-04-01-00.00.00.000000.'    | G   |
| IXOWNER      | CHAR(8)<br>NOT NULL WITH<br>DEFAULT     | Owner of unique non-primary index used for the parent key. '99999999' if the enforcing index has been dropped. Blank if the enforcing index is a primary index.       | G   |
| IXNAME       | VARCHAR(18)<br>NOT NULL WITH<br>DEFAULT | Name of unique non-primary index used for a parent key. '99999999' if the enforcing index has been dropped. Blank if the enforcing index is a primary index.          | G   |

**SYSIBM.SYSRESAUTH table**

Records CREATE IN and PACKADM ON privileges for collections; USAGE privileges for distinct types; and USE privileges for buffer pools, storage groups, and table spaces.

| Column name | Data type                             | Description                                                                                                                                                                                                                                                                                                                                                                                                     | Use |
|-------------|---------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| GRANTOR     | CHAR(8)<br>NOT NULL                   | Authorization ID of the user who granted the privilege.                                                                                                                                                                                                                                                                                                                                                         | G   |
| GRANTEE     | CHAR(8)<br>NOT NULL                   | Authorization ID of the user who holds the privilege. Could also be PUBLIC for a grant to PUBLIC.                                                                                                                                                                                                                                                                                                               | G   |
| QUALIFIER   | CHAR(8)<br>NOT NULL                   | Qualifier of the table space (the database name) if the privilege is for a table space (OBTYPE='R'). The schema name of the distinct type if the privilege is for a distinct type (OBTYPE='D'). The schema name of the JAR (OBTYPE='J') if the privilege is for a JAR. The value is PACKADM if the privilege is for a collection (OBTYPE='C') and the authority held is PACKADM. Otherwise, the value is blank. | G   |
| #<br>#      |                                       |                                                                                                                                                                                                                                                                                                                                                                                                                 |     |
| NAME        | CHAR(18)<br>NOT NULL                  | Name of the buffer pool, collection, DB2 storage group, distinct type, JAR, or table space. Could also be ALL when USE OF ALL BUFFERPOOLS is granted.                                                                                                                                                                                                                                                           | G   |
|             | CHAR(1)<br>NOT NULL                   | Internal use only                                                                                                                                                                                                                                                                                                                                                                                               | I   |
| AUTHHOWGOT  | CHAR(1)<br>NOT NULL                   | Authorization level of the user from whom the privileges were received. This authorization level is not necessarily the highest authorization level of the grantor.<br><b>blank</b> Not applicable<br><b>C</b> DBCTL<br><b>D</b> DBADM<br><b>L</b> SYSCTRL<br><b>M</b> DBMAINT<br><b>S</b> SYSADM<br><b>P</b> PACKADM (on a specific collection)<br><b>A</b> PACKADM (on collection *)                          | G   |
| OBTYPE      | CHAR(1)<br>NOT NULL                   | Type of object:<br><b>B</b> Buffer pool<br><b>C</b> Collection<br><b>D</b> Distinct type<br><b>R</b> Table space<br><b>S</b> Storage group<br><b>J</b> JAR (Java ARchive file)                                                                                                                                                                                                                                  | G   |
|             |                                       |                                                                                                                                                                                                                                                                                                                                                                                                                 |     |
|             | CHAR(12)<br>NOT NULL                  | Internal use only                                                                                                                                                                                                                                                                                                                                                                                               | I   |
|             | CHAR(6)<br>NOT NULL                   | Not used.                                                                                                                                                                                                                                                                                                                                                                                                       | N   |
|             | CHAR(8)<br>NOT NULL                   | Not used.                                                                                                                                                                                                                                                                                                                                                                                                       | N   |
| USEAUTH     | CHAR(1)<br>NOT NULL                   | Whether the privilege is held with the GRANT option:<br><b>G</b> Privilege is held with the GRANT option<br><b>Y</b> Privilege is held without the GRANT option                                                                                                                                                                                                                                                 | G   |
|             |                                       | The authority held is PACKADM when the OBTYPE is C (a collection) and QUALIFIER is PACKADM. The authority held is CREATE IN when the OBTYPE is C and QUALIFIER is blank.                                                                                                                                                                                                                                        |     |
| IBMREQD     | CHAR(1)<br>NOT NULL                   | A value of Y indicates that the row came from the basic machine-readable material (MRM) tape. For all other values, see "Release dependency indicators" on page 1005.                                                                                                                                                                                                                                           | G   |
| GRANTEDTS   | TIMESTAMP<br>NOT NULL WITH<br>DEFAULT | Time when the GRANT statement was executed.                                                                                                                                                                                                                                                                                                                                                                     | G   |

**SYSIBM.SYSROUTINEAUTH table**

Records the privileges that are held by users on routines. (A routine can be a user-defined function, cast function, or stored procedure.)

| Column name  | Data type             | Description                                                                                                                                                                                                                                                                                                                                                                                                    | Use |
|--------------|-----------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| GRANTOR      | CHAR(8)<br>NOT NULL   | Authorization ID of the user who granted the privilege.                                                                                                                                                                                                                                                                                                                                                        | G   |
| GRANTEE      | CHAR(8)<br>NOT NULL   | Authorization ID of the user who holds the privilege or the name of a plan or package that uses the privilege. Can also be PUBLIC for a grant to PUBLIC.                                                                                                                                                                                                                                                       | G   |
| SCHEMA       | CHAR(8)<br>NOT NULL   | Schema of the routine                                                                                                                                                                                                                                                                                                                                                                                          | G   |
| SPECIFICNAME | CHAR(18)<br>NOT NULL  | Specific name of the routine. An asterisk (*) if the privilege is held on all routines in the schema.                                                                                                                                                                                                                                                                                                          | G   |
| GRANTEDTS    | TIMESTAMP<br>NOT NULL | Time when the GRANT statement was executed.                                                                                                                                                                                                                                                                                                                                                                    | G   |
| ROUTINETYPE  | CHAR(1)<br>NOT NULL   | Type of routine:<br><b>F</b> User-defined function or cast function<br><b>P</b> Stored procedure                                                                                                                                                                                                                                                                                                               | G   |
| GRANTEETYPE  | CHAR(1)<br>NOT NULL   | Type of grantee:<br><b>blank</b> An authorization ID<br><b>P</b> An application plan or package. The grantee is a package if COLLID is not blank.<br><b>R</b> Internal use only                                                                                                                                                                                                                                | G   |
| AUTHHOWGOT   | CHAR(1)<br>NOT NULL   | Authorization level of the user from whom the privileges were received. This authorization level is not necessarily the highest authorization level of the grantor.<br><br>This field is also used to indicate that the privilege was held on all schemas by the grantor.<br><b>blank</b> Not applicable<br><b>1</b> Grantor had privilege on schema.* at time of grant<br><b>L</b> SYSCTRL<br><b>S</b> SYSADM | G   |
| EXECUTEAUTH  | CHAR(1)<br>NOT NULL   | Whether GRANTEE can execute the routine:<br><b>Y</b> Privilege is held without GRANT option.<br><b>G</b> Privilege is held with GRANT option.                                                                                                                                                                                                                                                                  | G   |
| COLLID       | CHAR(18)<br>NOT NULL  | If the GRANTEE is a package, its collection name. Otherwise, the value is blank.                                                                                                                                                                                                                                                                                                                               | G   |
| CONTOKEN     | CHAR(8)<br>NOT NULL   | If the GRANTEE is a package, the consistency token of the DBRM from which the package was derived. Otherwise, the value is blank.                                                                                                                                                                                                                                                                              | G   |
| IBMREQD      | CHAR(1)<br>NOT NULL   | A value of Y indicates that the row came from the basic machine-readable material (MRM) tape. For all other values, see "Release dependency indicators" on page 1005.                                                                                                                                                                                                                                          | G   |

## SYSIBM.SYSROUTINES

### SYSIBM.SYSROUTINES table

Contains a row for every routine. (A routine can be a user-defined function, cast function, or stored procedure.)

| Column name    | Data type            | Description                                                                                                                                                                                                                                                | Use |
|----------------|----------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| SCHEMA         | CHAR(8)<br>NOT NULL  | Schema of the routine.                                                                                                                                                                                                                                     | G   |
| OWNER          | CHAR(8)<br>NOT NULL  | Owner of the routine.                                                                                                                                                                                                                                      | G   |
| NAME           | CHAR(18)<br>NOT NULL | Name of the routine.                                                                                                                                                                                                                                       | G   |
| ROUTINETYPE    | CHAR(1)<br>NOT NULL  | Type of routine:<br><b>F</b> User-defined function or cast function<br><b>P</b> Stored procedure                                                                                                                                                           | G   |
| CREATEDBY      | CHAR(8)<br>NOT NULL  | Authorization ID under which the routine was created.                                                                                                                                                                                                      | G   |
| SPECIFICNAME   | CHAR(18)<br>NOT NULL | Specific name of the routine.                                                                                                                                                                                                                              | G   |
| ROUTINEID      | INTEGER<br>NOT NULL  | Internal identifier of the routine.                                                                                                                                                                                                                        | S   |
| RETURN_TYPE    | INTEGER<br>NOT NULL  | Internal identifier of the result data type of the function. The column contains a -2 if the function is a table function.                                                                                                                                 | S   |
| ORIGIN         | CHAR(1)<br>NOT NULL  | Origin of the routine:<br><b>E</b> External user-defined function (external table or external scalar) or stored procedure<br><b>Q</b> SQL function<br><b>U</b> Sourced on user-defined function or built-in function<br><b>S</b> System-generated function | G   |
| #              |                      |                                                                                                                                                                                                                                                            |     |
| FUNCTION_TYPE  | CHAR(1)<br>NOT NULL  | Type of function:<br><b>C</b> Column function<br><b>S</b> Scalar function<br><b>T</b> Table function<br><b>blank</b> For a stored procedure (ROUTINETYPE = 'P')                                                                                            | G   |
| PARM_COUNT     | SMALLINT<br>NOT NULL | Number of parameters for the routine.                                                                                                                                                                                                                      | G   |
| LANGUAGE       | CHAR(8)<br>NOT NULL  | Implementation language of the routine:<br><b>ASSEMBLE</b><br><b>C</b><br><b>COBOL</b><br><b>COMPJAVA</b><br><b>JAVA</b><br><b>PLI</b><br><b>REXX</b><br><b>SQL</b><br><b>blank</b> If ROUTINETYPE='F' and ORIGIN is not 'E' or not 'Q'.                   | G   |
| #              |                      |                                                                                                                                                                                                                                                            |     |
| COLLID         | CHAR(18)<br>NOT NULL | Name of the package collection to be used when the routine is executed. A blank value indicates the package collection is the same as the package collection of the program that invoked the routine.                                                      | G   |
| SOURCESCHEMA   | CHAR(8)<br>NOT NULL  | If ORIGIN is 'U' and ROUTINETYPE is 'F', the schema of the source user-defined function ('SYSIBM' for a source built-in function). Otherwise, the value is blank.                                                                                          | G   |
| SOURCESPECIFIC | CHAR(18)<br>NOT NULL | If ORIGIN is 'U' and ROUTINETYPE is 'F', the specific name of the source user-defined function or source built-in function name. Otherwise, the value is blank.                                                                                            | G   |

| Column name                  | Data type           | Description                                                                                                                                                                                                                                                                                                                                                | Use |
|------------------------------|---------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| DETERMINISTIC<br>#<br>#<br># | CHAR(1)<br>NOT NULL | The deterministic option of an external function or a stored procedure:<br><b>N</b> Indeterminate (results may differ with a given set of input values).<br><b>Y</b> Deterministic (results are consistent).<br><b>blank</b> ROUTINETYPE='F' and ORIGIN is not 'E' or not 'Q'(the routine is a function, but not an external function or an SQL function). | G   |
| EXTERNAL_ACTION              | CHAR(1)<br>NOT NULL | The external action option of an external function or an SQL function:<br><b>N</b> Function has no side effects.<br><b>E</b> Function has external side effects so that the number of invocations is important.<br><b>blank</b> ORIGIN is not 'E' or 'Q'for the function (ROUTINETYPE='F'), or it is a stored procedure (ROUTINETYPE='P').                 | G   |
| NULL_CALL                    | CHAR(1)<br>NOT NULL | The CALLED ON NOT NULL INPUT option of an external function or stored procedure:<br><b>N</b> The routine is not called if any parameter has a NULL value.<br><b>Y</b> The routine is called if any parameter has a NULL value.<br><b>blank</b> ROUTINETYPE='F' and ORIGIN is not 'E' (the routine is a function, but not an external function).            | G   |
| CAST_FUNCTION                | CHAR(1)<br>NOT NULL | Whether the routine is a cast function:<br><b>N</b> The routine is not a cast function.<br><b>Y</b> The routine is a cast function.                                                                                                                                                                                                                        | G   |
|                              |                     | A cast function is generated by DB2 for a CREATE DISTINCT TYPE statement,                                                                                                                                                                                                                                                                                  |     |
| SCRATCHPAD                   | CHAR(1)<br>NOT NULL | The SCRATCHPAD option of an external function:<br><b>N</b> This function does not have a SCRATCHPAD.<br><b>Y</b> This function has a SCRATCHPAD.<br><b>blank</b> ORIGIN is not 'E' for the function (ROUTINETYPE='F'), or it is a stored procedure (ROUTINETYPE='P').                                                                                      | G   |
| SCRATCHPAD_LENGTH            | INTEGER<br>NOT NULL | Length of the scratchpad if the ORIGIN is 'E' for the function (ROUTINETYPE='F') and NO SCRATCHPAD is not specified. Otherwise, the value is 0.                                                                                                                                                                                                            | G   |
| FINAL_CALL                   | CHAR(1)<br>NOT NULL | The FINAL CALL option of an external function:<br><b>N</b> A final call will not be made to the function.<br><b>Y</b> A final call will be made to the function.<br><b>blank</b> ORIGIN is not 'E' for the function (ROUTINETYPE='F'), or it is a stored procedure (ROUTINETYPE='P').                                                                      | G   |
| PARALLEL                     | CHAR(1)<br>NOT NULL | The PARALLEL option of an external function:<br><b>A</b> This function can be invoked by parallel tasks.<br><b>D</b> This function cannot be invoked by parallel tasks.<br><b>blank</b> ORIGIN is not 'E' for the function (ROUTINETYPE='F'), or it is a stored procedure (ROUTINETYPE='P').                                                               | G   |

## SYSIBM.SYSROUTINES

| Column name     | Data type            | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 | Use |
|-----------------|----------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| PARAMETER_STYLE | CHAR(1)<br>NOT NULL  | The PARAMETER STYLE option of an external function or stored procedure:<br><br><b>D</b> DB2SQL. All parameters are passed to the external function or stored procedure according to the DB2SQL standard convention.<br><b>G</b> GENERAL. All parameters are passed to the stored procedure according to the GENERAL standard convention.<br><b>N</b> GENERAL CALL WITH NULLS. All parameters are passed to the stored procedure according to the GENERAL WITH NULLS convention.<br><b>J</b> JAVA. All parameters are passed to the function or procedure according to the conventions for JAVA and SQLJ specifications.<br><b>blank</b> The column is blank if the ORIGIN is not 'E' or if LANGUAGE is SQL. | G   |
| FENCED          | CHAR(1)<br>NOT NULL  | <b>Y</b> Indicates that this routine runs separately in the DB2 address space. All user-defined routines run in the DB2 address space.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      | G   |
| #               |                      | <b>blank</b> ORIGIN is 'Q' .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |     |
| SQL_DATA_ACCESS | CHAR(1)<br>NOT NULL  | The SQL statements that are allowed in an external function, SQL function, or stored procedure:<br><br><b>C</b> CONTAINS SQL: Only SQL that does not read or modify data is allowed.<br><b>M</b> MODIFIES SQL DATA: All SQL is allowed, including SQL that reads or modifies data.<br><b>N</b> NO SQL: SQL is not allowed.<br><b>R</b> READS SQL DATA: Only SQL that reads data is allowed.<br><b>blank</b> Not applicable.                                                                                                                                                                                                                                                                                 | G   |
| DBINFO          | CHAR(1)<br>NOT NULL  | The DBINFO option of an external function or stored procedure:<br><br><b>N</b> No, the DBINFO parameter will not be passed to the external function or stored procedure.<br><b>Y</b> Yes, the DBINFO parameter will be passed to the external function or stored procedure.<br><b>blank</b> ORIGIN is not 'E' .                                                                                                                                                                                                                                                                                                                                                                                             | G   |
| #               |                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |     |
| STAYRESIDENT    | CHAR(1)<br>NOT NULL  | The STAYRESIDENT option of the routine, which determines whether the routine is to be deleted from memory when the routine ends.<br><br><b>N</b> The load module is to be deleted from memory after the routine terminates.<br><b>Y</b> The load module is to remain resident in memory after the routine terminates.<br><b>blank</b> ORIGIN is not 'E' .                                                                                                                                                                                                                                                                                                                                                   | G   |
| ASUTIME         | INTEGER<br>NOT NULL  | Number of CPU service units permitted for any single invocation of this routine. If ASUTIME is zero, the number of CPU service units is unlimited.<br><br>If a routine consumes more CPU service units than the ASUTIME value allows, DB2 cancels the routine.                                                                                                                                                                                                                                                                                                                                                                                                                                              | G   |
| #               |                      | The column is blank if ROUTINETYPE='F' and ORIGIN is not 'E' .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |     |
| WLM_ENVIRONMENT | CHAR(18)<br>NOT NULL | Name of the WLM environment to be used to run this routine.<br><br>If the ROUTINETYPE = 'P', the value might be blank. Blank causes the stored procedure to be run in the DB2 stored procedure address space.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               | G   |
| #               |                      | The column is blank if ROUTINETYPE='F' and ORIGIN is not 'E' .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |     |

| Column name        | Data type             | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       | Use |
|--------------------|-----------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| WLM_ENV_FOR_NESTED | CHAR(1)<br>NOT NULL   | For nested routine calls, indicates whether the address space of the calling stored procedure or user-defined function is used to run the nested stored procedure or user-defined function:<br><br><b>N</b> The nested stored procedure or user-defined function runs in an address space other than the specified WLM environment if the calling stored procedure or user-defined function is not running in the specified WLM environment. 'WLM ENVIRONMENT name' was specified.<br><br><b>Y</b> The nested stored procedure or user-defined function runs in the environment used by the calling stored procedure or user-defined function. 'WLM ENVIRONMENT(name,*)' was specified.<br><br><b>blank</b> If ROUTINETYPE = 'F' and ORIGIN is not 'E'. WLM_ENVIRONMENT is blank. | G   |
| PROGRAM_TYPE       | CHAR(1)<br>NOT NULL   | Indicates whether the routine runs as a Language Environment main routine or a subroutine:<br><br><b>M</b> The routine runs as a main routine.<br><b>S</b> The routine runs as a subroutine.<br><b>blank</b> ORIGIN is not 'E'.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   | G   |
| EXTERNAL_SECURITY  | CHAR(1)<br>NOT NULL   | Specifies the authorization ID to be used if the routine accesses resources protected by an external security product:<br><br><b>D</b> DB2 - The authorization ID associated with the WLM-established stored procedure address space.<br><b>U</b> USER - The authorization ID of the SQL user that invoked the routine.<br><b>C</b> DEFINER - The authorization ID of the owner of the routine.<br><br><b>blank</b> ORIGIN is not 'E'.                                                                                                                                                                                                                                                                                                                                            | G   |
| COMMIT_ON_RETURN   | CHAR(1)<br>NOT NULL   | If ROUTINETYPE = 'P', whether the transaction is always to be committed immediately on successful return (non-negative SQLCODE) from this stored procedure:<br><br><b>N</b> The unit of work is to continue.<br><b>Y</b> The unit of work is to be committed immediately.<br>If ROUTINETYPE = 'F', the value is blank.                                                                                                                                                                                                                                                                                                                                                                                                                                                            | G   |
| RESULT_SETS        | SMALLINT<br>NOT NULL  | If ROUTINETYPE = 'P', the maximum number of ad hoc result sets that this stored procedure can return.<br><br>If no ad hoc result exists or ROUTINETYPE = 'F', the value is zero.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  | G   |
| LOBCOLUMNS         | SMALLINT<br>NOT NULL  | If ORIGIN = 'E' or 'Q', the number of LOB columns found in the parameter list for this user-defined function.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     | I   |
| #                  |                       | If no LOB columns are found in the parameter list or ORIGIN is not 'E' or not 'Q', the value is 0.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |     |
| CREATEDTS          | TIMESTAMP<br>NOT NULL | Time when the CREATE statement was executed for this routine.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     | G   |
| ALTEREDTS          | TIMESTAMP<br>NOT NULL | Time when the last ALTER statement was executed for this routine.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 | G   |
| IBMREQD            | CHAR(1)<br>NOT NULL   | A value of Y indicates that the row came from the basic machine-readable material (MRM) tape. For all other values, see "Release dependency indicators" on page 1005.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             | G   |
| PARM1              | SMALLINT<br>NOT NULL  | Internal use only                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 | I   |
| PARM2              | SMALLINT<br>NOT NULL  | Internal use only                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 | I   |
| PARM3              | SMALLINT<br>NOT NULL  | Internal use only                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 | I   |
| PARM4              | SMALLINT<br>NOT NULL  | Internal use only                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 | I   |

## SYSIBM.SYSROUTINES

| Column name   | Data type                            | Description                                                                                                             | Use |
|---------------|--------------------------------------|-------------------------------------------------------------------------------------------------------------------------|-----|
| PARM5         | SMALLINT<br>NOT NULL                 | Internal use only                                                                                                       | I   |
| PARM6         | SMALLINT<br>NOT NULL                 | Internal use only                                                                                                       | I   |
| PARM7         | SMALLINT<br>NOT NULL                 | Internal use only                                                                                                       | I   |
| PARM8         | SMALLINT<br>NOT NULL                 | Internal use only                                                                                                       | I   |
| PARM9         | SMALLINT<br>NOT NULL                 | Internal use only                                                                                                       | I   |
| PARM10        | SMALLINT<br>NOT NULL                 | Internal use only                                                                                                       | I   |
| PARM11        | SMALLINT<br>NOT NULL                 | Internal use only                                                                                                       | I   |
| PARM12        | SMALLINT<br>NOT NULL                 | Internal use only                                                                                                       | I   |
| PARM13        | SMALLINT<br>NOT NULL                 | Internal use only                                                                                                       | I   |
| PARM14        | SMALLINT<br>NOT NULL                 | Internal use only                                                                                                       | I   |
| PARM15        | SMALLINT<br>NOT NULL                 | Internal use only                                                                                                       | I   |
| PARM16        | SMALLINT<br>NOT NULL                 | Internal use only                                                                                                       | I   |
| PARM17        | SMALLINT<br>NOT NULL                 | Internal use only                                                                                                       | I   |
| PARM18        | SMALLINT<br>NOT NULL                 | Internal use only                                                                                                       | I   |
| PARM19        | SMALLINT<br>NOT NULL                 | Internal use only                                                                                                       | I   |
| PARM20        | SMALLINT<br>NOT NULL                 | Internal use only                                                                                                       | I   |
| PARM21        | SMALLINT<br>NOT NULL                 | Internal use only                                                                                                       | I   |
| PARM22        | SMALLINT<br>NOT NULL                 | Internal use only                                                                                                       | I   |
| PARM23        | SMALLINT<br>NOT NULL                 | Internal use only                                                                                                       | I   |
| PARM24        | SMALLINT<br>NOT NULL                 | Internal use only                                                                                                       | I   |
| PARM25        | SMALLINT<br>NOT NULL                 | Internal use only                                                                                                       | I   |
| PARM26        | SMALLINT<br>NOT NULL                 | Internal use only                                                                                                       | I   |
| PARM27        | SMALLINT<br>NOT NULL                 | Internal use only                                                                                                       | I   |
| PARM28        | SMALLINT<br>NOT NULL                 | Internal use only                                                                                                       | I   |
| PARM29        | SMALLINT<br>NOT NULL                 | Internal use only                                                                                                       | I   |
| PARM30        | SMALLINT<br>NOT NULL                 | Internal use only                                                                                                       | I   |
| IOS_PER_INVOC | FLOAT<br>NOT NULL WITH<br>DEFAULT -1 | Estimated number of I/Os that required to execute the routine.<br>The value is -1 if the estimated number is not known. | S   |

| Column name     | Data type                                 | Description                                                                                                                                                                          | Use |
|-----------------|-------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| INSTS_PER_INVOC | FLOAT<br>NOT NULL WITH<br>DEFAULT -1      | Estimated number of machine instructions that required to execute the routine. The value is -1 if the estimated number is not known.                                                 | S   |
| INITIAL_IOS     | FLOAT<br>NOT NULL WITH<br>DEFAULT -1      | Estimated number of I/O's that are performed the first time or the last time the routine is invoked. The value is -1 if the estimated number is not known.                           | S   |
| INITIAL_INSTS   | FLOAT<br>NOT NULL WITH<br>DEFAULT -1      | Estimated number of machine instructions that are performed the first time or the last time the routine is invoked. The value is -1 if the estimated number is not known.            | S   |
| CARDINALITY     | FLOAT<br>NOT NULL WITH<br>DEFAULT -1      | The predicted cardinality of the routine, -1 to trigger DB2's use of the default value (10,000).                                                                                     | S   |
| RESULT_COLS     | SMALLINT<br>NOT NULL<br>DEFAULT 1         | For a table function, the number of columns in the result table. Otherwise, the value is 1.                                                                                          | S   |
| EXTERNAL_NAME   | VARCHAR(254)<br>NOT NULL                  | The path/module/function that DB2 should load to execute the routine. The column is blank if the ORIGIN is not 'E'.                                                                  | G   |
| PARM_SIGNATURE  | VARCHAR(150)<br>NOT NULL<br>FOR BIT DATA  | Internal use only                                                                                                                                                                    | I   |
| RUNOPTS         | VARCHAR(254)<br>NOT NULL                  | The Language Environment run-time options to be used for this routine. An empty string indicates that the installation default Language Environment run-time options are to be used. | G   |
| #               |                                           | The column is blank if ROUTINETYPE='F' and ORIGIN is not 'E'.                                                                                                                        |     |
| #               |                                           |                                                                                                                                                                                      |     |
| REMARKS         | VARCHAR(254)<br>NOT NULL                  | A character string provided by the user with the COMMENT ON statement.                                                                                                               | G   |
| JAVA_SIGNATURE  | VARCHAR(1024)<br>NOT NULL WITH<br>DEFAULT | The signature of the jar file.                                                                                                                                                       | G   |
| #               |                                           | The column is blank if ROUTINETYPE='F' and ORIGIN is not 'E'.                                                                                                                        |     |
| #               |                                           |                                                                                                                                                                                      |     |
| CLASS           | VARCHAR(128)<br>NOT NULL WITH<br>DEFAULT  | The class name contained in the jar file.                                                                                                                                            | G   |
| #               |                                           | The column is blank if ROUTINETYPE='F' and ORIGIN is not 'E'.                                                                                                                        |     |
| #               |                                           |                                                                                                                                                                                      |     |
| JARSCHEMA       | CHAR(8)<br>NOT NULL WITH<br>DEFAULT       | The schema of the jar file.                                                                                                                                                          | G   |
| #               |                                           | The column is blank if ROUTINETYPE='F' and ORIGIN is not 'E'.                                                                                                                        |     |
| #               |                                           |                                                                                                                                                                                      |     |
| JAR_ID          | CHAR(18)<br>NOT NULL WITH<br>DEFAULT      | The name of the jar file.                                                                                                                                                            | G   |
| #               |                                           | The column is blank if ROUTINETYPE='F' and ORIGIN is not 'E'.                                                                                                                        |     |
| #               |                                           |                                                                                                                                                                                      |     |
| SPECIAL_REGS    | CHAR(1)<br>NOT NULL WITH<br>DEFAULT 'I'   | The SPECIAL REGISTER option for a routine.<br><br>I      INHERIT SPECIAL REGISTERS<br>D      DEFAULT SPECIAL REGISTERS                                                               | G   |
| #               |                                           | The column is blank if ROUTINETYPE='F' and ORIGIN is not 'E' or not 'Q'.                                                                                                             |     |
| #               |                                           |                                                                                                                                                                                      |     |

## SYSIBM.SYSROUTINES\_OPTS

### SYSIBM.SYSROUTINES\_OPTS table

Contains a row for each generated routine, such as one created by the DB2 Stored Procedure Builder tool, that records the build options for the routine. Rows in this table can be inserted, updated, and deleted.

| Column name     | Data type                                 | Description                                                                                                                                                           | Use |
|-----------------|-------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| SCHEMA          | CHAR(8)<br>NOT NULL                       | Schema of the routine.                                                                                                                                                | G   |
| ROUTINENAME     | CHAR(18)<br>NOT NULL                      | Name of the routine.                                                                                                                                                  | G   |
| BUILDDATE       | DATE<br>NOT NULL WITH<br>DEFAULT          | Date the routine was built                                                                                                                                            | G   |
| BUILDTIME       | TIME<br>NOT NULL WITH<br>DEFAULT          | Time the routine was built                                                                                                                                            | G   |
| BUILDSTATUS     | CHAR(1)<br>NOT NULL WITH<br>DEFAULT 'C'   | Whether this version of the routine's options is the current version                                                                                                  | G   |
| BUILDSchema     | CHAR(8)<br>NOT NULL                       | Schema name for BUILDNAME.                                                                                                                                            | G   |
| BUILDNAME       | CHAR(18)<br>NOT NULL                      | Procedure used to create the routine.                                                                                                                                 | G   |
| BUILDOwner      | CHAR(8)<br>NOT NULL                       | Authorization ID used to create the routine.                                                                                                                          | G   |
| IBMREQD         | CHAR(1)<br>NOT NULL WITH<br>DEFAULT 'N'   | A value of Y indicates that the row came from the basic machine-readable material (MRM) tape. For all other values, see "Release dependency indicators" on page 1005. | G   |
| PRECOMPILE_OPTS | VARCHAR(255)<br>NOT NULL WITH<br>DEFAULT  | Precompiler options used to build the routine.                                                                                                                        | G   |
| COMPILE_OPTS    | VARCHAR(255)<br>NOT NULL WITH<br>DEFAULT  | Compiler options used to build the routine.                                                                                                                           | G   |
| PRELINK_OPTS    | VARCHAR(255)<br>NOT NULL WITH<br>DEFAULT  | Prelink-edit options used to build the routine.                                                                                                                       | G   |
| LINK_OPTS       | VARCHAR(255)<br>NOT NULL WITH<br>DEFAULT  | Link-edit options used to build the routine.                                                                                                                          | G   |
| BIND_OPTS       | VARCHAR(1024)<br>NOT NULL WITH<br>DEFAULT | Bind options used to build the routine.                                                                                                                               | G   |
| SOURCEDSN       | VARCHAR(255)<br>NOT NULL WITH<br>DEFAULT  | Name of the source data set.                                                                                                                                          | G   |

---

## SYSIBM.SYSROUTINES\_SRC table

Contains source for generated routines, such as those created by the DB2 Stored Procedure Builder tool. Rows in this table can be inserted, updated, and deleted.

| Column name | Data type                               | Description                                                                                                                                                           | Use |
|-------------|-----------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| SCHEMA      | CHAR(8)<br>NOT NULL                     | Schema of the routine.                                                                                                                                                | G   |
| ROUTINENAME | CHAR(18)<br>NOT NULL                    | Name of the routine.                                                                                                                                                  | G   |
| BUILDDATE   | DATE<br>NOT NULL WITH<br>DEFAULT        | Date the routine was built                                                                                                                                            | G   |
| BUILDTIME   | TIME<br>NOT NULL WITH<br>DEFAULT        | Time the routine was built                                                                                                                                            | G   |
| BUILDSTATUS | CHAR(1)<br>NOT NULL WITH<br>DEFAULT 'C' | Whether this version of the routine's source is the current version                                                                                                   | G   |
| SEQNO       | INTEGER<br>NOT NULL                     | Number of the source statement piece in CREATESTMT.                                                                                                                   | G   |
| IBMRREQD    | CHAR(1)<br>NOT NULL WITH<br>DEFAULT 'N' | A value of Y indicates that the row came from the basic machine-readable material (MRM) tape. For all other values, see "Release dependency indicators" on page 1005. | G   |
| CREATESTMT  | VARCHAR(3800)<br>NOT NULL               | Routine source statement.                                                                                                                                             | G   |

---

## SYSIBM.SYSSCHEMAAUTH

### SYSIBM.SYSSCHEMAAUTH table

Contains one or more rows for each user that is granted a privilege on a particular schema in the database.

| Column name  | Data type             | Description                                                                                                                                                                                                                                                                                                                                                                        | Use |
|--------------|-----------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| GRANTOR      | CHAR(8)<br>NOT NULL   | Authorization ID of the user who granted the privileges or SYSADM.                                                                                                                                                                                                                                                                                                                 | G   |
| GRANTEE      | CHAR(8)<br>NOT NULL   | Authorization ID of the user or group who holds the privileges. Can also be PUBLIC for a grant to PUBLIC.                                                                                                                                                                                                                                                                          | G   |
| SCHEMANAME   | CHAR(8)<br>NOT NULL   | Name of the schema or ** for all schemas.                                                                                                                                                                                                                                                                                                                                          | G   |
| AUTHHOWGOT   | CHAR(1)<br>NOT NULL   | Authorization level of the user from whom the privileges were received. This authorization level is not necessarily the highest authorization level of the grantor.<br><br>This field is also used to indicate that the privilege was held on all schemas by the grantor.<br><b>1</b> Grantor had privilege on all schemas at time of grant<br><b>L</b> SYSCTRL<br><b>S</b> SYSADM | G   |
| CREATEINAUTH | CHAR(1)<br>NOT NULL   | Indicates whether grantee holds CREATEIN privilege on the schema:<br><br><b>blank</b> Privilege is not held<br><b>G</b> Privilege is held with the GRANT option<br><b>Y</b> Privilege is held without the GRANT option                                                                                                                                                             | G   |
| ALTERINAUTH  | CHAR(1)<br>NOT NULL   | Indicates whether grantee holds ALTERIN privilege on the schema:<br><br><b>blank</b> Privilege is not held<br><b>G</b> Privilege is held with the GRANT option<br><b>Y</b> Privilege is held without the GRANT option                                                                                                                                                              | G   |
| DROPINAUTH   | CHAR(1)<br>NOT NULL   | Indicates whether grantee holds DROPIN privilege on the schema:<br><br><b>blank</b> Privilege is not held<br><b>G</b> Privilege is held with the GRANT option<br><b>Y</b> Privilege is held without the GRANT option                                                                                                                                                               | G   |
| GRANTEDTS    | TIMESTAMP<br>NOT NULL | Time when the GRANT statement was executed.                                                                                                                                                                                                                                                                                                                                        | G   |
| IBMREQD      | CHAR(1)<br>NOT NULL   | A value of Y indicates that the row came from the basic machine-readable material (MRM) tape. For all other values, see "Release dependency indicators" on page 1005.                                                                                                                                                                                                              | G   |

**SYSIBM.SYSSEQUENCES table**

Contains one row for each identity column.

| Column name    | Data type                 | Description                                                                                                                                                                                  | Use |
|----------------|---------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| SCHEMA         | CHAR(8)<br>NOT NULL       | The value of TBCREATOR from the SYSCOLUMNS entry for the identity column.                                                                                                                    | G   |
| OWNER          | CHAR(8)<br>NOT NULL       | The value of TBCREATOR from the SYSCOLUMNS entry for the identity column.                                                                                                                    | G   |
| NAME           | CHAR(18)<br>NOT NULL      | Name that DB2 generated for the identity column.                                                                                                                                             | G   |
| SEQTYPE        | CHAR(1)<br>NOT NULL       | Type of entry:<br>I For an identity column                                                                                                                                                   | G   |
| SEQUENCEID     | INTEGER<br>NOT NULL       | Internal identifier of the identity column.                                                                                                                                                  | G   |
| CREATEDBY      | CHAR(8)<br>NOT NULL       | The authorization ID under which the identity column was created.                                                                                                                            | G   |
| INCREMENT      | DECIMAL(31,0)<br>NOT NULL | Increment value (positive or negative, within INTEGER scope).                                                                                                                                | G   |
| START          | DECIMAL(31,0)<br>NOT NULL | Start value.                                                                                                                                                                                 | G   |
| MAXVALUE       | DECIMAL(31,0)<br>NOT NULL | Maximum value allowed for the identity column.                                                                                                                                               | G   |
| MINVALUE       | DECIMAL(31,0)<br>NOT NULL | Minimum value allowed for the identity column.                                                                                                                                               | G   |
| CYCLE          | CHAR(1)<br>NOT NULL       | Whether values for the identity column are wrapped (values continue to be generated for the column after the maximum or minimum value is reached):<br>N No<br>Y Yes                          | G   |
| CACHE          | INTEGER<br>NOT NULL       | Number of identity column values to preallocate in memory for faster access. A value of 0 indicates that values are not to be preallocated.                                                  | G   |
| ORDER          | CHAR(1)<br>NOT NULL       | The value is always 'N' for an identity column.                                                                                                                                              | G   |
| DATATYPEID     | INTEGER<br>NOT NULL       | For a built-in data type, the internal ID of the built-in type. For a distinct type, the internal ID of the distinct type.                                                                   | S   |
| SOURCETYPEID   | INTEGER<br>NOT NULL       | For a built-in data type, 0. For a distinct type, the internal ID of the built-in data type upon which the distinct type is sourced.                                                         | S   |
| CREATEDTS      | TIMESTAMP<br>NOT NULL     | Timestamp when the identity column was created.                                                                                                                                              | G   |
| ALTEREDTS      | TIMESTAMP<br>NOT NULL     | Timestamp when the identity column was created.                                                                                                                                              | G   |
| MAXASSIGNEDVAL | DECIMAL(31,0)             | Last possible assigned value. Initialized to null when the sequence object is created. Updated each time the next chunk of <i>n</i> values is cached, where <i>n</i> is the value for CACHE. | G   |
| IBMREQD        | CHAR(1)<br>NOT NULL       | A value of Y indicates that the row came from the basic machine-readable material (MRM) tape. For all other values, see "Release dependency indicators" on page 1005.                        | G   |
| REMARKS        | VARCHAR(254)<br>NOT NULL  | The value is always blank for an identity column.                                                                                                                                            | G   |

**SYSIBM.SYSSEQUENCESDEP table**

Records the dependencies of identity columns on tables.

| Column name | Data type            | Description                                                                                                                                                           | Use |
|-------------|----------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| BSEQUENCEID | INTEGER<br>NOT NULL  | Internal identifier of the identity column in SYSSEQUENCES.                                                                                                           | G   |
| DCREATOR    | CHAR(8)<br>NOT NULL  | Authorization ID of the owner of the table that contains the identity column.                                                                                         | G   |
| IBMREQD     | CHAR(1)<br>NOT NULL  | A value of Y indicates that the row came from the basic machine-readable material (MRM) tape. For all other values, see "Release dependency indicators" on page 1005. | G   |
| DNAME       | CHAR(18)<br>NOT NULL | Name of the table containing the identity column.                                                                                                                     | G   |
| DCOLNAME    | CHAR(18)<br>NOT NULL | Name of the identity column.                                                                                                                                          | G   |

---

## SYSIBM.SYSSTMT table

Contains one or more rows for each SQL statement of each DBRM.

| Column name | Data type                           | Description                                                                                                                                                                                                                                                                                                                                                                                                                               | Use |
|-------------|-------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| NAME        | CHAR(8)<br>NOT NULL                 | Name of the DBRM.                                                                                                                                                                                                                                                                                                                                                                                                                         | G   |
| PLNAME      | CHAR(8)<br>NOT NULL                 | Name of the application plan.                                                                                                                                                                                                                                                                                                                                                                                                             | G   |
| PLCREATOR   | CHAR(8)<br>NOT NULL                 | Authorization ID of the owner of the application plan.                                                                                                                                                                                                                                                                                                                                                                                    | G   |
| SEQNO       | SMALLINT<br>NOT NULL                | Sequence number of this row with respect to a statement of the DBRM <sup>53</sup> . The numbering starts with zero.                                                                                                                                                                                                                                                                                                                       | G   |
| STMTNO      | SMALLINT<br>NOT NULL                | The statement number of the statement in the source program. A statement number greater than 32767 is stored as zero (if the value is zero, see STMTNOI for the statement number). <sup>53</sup>                                                                                                                                                                                                                                          | G   |
| SECTNO      | SMALLINT<br>NOT NULL                | The section number of the statement. <sup>53</sup>                                                                                                                                                                                                                                                                                                                                                                                        | G   |
| IBMREQD     | CHAR(1)<br>NOT NULL                 | A value of Y indicates that the row came from the basic machine-readable material (MRM) tape. For all other values, see "Release dependency indicators" on page 1005.                                                                                                                                                                                                                                                                     | G   |
| TEXT        | VARCHAR(254)<br>NOT NULL            | Text or portion of the text of the SQL statement.                                                                                                                                                                                                                                                                                                                                                                                         | S   |
| ISOLATION   | CHAR(1)<br>NOT NULL WITH<br>DEFAULT | Isolation level for the SQL statement:<br><b>R</b> RR (repeatable read)<br><b>T</b> RS (read stability)<br><b>S</b> CS (cursor stability)<br><b>U</b> UR (uncommitted read)<br><b>L</b> KEEP UPDATE LOCKS for an RS isolation<br><b>X</b> KEEP UPDATE LOCKS for an RR isolation<br><b>blank</b> The WITH clause was not specified on this statement.<br>The isolation level is recorded in SYSPACKAGE.ISOLATION and in SYSPLAN.ISOLATION. | G   |

53. Rows in which the values of SEQNO, STMTNO, and SECTNO are zero are for internal use.

## SYSIBM.SYSSTMT

| Column name | Data type                           | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    | Use |
|-------------|-------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| STATUS      | CHAR(1)<br>NOT NULL WITH<br>DEFAULT | Status of binding the statement:<br><br><b>A</b> Distributed - statement uses DB2 private protocol access. The statement will be parsed and executed at the server using defaults for input variables during access path selection.<br><b>B</b> Distributed - statement uses DB2 private protocol access. The statement will be parsed and executed at the server using values for input variables during access path selection.<br><b>C</b> Compiled - statement was bound successfully using defaults for input variables during access path selection.<br><b>D</b> Distributed - statement references a remote object using DB2 private protocol access (a three-part name), but DB2 will implicitly use DRDA access instead because the statement was bound with bind option DBPROTOCOL(DRDA). This option allows the use of three-part names with DRDA access, but it requires that the package be bound at the target remote site.<br><b>E</b> Explain - statement is an SQL EXPLAIN statement. The explain is done at bind time using defaults for input variables during access path selection.<br><b>F</b> Parsed - statement did not bind successfully and VALIDATE(RUN) was used. The statement will be rebound at execution time using values for input variables during access path selection.<br><b>G</b> Compiled - statement bound successfully, but REOPT is specified. The statement will be rebound at execution time using values for input variables during access path selection.<br><b>H</b> Parsed - statement is either a data definition statement or a statement that did not bind successfully and VALIDATE(RUN) was used. The statement will be rebound at execution time using defaults for input variables during access path selection. Data manipulation statements use defaults for input variables during access path selection.<br><b>I</b> Indefinite - statement is dynamic. The statement will be bound at execution time using defaults for input variables during access path selection.<br><b>J</b> Indefinite - statement is dynamic. The statement will be bound at execution time using values for input variables during access path selection.<br><b>K</b> Control - CALL statement.<br><b>L</b> Bad - the statement has some allowable error. The bind continues but the statement cannot be executed.<br><b>M</b> Parsed - statement references a table that is qualified with SESSION and was not bound because the table reference could be for a declared temporary table that will not be defined until the package or plan is run. The SQL statement will be rebound at execution time using values for input variables during access path selection.<br><b>blank</b> The statement is non-executable, or was bound in a DB2 release prior to Version 5. | S   |
| ACCESSPATH  | CHAR(1)<br>NOT NULL WITH<br>DEFAULT | For static statements, indicates if the access path for the statement is based on user-specified optimization hints. A value of 'H' indicates that optimization hints were used. A blank value indicates that the access path was determined without the use of optimization hints, or that there is no access path associated with the statement.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             | G   |
|             |                                     | For dynamic statements, the value is blank.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |     |
| # STMTNOI   | INTEGER<br>NOT NULL WITH<br>DEFAULT | If the value of STMTNO is 0, the column contains the statement number of the statement in the source program. If both STMTNO and STMTNOI are 0, the statement number is greater than 32767.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    | G   |

| Column name | Data type                              | Description                                                                                                                                                                                                                                                                                                                                                                                             | Use |
|-------------|----------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| SECTNOI     | INTEGER<br>NOT NULL WITH<br>DEFAULT    | The section number of the statement.                                                                                                                                                                                                                                                                                                                                                                    | G   |
| EXPLAINABLE | CHAR(1)<br>NOT NULL WITH<br>DEFAULT    | Contains one of the following values:<br><br>Y      Indicates that the SQL statement can be used with the EXPLAIN function and may have rows describing its access path in the usid.PLAN_TABLE.<br><br>N      Indicates that the SQL statement does not have any rows describing its access path in the usid.PLAN_TABLE.<br><br>blank    Indicates that the SQL statement was bound prior to Version 7. | G   |
| QUERYNO     | INTEGER<br>NOT NULL WITH<br>DEFAULT -1 | The query number of the SQL statement in the source program. SQL statements bound prior to Version 7 have a default value of -1. Statements bound in Version 7 or later use the value specified on the QUERYNO clause on SELECT, UPDATE, INSERT, DELETE, EXPLAIN, and DECLARE CURSOR statements. If the QUERYNO clause is not specified, the query number is set to the statement number.               | G   |

**SYSIBM.SYSSTOGROUP table**

Contains one row for each storage group.

| Column name | Data type                             | Description                                                                                                                                                                   | Use |
|-------------|---------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| NAME        | CHAR(8)<br>NOT NULL                   | Name of the storage group.                                                                                                                                                    | G   |
| CREATOR     | CHAR(8)<br>NOT NULL                   | Authorization ID of the owner of the storage group.                                                                                                                           | G   |
| VCATNAME    | CHAR(8)<br>NOT NULL                   | Name of the integrated catalog facility catalog.                                                                                                                              | G   |
|             |                                       | Not used                                                                                                                                                                      | N   |
| SPACE       | INTEGER<br>NOT NULL                   | Number of kilobytes of DASD storage allocated to the storage group as determined by the last execution of the STOSPACE utility.                                               | G   |
| SPCDATE     | CHAR(5)<br>NOT NULL                   | Date when the SPACE column was last updated, in the form <i>yyddd</i> .                                                                                                       | G   |
| IBMREQD     | CHAR(1)<br>NOT NULL                   | A value of Y indicates that the row came from the basic machine-readable material (MRM) tape. For all other values, see "Release dependency indicators" on page 1005.         | G   |
| CREATEDBY   | CHAR(8)<br>NOT NULL WITH<br>DEFAULT   | Primary authorization ID of the user who created the storage group.                                                                                                           | G   |
| STATSTIME   | TIMESTAMP<br>NOT NULL WITH<br>DEFAULT | If the STOSPACE utility was executed for the storage group, date and time when STOSPACE was last executed.                                                                    | G   |
| CREATEDTS   | TIMESTAMP<br>NOT NULL WITH<br>DEFAULT | Time when the CREATE statement was executed for the storage group.                                                                                                            | G   |
| ALTEREDTS   | TIMESTAMP<br>NOT NULL WITH<br>DEFAULT | Time when the most recent ALTER STOGROUP statement was executed for the storage group. If no ALTER STOGROUP statement has been applied, ALTEREDTS has the value of CREATEDTS. | G   |

## SYSIBM.SYSSTRINGS table

Contains information about character conversion. Each row describes a conversion from one coded character set to another.

If OS/390 Version 2 Release 9 (or a subsequent release) is installed, refer to OS/390 C/C++ Programming Guide for information on the additional conversions that are supported.

| Column name | Data type                                                | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                 | Use |
|-------------|----------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| INCCSID     | INTEGER<br>NOT NULL                                      | The source CCSID for the character conversion represented by this row.                                                                                                                                                                                                                                                                                                                                                                                                      | G   |
| OUTCCSID    | INTEGER<br>NOT NULL                                      | The target CCSID for the character conversion represented by this row.                                                                                                                                                                                                                                                                                                                                                                                                      | G   |
| TRANSTYPE   | CHAR(2)<br>NOT NULL                                      | Indicates the nature of the conversion. Values can be:<br><b>GG</b> GRAPHIC to GRAPHIC<br><b>MM</b> EBCDIC MIXED to EBCDIC MIXED<br><b>MS</b> EBCDIC MIXED to SBCS<br><b>PM</b> ASCII MIXED to EBCDIC MIXED<br><b>PS</b> ASCII MIXED to SBCS<br><b>SM</b> SBCS to EBCDIC MIXED<br><b>SS</b> SBCS to SBCS<br><b>MP</b> EBCDIC MIXED to ASCII MIXED<br><b>PP</b> ASCII MIXED to ASCII MIXED<br><b>SP</b> SBCS to ASCII MIXED                                                  | G   |
| ERRORBYTE   | CHAR(1)<br>FOR BIT DATA<br>Nulls allowed                 | The byte used in the conversion table as an error byte. Null indicates the absence of an error byte.                                                                                                                                                                                                                                                                                                                                                                        | S   |
| SUBBYTE     | CHAR(1)<br>FOR BIT DATA<br>Nulls allowed                 | The byte used in the conversion table as a substitution character. Null indicates the absence of a substitution character.                                                                                                                                                                                                                                                                                                                                                  | S   |
| TRANSPROC   | CHAR(8)<br>NOT NULL WITH<br>DEFAULT                      | The name of a module or blanks. If IBMREQD is 'N', a nonblank value is the name of a conversion procedure provided by the user. If IBMREQD is 'Y', a nonblank value is the name of a DB2 module that contains DBCS conversion tables. The first five characters of the name of a user-provided conversion procedure must not be 'DSNXV'; these characters are used to distinguish user-provided conversion procedures from DB2 modules that contain DBCS conversion tables. | G   |
| IBMREQD     | CHAR(1)<br>NOT NULL                                      | A value of Y indicates that the row came from the basic machine-readable material (MRM) tape. For all other values, see "Release dependency indicators" on page 1005.                                                                                                                                                                                                                                                                                                       | G   |
| TRANSTAB    | VARCHAR(256)<br>FOR BIT DATA<br>NOT NULL WITH<br>DEFAULT | Either a conversion table or an empty string.                                                                                                                                                                                                                                                                                                                                                                                                                               | S   |

Each row in the table must have a unique combination of values for its INCCSID, OUTCCSID, and IBMREQD columns. Rows for which the value of IBMREQD is N can be deleted, inserted, and updated subject to this uniqueness constraint and to the constraints imposed by a VALIDPROC defined on the table. An inserted row could have values for the INCCSID and OUTCCSID columns that match those of a row for which the value of IBMREQD is Y. DB2 then uses the information in the inserted row instead of the information in the IBM-supplied row. Rows for which the value of IBMREQD is Y cannot be deleted, inserted, or updated. For information about the use of inserted rows for character conversion, see Appendix C of *DB2 Installation Guide*.

## SYSIBM.SYSSTRINGS

| DB2 has three methods for character conversions and applies them in the following  
| order:

1. Conversions specified by the various combinations of the INCCSID and OUTCCSID columns in the SYSIBM.SYSSTRINGS catalog table
2. Conversions provided by OS/390 support for Unicode. The use of OS/390 support for Unicode requires OS/390 Version 2 Release 8 or later. For further information, see z/OS Support for Unicode: Using Conversion Services.
3. Conversions provided by the Language Environment. The use of Language Environment capabilities requires OS/390 Version 2 Release 9 or later.

| If none of these methods can be used for a particular character conversion, DB2  
| returns an error.

---

## SYSIBM.SYSSYNONYMS table

Contains one row for each synonym of a table or view.

| Column name | Data type                             | Description                                                                                                                                                           | Use |
|-------------|---------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| NAME        | VARCHAR(18)<br>NOT NULL               | Synonym for the table or view.                                                                                                                                        | G   |
| CREATOR     | CHAR(8)<br>NOT NULL                   | Authorization ID of the owner of the synonym.                                                                                                                         | G   |
| TBNAME      | VARCHAR(18)<br>NOT NULL               | Name of the table or view.                                                                                                                                            | G   |
| TBCREATOR   | CHAR(8)<br>NOT NULL                   | Authorization ID of the owner of the table or view.                                                                                                                   | G   |
| IBMREQD     | CHAR(1)<br>NOT NULL                   | A value of Y indicates that the row came from the basic machine-readable material (MRM) tape. For all other values, see "Release dependency indicators" on page 1005. | G   |
| CREATEDBY   | CHAR(8)<br>NOT NULL WITH<br>DEFAULT   | Primary authorization ID of the user who created the synonym.                                                                                                         | G   |
| CREATEDTS   | TIMESTAMP<br>NOT NULL WITH<br>DEFAULT | Time when the CREATE statement was executed for the synonym. The value is '0001-01-01-00.00.000000' for synonyms created in a DB2 release prior to Version 5.         | G   |

## SYSIBM.SYSTABAUTH

### SYSIBM.SYSTABAUTH table

Records the privileges that users hold on tables and views.

| Column name | Data type               | Description                                                                                                                                                                                                                                                                                                                                                                                         | Use |
|-------------|-------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| GRANTOR     | CHAR(8)<br>NOT NULL     | Authorization ID of the user who granted the privileges. Could also be PUBLIC, or PUBLIC followed by an asterisk. <sup>54</sup>                                                                                                                                                                                                                                                                     | G   |
| GRANTEE     | CHAR(8)<br>NOT NULL     | Authorization ID of the user who holds the privileges or the name of an application plan or package that uses the privileges. PUBLIC for a grant to PUBLIC. PUBLIC followed by an asterisk for a grant to PUBLIC AT ALL LOCATIONS.                                                                                                                                                                  | G   |
| GRANTEETYPE | CHAR(1)<br>NOT NULL     | Type of grantee:<br><b>blank</b> An authorization ID<br><b>P</b> An application plan or a package. The grantee is a package if COLLID is not blank.                                                                                                                                                                                                                                                 | G   |
| DBNAME      | CHAR(8)<br>NOT NULL     | If the privileges were received from a user with DBADM, DBCTRL, or DBMAINT authority, DBNAME is the name of the database on which the GRANTOR has that authority. Otherwise, DBNAME is blank.                                                                                                                                                                                                       | G   |
| SCREATOR    | CHAR(8)<br>NOT NULL     | If the row of SYSIBM.SYSTABAUTH was created as a result of a CREATE VIEW statement, SCREATOR is the authorization ID of the owner of a table or view referred to in the CREATE VIEW statement. Otherwise, SCREATOR is the same as TCREATOR.                                                                                                                                                         | G   |
| STNAME      | VARCHAR(18)<br>NOT NULL | If the row of SYSIBM.SYSTABAUTH was created as a result of a CREATE VIEW statement, STNAME is the name of a table or view referred to in the CREATE VIEW statement. Otherwise, STNAME is the same as TTNAME.                                                                                                                                                                                        | G   |
| TCREATOR    | CHAR(8)<br>NOT NULL     | Authorization ID of the owner of the table or view.                                                                                                                                                                                                                                                                                                                                                 | G   |
| TTNAME      | VARCHAR(18)<br>NOT NULL | Name of the table or view.                                                                                                                                                                                                                                                                                                                                                                          | G   |
| AUTHHOWGOT  | CHAR(1)<br>NOT NULL     | Authorization level of the user from whom the privileges were received. This authorization level is not necessarily the highest authorization level of the grantor.<br><b>blank</b> Not applicable<br><b>C</b> DBCTL<br><b>D</b> DBADM<br><b>L</b> SYSCTRL<br><b>M</b> DBMAINT<br><b>S</b> SYSADM                                                                                                   | G   |
|             | CHAR(12)<br>NOT NULL    | Internal use only                                                                                                                                                                                                                                                                                                                                                                                   | I   |
|             | CHAR(6)<br>NOT NULL     | Not used.                                                                                                                                                                                                                                                                                                                                                                                           | N   |
|             | CHAR(8)<br>NOT NULL     | Not used.                                                                                                                                                                                                                                                                                                                                                                                           | N   |
| UPDATECOLS  | CHAR(1)<br>NOT NULL     | The value of this column is blank if the value of UPDATEAUTH applies uniformly to all columns of the table or view. The value is an asterisk (*) if the value of UPDATEAUTH applies to some columns but not to others. In this case, rows will exist in SYSIBM.SYSCOLAUTH with matching timestamps and PRIVILEGE = blank. These rows list the columns on which update privileges have been granted. | G   |
| ALTERAUTH   | CHAR(1)<br>NOT NULL     | Whether the GRANTEE can alter the table:<br><b>blank</b> Privilege is not held<br><b>G</b> Privilege is held with the GRANT option<br><b>Y</b> Privilege is held without the GRANT option                                                                                                                                                                                                           | G   |

54. PUBLIC followed by an asterisk (PUBLIC\*) denotes PUBLIC AT ALL LOCATIONS. For the conditions where GRANTOR can be PUBLIC or PUBLIC\*, see Part 3 (Volume 1) of *DB2 Administration Guide*.

| Column name    | Data type                             | Description                                                                                                                                                                                                                                                                                                                                                                                | Use |
|----------------|---------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| DELETEAUTH     | CHAR(1)<br>NOT NULL                   | Whether the GRANTEE can delete rows from the table or view:<br><br><b>blank</b> Privilege is not held<br><b>G</b> Privilege is held with the GRANT option<br><b>Y</b> Privilege is held without the GRANT option                                                                                                                                                                           | G   |
| INDEXAUTH      | CHAR(1)<br>NOT NULL                   | Whether the GRANTEE can create indexes on the table:<br><br><b>blank</b> Privilege is not held<br><b>G</b> Privilege is held with the GRANT option<br><b>Y</b> Privilege is held without the GRANT option                                                                                                                                                                                  | G   |
| INSERTAUTH     | CHAR(1)<br>NOT NULL                   | Whether the GRANTEE can insert rows into the table or view:<br><br><b>blank</b> Privilege is not held<br><b>G</b> Privilege is held with the GRANT option<br><b>Y</b> Privilege is held without the GRANT option                                                                                                                                                                           | G   |
| SELECTAUTH     | CHAR(1)<br>NOT NULL                   | Whether the GRANTEE can select rows from the table or view:<br><br><b>blank</b> Privilege is not held<br><b>G</b> Privilege is held with the GRANT option<br><b>Y</b> Privilege is held without the GRANT option                                                                                                                                                                           | G   |
| UPDATEAUTH     | CHAR(1)<br>NOT NULL                   | Whether the GRANTEE can update rows of the table or view:<br><br><b>blank</b> Privilege is not held<br><b>G</b> Privilege is held with the GRANT option<br><b>Y</b> Privilege is held without the GRANT option                                                                                                                                                                             | G   |
| IBMREQD        | CHAR(1)<br>NOT NULL                   | A value of Y indicates that the row came from the basic machine-readable material (MRM) tape. For all other values, see "Release dependency indicators" on page 1005.                                                                                                                                                                                                                      | G   |
|                | CHAR(16)<br>NOT NULL WITH<br>DEFAULT  | Not used                                                                                                                                                                                                                                                                                                                                                                                   | N   |
|                | CHAR(16)<br>NOT NULL WITH<br>DEFAULT  | Not used                                                                                                                                                                                                                                                                                                                                                                                   | N   |
| COLLID         | CHAR(18)<br>NOT NULL WITH<br>DEFAULT  | If the GRANTEE is a package, its collection name. Otherwise, the value is blank.                                                                                                                                                                                                                                                                                                           | G   |
| CONTOKEN       | CHAR(8) NOT NULL<br>WITH DEFAULT      | If the GRANTEE is a package, the consistency token of the DBRM from which the package was derived. Otherwise, the value is blank.                                                                                                                                                                                                                                                          | S   |
|                | CHAR(1)<br>NOT NULL WITH<br>DEFAULT   | Not used                                                                                                                                                                                                                                                                                                                                                                                   | N   |
| REFERENCESAUTH | CHAR(1)<br>NOT NULL WITH<br>DEFAULT   | Whether the GRANTEE can create or drop referential constraints in which the table is a parent.<br><br><b>blank</b> Privilege is not held<br><b>G</b> Privilege held with the GRANT option<br><b>Y</b> Privilege held without the GRANT option                                                                                                                                              | G   |
| REFCOLS        | CHAR(1)<br>NOT NULL WITH<br>DEFAULT   | The value of this column is blank if the value of REFERENCESAUTH applies uniformly to all columns of the table. The value is an asterisk(*) if the value of REFERENCESAUTH applies to some columns but not to others. In this case, rows will exist in SYSIBM.SYSCOLAUTH with PRIVILEGE = R and matching timestamps that list the columns on which reference privileges have been granted. | G   |
| GRANTEDTS      | TIMESTAMP<br>NOT NULL WITH<br>DEFAULT | Time when the GRANT statement was executed.                                                                                                                                                                                                                                                                                                                                                | G   |
| TRIGGERAUTH    | CHAR(1)<br>NOT NULL WITH<br>DEFAULT   | Whether the GRANTEE can create triggers in which the table is named as the subject table:<br><br><b>blank</b> Privilege is not held<br><b>G</b> Privilege is held with the GRANT option<br><b>Y</b> Privilege is held without the GRANT option                                                                                                                                             | G   |

**SYSIBM.SYSTABCONST table**

Contains one row for each unique constraint (primary key or unique key) created in DB2 for OS/390 and z/OS Version 7 or later.

| Column name | Data type                               | Description                                                                                                                                                           | Use |
|-------------|-----------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| CONSTNAME   | VARCHAR(128)<br>NOT NULL                | Name of the constraint.                                                                                                                                               | G   |
| TBCREATOR   | CHAR(8)<br>NOT NULL                     | Authorization ID of the owner of the table on which the constraint is defined.                                                                                        | G   |
| TBNAME      | VARCHAR(18)<br>NOT NULL                 | Name of the table on which the constraint is defined.                                                                                                                 | G   |
| CREATOR     | CHAR(8)<br>NOT NULL                     | Authorization ID under which the constraint was created.                                                                                                              | G   |
| TYPE        | CHAR(1)<br>NOT NULL                     | Type of constraint:<br><b>P</b> Primary key<br><b>U</b> Unique key                                                                                                    | G   |
| IXOWNER     | CHAR(8)<br>NOT NULL                     | Owner of the index enforcing the constraint or blank if index has not been created yet.                                                                               | G   |
| IXNAME      | VARCHAR(18)<br>NOT NULL                 | Name of the index enforcing the constraint or blank if index has not been created yet.                                                                                | G   |
| CREATEDTS   | TIMESTAMP<br>NOT NULL                   | Time when the statement to create the constraint was executed.                                                                                                        | G   |
| IBMREQD     | CHAR(1)<br>NOT NULL WITH<br>DEFAULT 'N' | A value of Y indicates that the row came from the basic machine-readable material (MRM) tape. For all other values, see "Release dependency indicators" on page 1005. | G   |
| COLCOUNT    | SMALLINT<br>NOT NULL                    | Number of columns in the constraint.                                                                                                                                  | G   |

## SYSIBM.SYSTABLEPART table

Contains one row for each nonpartitioned table space and one row for each partition of a partitioned table space.

| Column name | Data type               | Description                                                                                                                                                                                                                                                                                                                                                                                          | Use |
|-------------|-------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| PARTITION   | SMALLINT<br>NOT NULL    | Partition number; 0 if table space is not partitioned.                                                                                                                                                                                                                                                                                                                                               | G   |
| TSNAME      | CHAR(8)<br>NOT NULL     | Name of the table space.                                                                                                                                                                                                                                                                                                                                                                             | G   |
| DBNAME      | CHAR(8)<br>NOT NULL     | Name of the database that contains the table space.                                                                                                                                                                                                                                                                                                                                                  | G   |
| IXNAME      | VARCHAR(18)<br>NOT NULL | Name of the partitioning index. This column is blank if the table space is not partitioned.                                                                                                                                                                                                                                                                                                          | G   |
| IXCREATOR   | CHAR(8)<br>NOT NULL     | Authorization ID of the owner of the partitioning index. This column is blank if the table space is not partitioned.                                                                                                                                                                                                                                                                                 | G   |
| PQTY        | INTEGER<br>NOT NULL     | Primary space allocation in units of 4KB storage blocks. For user-managed data sets, the value is set to the primary space allocation only if RUNSTATS TABLESPACE with UPDATE(ALL) or UPDATE(SPACE) is executed; otherwise, the value is zero. PQTY is based on a value of PRIQTY in the appropriate CREATE or ALTER TABLESPACE statement. Unlike PQTY, however, PRIQTY asks for space in 1KB units. | G   |
| SQTY        | SMALLINT<br>NOT NULL    | Secondary space allocation in units of 4KB blocks. For user-managed data sets, the value is set to the secondary space allocation only if RUNSTATS TABLESPACE with UPDATE(ALL) or UPDATE(SPACE) is executed; otherwise, the value is zero. SQTY is based on a value of SECQTY in the appropriate CREATE or ALTER TABLESPACE statement. Unlike SQTY, however, SECQTY asks for space in 1KB units.     | G   |
|             |                         | If the value does not fit into the column, the value of the column is 32767. See the description of column SECQTYI.                                                                                                                                                                                                                                                                                  |     |
| STORTYPE    | CHAR(1)<br>NOT NULL     | Type of storage allocation:<br><b>E</b> Explicit (storage group not used)<br><b>I</b> Implicit (storage group used)                                                                                                                                                                                                                                                                                  | G   |
| STORNAME    | CHAR(8)<br>NOT NULL     | Name of storage group used for space allocation. Blank if storage group not used.                                                                                                                                                                                                                                                                                                                    | G   |
| VCATNAME    | CHAR(8)<br>NOT NULL     | Name of integrated catalog facility catalog used for space allocation.                                                                                                                                                                                                                                                                                                                               | G   |
| CARD        | INTEGER<br>NOT NULL     | Number of rows in the table space or partition or, if the table space is a LOB table space, the number of LOBs in the table space. The value is 2 147 483 647 if the number of rows is greater than or equal to 2 147 483 647. The value is -1 if statistics have not been gathered.                                                                                                                 | G   |
| FARINDREF   | INTEGER<br>NOT NULL     | Number of rows that have been relocated far from their original page. The value is -1 if statistics have not been gathered. Not applicable if the table space is a LOB table space.                                                                                                                                                                                                                  | S   |
| NEARINDREF  | INTEGER<br>NOT NULL     | Number of rows that have been relocated near their original page. The value is -1 if statistics have not been gathered. Not applicable if the table space is a LOB table space.                                                                                                                                                                                                                      | S   |
| PERCACTIVE  | SMALLINT<br>NOT NULL    | Percentage of space occupied by rows of data from active tables. The value is -1 if statistics have not been gathered. The value is -2 if the table space is a LOB table space.                                                                                                                                                                                                                      | S   |
| PERCDROP    | SMALLINT<br>NOT NULL    | Percentage of space occupied by rows of dropped tables. The value is -1 if statistics have not been gathered. The value is 0 for segmented table spaces. Not applicable if the table is an auxiliary table.                                                                                                                                                                                          | S   |

## SYSIBM.SYSTABLEPART

| Column name | Data type                                           | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         | Use |
|-------------|-----------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| IBMREQD     | CHAR(1)<br>NOT NULL                                 | A value of Y indicates that the row came from the basic machine-readable material (MRM) tape. For all other values, see "Release dependency indicators" on page 1005.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               | G   |
| LIMITKEY    | VARCHAR(512)<br>NOT NULL                            | The high value of the partition in external format. The value is 0 if the table space is not partitioned.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           | G   |
| FREEPAGE    | SMALLINT<br>NOT NULL                                | Number of pages loaded before a page is left as free space.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         | G   |
| PCTFREE     | SMALLINT<br>NOT NULL                                | Percentage of each page left as free space.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         | G   |
| CHECKFLAG   | CHAR(1)<br>NOT NULL WITH<br>DEFAULT                 | <p><b>C</b> The table space partition is in a check pending status and there are rows in the table that can violate referential constraints, table check constraints, or both.</p> <p><b>blank</b> The table space is not a partition, or does not contain rows that may violate referential constraints, table check constraints, or both.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     | G   |
|             | CHAR(4)<br>NOT NULL WITH<br>DEFAULT<br>FOR BIT DATA | Not used                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            | N   |
| SPACE       | INTEGER<br>NOT NULL WITH<br>DEFAULT                 | Number of kilobytes of DASD storage allocated to the table space partition, as determined by the last execution of the STOSPACE utility or RUNSTATS utility. The value is 0 if STOSPACE or RUNSTATS has not been run. The value is updated by STOSPACE if the table space is related to a storage group. The value is updated by RUNSTATS if the utility is executed as RUNSTATS TABLESPACE with UPDATE(ALL) or UPDATE(SPACE). The value is -1 if the table space was defined with the DEFINE NO clause, which defers the physical creation of the data sets until data is first inserted into one of the partitions, and data has yet to be inserted.                                                                                                                                                                                                                              | G   |
| COMPRESS    | CHAR(1)<br>NOT NULL WITH<br>DEFAULT                 | <p>Indicates the following:</p> <ul style="list-style-type: none"> <li>For a table space partition, whether the COMPRESS attribute for the partition is YES.</li> <li>For a nonpartitioned table space, whether the COMPRESS attribute is YES for the table space.</li> </ul> <p>Values for the column can be:</p> <p><b>Y</b> Compression is defined for the table space</p> <p><b>blank</b> No compression</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                    | G   |
| PAGESAVE    | SMALLINT<br>NOT NULL WITH<br>DEFAULT                | Percentage of pages saved in the table space or partition as a result of defining the table space with COMPRESS YES or other compression routines. For example, a value of 25 indicates a savings of 25 percent, so that the pages required are only 75 percent of what would be required without data compression. The calculation includes overhead bytes for each row, the bytes required for dictionary, and the bytes required for the current FREEPAGE and PCTFREE specification for the table space or partition. This calculation is based on an average row length, and the result varies depending on the actual lengths of the rows. The value is 0 if there are no savings from using data compression, or if statistics have not been gathered. The value can be negative, if for example, data compression causes an increase in the number of pages in the data set. | S   |
| STATSTIME   | TIMESTAMP<br>NOT NULL WITH<br>DEFAULT               | If RUNSTATS updated the statistics, the date and time when the last invocation of RUNSTATS updated the statistics. The default value is '0001-01-01-00.00.000000'.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  | G   |

| Column name | Data type                               | Description                                                                                                                                                                                                                                                                                                                                                                                                                                          | Use |
|-------------|-----------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| GBPCACHE    | CHAR(1)<br>NOT NULL WITH<br>DEFAULT     | Group buffer pool cache option specified for this table space or table space partition.<br><br><b>A</b> Changed and unchanged pages are cached in the group buffer pool.<br><b>N</b> No data is cached in the group buffer pool.<br><b>S</b> Only changed system pages, such as space map pages that do not contain actual data values, are cached in the group buffer pool.<br><b>blank</b> Only changed pages are cached in the group buffer pool. | G   |
| CHECKRID5B  | CHAR(5)<br>NOT NULL WITH<br>DEFAULT     | Blank if the table or partition is not in a check pending status (CHECKFLAG is blank), or if the table space is not partitioned. Otherwise, the RID of the first row of the table space partition that can violate referential constraints, table check constraints, or both; or the value is X'0000000000', indicating that any row can violate referential constraints.                                                                            | S   |
| TRACKMOD    | CHAR(1)<br>NOT NULL WITH<br>DEFAULT     | Whether to track the page modifications in the space map pages:<br><br><b>N</b> No<br><b>blank</b> Yes                                                                                                                                                                                                                                                                                                                                               | G   |
| # EPOCH     | INTEGER<br>NOT NULL WITH<br>DEFAULT     | A number that increments whenever a utility operation that changes the location of rows in a table occurs.                                                                                                                                                                                                                                                                                                                                           | G   |
| SECQTYI     | INTEGER<br>NOT NULL WITH<br>DEFAULT     | Secondary space allocation in units of 4KB storage. If a storage group is not used, the value is zero.                                                                                                                                                                                                                                                                                                                                               | G   |
| CARDF       | FLOAT<br>NOT NULL WITH<br>DEFAULT -1    | Number of rows in the table space or partition, or if the table space is a LOB table space, the number of LOBs in the table space. The value is -1 if statistics have not been gathered.                                                                                                                                                                                                                                                             | G   |
| IPREFIX     | CHAR(1)<br>NOT NULL WITH<br>DEFAULT 'I' | The first character of the instance qualifier for the data set name for the table space or partition. 'I' or 'J' are the only valid characters for this field. The default is 'I'.                                                                                                                                                                                                                                                                   | G   |
| # ALTEREDTS | TIMESTAMP<br>NOT NULL WITH<br>DEFAULT   | Time when the most recent ALTER TABLESPACE statement was executed for the table space or partition. If no ALTER TABLESPACE statement has been applied, the value is '0001-01-01-00.00.00.000000'.                                                                                                                                                                                                                                                    | G   |
| SPACEF      | FLOAT(8)<br>NOT NULL WITH<br>DEFAULT -1 | Kilobytes of DASD storage. The value is -1 if statistics have not been gathered. This is an updatable column.                                                                                                                                                                                                                                                                                                                                        | G   |
| DSNUM       | INTEGER<br>NOT NULL WITH<br>DEFAULT -1  | Number of data sets. The value is -1 if statistics have not been gathered. This is an updatable column.                                                                                                                                                                                                                                                                                                                                              | G   |
| EXTENTS     | INTEGER<br>NOT NULL WITH<br>DEFAULT -1  | Number of data set extents. The value is -1 if statistics have not been gathered. This is an updatable column. This value is only for the last DSNUM for the object.                                                                                                                                                                                                                                                                                 | G   |

## SYSIBM.SYSTABLEPART\_HIST

### SYSIBM.SYSTABLEPART\_HIST table

# Contains rows from SYSTABLEPART. Rows are added or changed in this table  
# when RUNSTATS collects history statistics. Rows in this table can also be inserted,  
# updated, and deleted.

| Column name | Data type                               | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         | Use |
|-------------|-----------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| PARTITION   | SMALLINT<br>NOT NULL                    | Partition number. 0 if table space is not partitioned.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              | G   |
| TSNAME      | CHAR(8)<br>NOT NULL                     | Name of the table space.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            | G   |
| DBNAME      | CHAR(8)<br>NOT NULL                     | Name of the database that contains the table space.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 | G   |
| PQTY        | INTEGER<br>NOT NULL                     | Primary space allocation in units of 4KB storage blocks. The value of this column is 0 if a storage group is not used. PQTY is based on a value of PRIQTY in the appropriate CREATE or ALTER TABLESPACE statement.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  | G   |
| SECQTYI     | INTEGER<br>NOT NULL                     | Secondary space allocation in units of 4KB storage. If a storage group is not used, the value is 0.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 | G   |
| FARINDREF   | INTEGER<br>NOT NULL WITH<br>DEFAULT -1  | Number of rows that have been relocated far from their original page. The value is -1 if statistics have not been gathered. Not applicable if the table space is a LOB table space.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 | S   |
| NEARINDREF  | INTEGER<br>NOT NULL WITH<br>DEFAULT -1  | Number of rows that have been relocated near their original page. The value is -1 if statistics have not been gathered. Not applicable if the table space is a LOB table space.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     | S   |
| PERCACTIVE  | SMALLINT<br>NOT NULL WITH<br>DEFAULT -1 | Percentage of space occupied by rows of data from active tables. The value is -1 if statistics have not been gathered. The value is -2 if the table space is a LOB table space.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     | S   |
| PERCDROP    | SMALLINT<br>NOT NULL WITH<br>DEFAULT -1 | Percentage of space occupied by rows of dropped tables. The value is -1 if statistics have not been gathered. The value is 0 for segmented table spaces. Not applicable if the table is an auxiliary table.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         | S   |
| SPACEF      | FLOAT(8)<br>NOT NULL WITH<br>DEFAULT -1 | Number of kilobytes of DASD storage allocated to the table space partition. The value is -1 if statistics have not been gathered.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   | G   |
| PAGESAVE    | SMALLINT<br>NOT NULL                    | Percentage of pages saved in the table space or partition as a result of defining the table space with COMPRESS YES or other compression routines. For example, a value of 25 indicates a savings of 25 percent, so that the pages required are only 75 percent of what would be required without data compression. The calculation includes overhead bytes for each row, the bytes required for dictionary, and the bytes required for the current FREEPAGE and PCTFREE specification for the table space or partition. This calculation is based on an average row length, and the result varies depending on the actual lengths of the rows. The value is 0 if there are no savings from using data compression, or if statistics have not been gathered. The value can be negative, if for example, data compression causes an increase in the number of pages in the data set. | S   |
| STATSTIME   | TIMESTAMP<br>NOT NULL                   | If RUNSTATS updated the statistics, the date and time when the last invocation of RUNSTATS updated the statistics. The default value is '0001-01-01-00.00.00.000000'.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               | G   |
| CARDF       | FLOAT(8)<br>NOT NULL WITH<br>DEFAULT -1 | Number of rows in the table space or partition, or if the table space is a LOB table space, the number of LOBS in the table space. The value is -1 if statistics have not been gathered.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            | S   |
| EXTENTS     | INTEGER<br>NOT NULL WITH<br>DEFAULT -1  | Number of data set extents. The value is -1 if statistics have not been gathered. This value is only for the last DSNUM for the object.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             | G   |

**SYSIBM.SYSTABLEPART\_HIST**

| Column name | Data type                               | Description                                                                                                                                                                                                                                                                  | Use |
|-------------|-----------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| DSNUM       | INTEGER<br>NOT NULL WITH<br>DEFAULT -1  | Data set number within the table space. For partitioned table spaces, this value corresponds to the partition number for a single partition copy, or 0 for a copy of an entire partitioned table space or index space. The value is -1 if statistics have not been gathered. | G   |
| IBMRREQD    | CHAR(1)<br>NOT NULL WITH<br>DEFAULT 'N' | A value of Y indicates that the row came from the basic machine-readable material (MRM) tape. For all other values, see "Release dependency indicators" on page 1005.                                                                                                        | G   |

## SYSIBM.SYSTABLES

### SYSIBM.SYSTABLES table

Contains one row for each table, view, or alias.

| Column name | Data type                | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                 | Use |
|-------------|--------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| NAME        | VARCHAR(18)<br>NOT NULL  | Name of the table, view, or alias.                                                                                                                                                                                                                                                                                                                                                                                                                                          | G   |
| CREATOR     | CHAR(8)<br>NOT NULL      | Authorization ID of the owner of the table, view, or alias.                                                                                                                                                                                                                                                                                                                                                                                                                 | G   |
| TYPE        | CHAR(1)<br>NOT NULL      | Type of object:<br><b>A</b> Alias<br><b>G</b> Created global temporary table<br><b>T</b> Table<br><b>V</b> View<br><b>X</b> Auxiliary table                                                                                                                                                                                                                                                                                                                                 | G   |
| DBNAME      | CHAR(8)<br>NOT NULL      | For a table, or a view of tables, the name of the database that contains the table space named in TSNAME. For a created temporary table, an alias, or a view of a view, the value is DSNDB06.                                                                                                                                                                                                                                                                               | G   |
| TSNAME      | CHAR(8)<br>NOT NULL      | For a table, or a view of one table, the name of the table space that contains the table. For a view of more than one table, the name of a table space that contains one of the tables. For a created temporary table, the value is SYSPKAGE. Although SYSPKAGE is used as the value, created temporary tables are not stored in the SYSPKAGE table space. For a view of a view, the value is SYSVIEWS. For an alias, it is SYSDBAUT.                                       | G   |
| DBID        | SMALLINT<br>NOT NULL     | Internal identifier of the database; 0 if the row describes a view, alias, or created temporary table.                                                                                                                                                                                                                                                                                                                                                                      | S   |
| OBID        | SMALLINT<br>NOT NULL     | Internal identifier of the table; 0 if the row describes a view, an alias, or a created temporary table.                                                                                                                                                                                                                                                                                                                                                                    | S   |
| COLCOUNT    | SMALLINT<br>NOT NULL     | Number of columns in the table or view. The value is 0 if the row describes an alias.                                                                                                                                                                                                                                                                                                                                                                                       | G   |
| EDPROC      | CHAR(8)<br>NOT NULL      | Name of the edit procedure; blank if the row describes a view or alias or a table without an edit procedure.                                                                                                                                                                                                                                                                                                                                                                | G   |
| VALPROC     | CHAR(8)<br>NOT NULL      | Name of the validation procedure; blank if the row describes a view or alias or a table without a validation procedure.                                                                                                                                                                                                                                                                                                                                                     | G   |
| CLUSTERTYPE | CHAR(1)<br>NOT NULL      | Whether RESTRICT ON DROP applies:<br><b>blank</b> No<br><b>Y</b> Yes. Neither the table nor any table space or database that contains the table can be dropped.                                                                                                                                                                                                                                                                                                             | G   |
|             | INTEGER<br>NOT NULL      | Not used                                                                                                                                                                                                                                                                                                                                                                                                                                                                    | N   |
|             | INTEGER<br>NOT NULL      | Not used                                                                                                                                                                                                                                                                                                                                                                                                                                                                    | N   |
| NPAGES      | INTEGER<br>NOT NULL      | Total number of pages on which rows of the table appear. The value is -1 if statistics have not been gathered, or the row describes a view, an alias, a created temporary table, or an auxiliary table. This is an updatable column.                                                                                                                                                                                                                                        | S   |
| PCTPAGES    | SMALLINT<br>NOT NULL     | Percentage of active table space pages that contain rows of the table. A page is termed active if it is formatted for rows, regardless of whether it contains any. If the table space is segmented, the percentage is based on the number of active pages in the set of segments assigned to the table. The value is -1 if statistics have not been gathered, or the row describes a view, alias, created temporary table, or auxiliary table. This is an updatable column. | S   |
| IBMRREQD    | CHAR(1)<br>NOT NULL      | A value of Y indicates that the row came from the basic machine-readable material (MRM) tape. For all other values, see "Release dependency indicators" on page 1005.                                                                                                                                                                                                                                                                                                       | G   |
| REMARKS     | VARCHAR(254)<br>NOT NULL | A character string provided by the user with the COMMENT ON statement.                                                                                                                                                                                                                                                                                                                                                                                                      | G   |

| Column name | Data type                                           | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                | Use |
|-------------|-----------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| PARENTS     | SMALLINT<br>NOT NULL                                | Number of relationships in which the table is a dependent. The value is 0 if the row describes a view, an alias, or a created temporary table.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             | G   |
| CHILDREN    | SMALLINT<br>NOT NULL                                | Number of relationships in which the table is a parent. The value is 0 if the row describes a view, an alias, or a created temporary table.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                | G   |
| KEYCOLUMNS  | SMALLINT<br>NOT NULL                                | Number of columns in the table's primary key. The value is 0 if the row describes a view, an alias, or a created temporary table.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          | G   |
| RECLENGTH   | SMALLINT<br>NOT NULL                                | <p>For user tables, the maximum length of any record in the table. Length is <math>8+N+L</math>, where:</p> <ul style="list-style-type: none"> <li>• The number 8 accounts for the header (6 bytes) and the ID map entry (2 bytes).</li> <li>• N is 10 if the table has an edit procedure, or 0 otherwise.</li> <li>• L is the sum of the maximum column lengths. In determining a column's maximum length, take into account whether the column allows nulls and the data type of the column. If the column can contain nulls and is not a LOB or ROWID column, add 1 byte for a null indicator. Use 4 bytes for the length of a LOB column and 19 bytes for the length of a ROWID column. If the column has a varying-length data type (for example, VARCHAR, CLOB, or BLOB), add 2 bytes for a length indicator. For more information on column lengths, see "Data types" on page 48.</li> </ul> <p>The value is 0 if the row describes a view, alias, or auxiliary table. For maximum row and record sizes, see "Maximum record size" on page 676.</p> | G   |
| STATUS      | CHAR(1)<br>NOT NULL                                 | <p>Indicates the status of the table definition:</p> <p><b>I</b> The definition of the table is incomplete. The TABLESTATUS column indicates the reason for the table definition being incomplete.</p> <p><b>X</b> The table has a primary index and the table definition is complete.</p> <p><b>blank</b> The table has no primary index, or is a catalog table, or the row describes a view or alias. The definition of the table, view, or alias is complete.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       | G   |
| KEYOBID     | SMALLINT<br>NOT NULL                                | Internal DB2 identifier of the index that enforces uniqueness of the table's primary key; 0 if not applicable.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             | S   |
| LABEL       | VARCHAR(30)<br>NOT NULL                             | The label as given by a LABEL ON statement; otherwise an empty string.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     | G   |
| CHECKFLAG   | CHAR(1)<br>NOT NULL WITH<br>DEFAULT                 | <p><b>C</b> The table space that contains the table is in a check pending status and there are rows in the table that can violate referential constraints, table check constraints, or both.</p> <p><b>blank</b> The table contains no rows that violate referential constraints, table check constraints, or both; or the row describes a view, alias, or created temporary table.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    | G   |
|             | CHAR(4)<br>NOT NULL WITH<br>DEFAULT<br>FOR BIT DATA | Not used                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   | N   |
| AUDITING    | CHAR(1)<br>NOT NULL WITH<br>DEFAULT                 | <p>Value of the audit option:</p> <p><b>A</b> AUDIT ALL</p> <p><b>C</b> AUDIT CHANGE</p> <p><b>blank</b> AUDIT NONE, or the row describes a view, an alias, or a created temporary table.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              | G   |
| CREATEDBY   | CHAR(8)<br>NOT NULL WITH<br>DEFAULT                 | Primary authorization ID of the user who created the table, view, or alias.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                | G   |

## SYSIBM.SYSTABLES

| Column name  | Data type                                           | Description                                                                                                                                                                                                                                                                                                                                                                                                          | Use |
|--------------|-----------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| LOCATION     | CHAR(16)<br>NOT NULL WITH<br>DEFAULT                | Location name of the object of an alias. Blank for a table, a view, or for an alias that was not defined with a three-part object name.                                                                                                                                                                                                                                                                              | G   |
| TBCREATOR    | CHAR(8)<br>NOT NULL WITH<br>DEFAULT                 | For an alias, the authorization ID of the owner of the referred to table or view; blank otherwise.                                                                                                                                                                                                                                                                                                                   | G   |
| TBNAME       | VARCHAR(18)<br>NOT NULL WITH<br>DEFAULT             | For an alias, the name for the referred to table or view; blank otherwise.                                                                                                                                                                                                                                                                                                                                           | G   |
| CREATEDTS    | TIMESTAMP<br>NOT NULL WITH<br>DEFAULT               | Time when the CREATE statement was executed for the table, view, or alias                                                                                                                                                                                                                                                                                                                                            | G   |
| ALTEREDTS    | TIMESTAMP<br>NOT NULL WITH<br>DEFAULT               | For a table, the time when the latest ALTER TABLE statement was applied. If no ALTER TABLE statement has been applied, or if the row is for a view or alias, ALTEREDTS has the value of CREATEDTS.                                                                                                                                                                                                                   | G   |
| DATA_CAPTURE | CHAR(1)<br>NOT NULL WITH<br>DEFAULT                 | Records the value of the DATA CAPTURE option for a table:<br><b>blank</b> No<br><b>Y</b> Yes                                                                                                                                                                                                                                                                                                                         | G   |
|              |                                                     | For a created temporary table, DATA_CAPTURE is always blank.                                                                                                                                                                                                                                                                                                                                                         |     |
| RBA1         | CHAR(6)<br>NOT NULL WITH<br>DEFAULT<br>FOR BIT DATA | The log RBA when the table was created. Otherwise, RBA1 is X'000000000000', indicating that the log RBA is not known, or that the object is a view, an alias, or a created temporary table. In a data sharing environment, RBA1 is the LRSN (Log Record Sequence Number) value.                                                                                                                                      | S   |
| RBA2         | CHAR(6)<br>NOT NULL WITH<br>DEFAULT<br>FOR BIT DATA | The log RBA when the table was last altered. Otherwise, RBA2 is X'000000000000' indicating that the log RBA is not known, or that the object is a view, an alias, or a created temporary table. RBA1 will equal RBA2 if the table has not been altered. In a data sharing environment, RBA2 is the LRSN (Log Record Sequence Number) value.                                                                          | S   |
| PCTROWCOMP   | SMALLINT<br>NOT NULL WITH<br>DEFAULT                | Percentage of rows compressed within the total number of active rows in the table. This includes any row in a table space that is defined with COMPRESS YES. The value is -1 if statistics have not been gathered, or the row describes a view, alias, created temporary table, or auxiliary table. This is an updatable column.                                                                                     | S   |
| STATSTIME    | TIMESTAMP<br>NOT NULL WITH<br>DEFAULT               | If RUNSTATS updated the statistics, the date and time when the last invocation of RUNSTATS updated the statistics. The default value is '0001-01-01-00.00.00.000000'. For a created temporary table, the value of STATSTIME is always the default value. This is an updatable column.                                                                                                                                | G   |
| CHECKS       | SMALLINT<br>NOT NULL WITH<br>DEFAULT                | Number of check constraints defined on the table. The value is 0 if the row describes a view, an alias, or a created temporary table, or if no constraints are defined on the table.                                                                                                                                                                                                                                 | G   |
| CARDF        | FLOAT<br>NOT NULL WITH<br>DEFAULT -1                | Total number of rows in the table or total number of LOBs in an auxiliary table. The value is -1 if statistics have not been gathered or the row describes a view, alias, or created temporary table. This is an updatable column.                                                                                                                                                                                   | S   |
| CHECKRID5B   | CHAR(5)<br>NOT NULL WITH<br>DEFAULT                 | Blank if the table or partition is not in a check pending status (CHECKFLAG is blank), if the table space is not partitioned, or if the table is a created temporary table. Otherwise, the RID of the first row of the table space partition that can violate referential constraints, table check constraints, or both; or the value is X'0000000000', indicating that any row can violate referential constraints. | S   |

## SYSIBM.SYSTABLES

| Column name     | Data type                               | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        | Use |
|-----------------|-----------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| ENCODING_SCHEME | CHAR(1)<br>NOT NULL WITH<br>DEFAULT 'E' | Encoding scheme for tables, views, and local aliases:<br><b>E</b> EBCDIC<br><b>A</b> ASCII<br><b>U</b> UNICODE<br><b>blank</b> For remote aliases<br>The value is 'E' for tables in non work file databases and blank for tables in work file databases created prior to Version 5 or the default database, DSNDB04.                                                                                                                                                                               | G   |
| TABLESTATUS     | VARCHAR(10)<br>NOT NULL WITH<br>DEFAULT | Indicates the reason for an incomplete table definition:<br><b>L</b> Definition is incomplete because an auxiliary table or auxiliary index has not been defined for a LOB column.<br><b>P</b> Definition is incomplete because the table lacks a primary index.<br><b>R</b> Definition is incomplete because the table lacks a required index on a row ID.<br><b>U</b> Definition is incomplete because the table lacks a required index on a unique key.<br><b>blank</b> Definition is complete. | G   |
| NPAGESF         | FLOAT(8)<br>NOT NULL WITH<br>DEFAULT -1 | Number of pages used by the table. The value is -1 if statistics have not been gathered. This is an updatable column.                                                                                                                                                                                                                                                                                                                                                                              | G   |
| SPACEF          | FLOAT(8)<br>NOT NULL WITH<br>DEFAULT -1 | Kilobytes of DASD storage. The value is -1 if statistics have not been gathered. This is an updatable column.                                                                                                                                                                                                                                                                                                                                                                                      | G   |
| AVGROWLEN       | INTEGER<br>NOT NULL WITH<br>DEFAULT -1  | Average length of rows for the tables in the table space. If the table space is compressed, the value is the compressed row length. If the table space is not compressed, the value is the uncompressed row length. The value is -1 if statistics have not been gathered.                                                                                                                                                                                                                          | G   |
| RELCREATED      | CHAR(1)<br>NOT NULL WITH<br>DEFAULT     | Release of DB2 that was used to create the object:<br><b>blank</b> Created prior to Version 7.<br><b>K</b> Created on Version 7                                                                                                                                                                                                                                                                                                                                                                    | G   |

## SYSIBM.SYSTABLES\_HIST

### SYSIBM.SYSTABLES\_HIST table

# Contains rows from SYSTABLES. Rows are added or changed in this table when  
# RUNSTATS collects history statistics. Rows in this table can also be inserted,  
| updated, and deleted.

| Column name | Data type                               | Description                                                                                                                                                                                                                                                                                                                                                                                                                            | Use |
|-------------|-----------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| NAME        | VARCHAR(18)<br>NOT NULL                 | Name of the table, view, or alias.                                                                                                                                                                                                                                                                                                                                                                                                     | G   |
| CREATOR     | CHAR(8)<br>NOT NULL                     | Authorization ID of the owner of the table, view, or alias.                                                                                                                                                                                                                                                                                                                                                                            | G   |
| DBNAME      | CHAR(8)<br>NOT NULL                     | For a table, or a view of tables, the name of the database that contains the table space named in TSNAME. For a temporary table, an alias, or a view of a view, the value is DSNDB06.                                                                                                                                                                                                                                                  | G   |
| TSNAME      | CHAR(8)<br>NOT NULL                     | For a table, or a view of one table, the name of the table space that contains the table. For a view of more than one table, the name of a table space that contains one of the tables. For a temporary table, the value is SYSPKGAGE. For a view of a view, the value is SYSVIEWS. For an alias, it is SYSDBAUT.                                                                                                                      | G   |
| COLCOUNT    | SMALLINT<br>NOT NULL                    | Number of columns in the table or view. The value is 0 if the row describes an alias.                                                                                                                                                                                                                                                                                                                                                  | G   |
| PCTPAGES    | SMALLINT<br>NOT NULL WITH<br>DEFAULT -1 | Percentage of active table space pages that contain rows of the table. A page is termed active if it is formatted for rows, regardless of whether it contains any. If the table space is segmented, the percentage is based on the number of active pages in the set of segments assigned to the table. The value is -1 if statistics have not been gathered, or the row describes a view, alias, temporary table, or auxiliary table. | S   |
| PCTROWCOMP  | SMALLINT<br>NOT NULL WITH<br>DEFAULT -1 | Percentage of rows compressed within the total number of active rows in the table. This includes any row in a table space that is defined with COMPRESS YES. The value is -1 if statistics have not been gathered, or the row describes a view, alias, temporary table, or auxiliary table.                                                                                                                                            | G   |
| STATSTIME   | TIMESTAMP<br>NOT NULL                   | If RUNSTATS updated the statistics, the date and time when the last invocation of RUNSTATS updated the statistics. The default value is '0001-01-01-00.00.00.000000'. For a temporary table, the value of STATSTIME is always the default value.                                                                                                                                                                                       | G   |
| CARDF       | FLOAT(8)<br>NOT NULL WITH<br>DEFAULT -1 | Total number of rows in the table or total number of LOBs in an auxiliary table. The value is -1 if statistics have not been gathered or the row describes a view, alias, or temporary table.                                                                                                                                                                                                                                          | S   |
| NPAGESF     | FLOAT(8)<br>NOT NULL WITH<br>DEFAULT -1 | Total number of pages on which rows of the partition appear. The value is -1 if statistics have not been gathered.                                                                                                                                                                                                                                                                                                                     | S   |
| AVGROWLEN   | INTEGER<br>NOT NULL WITH<br>DEFAULT -1  | Average row length of the table specified in the table space. The value is -1 if statistics have not been gathered.                                                                                                                                                                                                                                                                                                                    | G   |
| SPACEF      | FLOAT(8)<br>NOT NULL WITH<br>DEFAULT -1 | Kilobytes of DASD storage. The value is -1 if statistics have not been gathered. This is an updatable column.                                                                                                                                                                                                                                                                                                                          | G   |
| IBMREQD     | CHAR(1)<br>NOT NULL WITH<br>DEFAULT 'N' | A value of Y indicates that the row came from the basic machine-readable material (MRM) tape. For all other values, see "Release dependency indicators" on page 1005.                                                                                                                                                                                                                                                                  | G   |

**SYSIBM.SYSTABLESPACE table**

Contains one row for each table space.

| Column name | Data type            | Description                                                                                                                                                                                                                                                                                                                                                | Use |
|-------------|----------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| NAME        | CHAR(8)<br>NOT NULL  | Name of the table space.                                                                                                                                                                                                                                                                                                                                   | G   |
| CREATOR     | CHAR(8)<br>NOT NULL  | Authorization ID of the owner of the table space.                                                                                                                                                                                                                                                                                                          | G   |
| DBNAME      | CHAR(8)<br>NOT NULL  | Name of the database that contains the table space.                                                                                                                                                                                                                                                                                                        | G   |
| DBID        | SMALLINT<br>NOT NULL | Internal identifier of the database which contains the table space.                                                                                                                                                                                                                                                                                        | S   |
| OBID        | SMALLINT<br>NOT NULL | Internal identifier of the table space file descriptor.                                                                                                                                                                                                                                                                                                    | S   |
| PSID        | SMALLINT<br>NOT NULL | Internal identifier of the table space page set descriptor.                                                                                                                                                                                                                                                                                                | S   |
| BPOOL       | CHAR(8)<br>NOT NULL  | Name of the buffer pool used for the table space.                                                                                                                                                                                                                                                                                                          | G   |
| PARTITIONS  | SMALLINT<br>NOT NULL | Number of partitions of the table space; 0 if the table space is not partitioned.                                                                                                                                                                                                                                                                          | G   |
| LOCKRULE    | CHAR(1)<br>NOT NULL  | Lock size of the table space:<br><br>A Any<br>L Large object (LOB)<br>P Page<br>R Row<br>S Table space<br>T Table                                                                                                                                                                                                                                          | G   |
| PGSIZE      | SMALLINT<br>NOT NULL | Size of pages in the table space in kilobytes.                                                                                                                                                                                                                                                                                                             | G   |
| ERASERULE   | CHAR(1)<br>NOT NULL  | Whether the data sets are to be erased when dropped. The value is meaningless if the table space is partitioned.<br><br>N No erase<br>Y Erase                                                                                                                                                                                                              | G   |
| STATUS      | CHAR(1)<br>NOT NULL  | Availability status of the table space:<br><br>A Available<br>C Definition is incomplete because a partitioning index has not been created.<br>P Table space is in a check pending status.<br>S Table space is in a check pending status with the scope less than the entire table space.<br>T Definition is incomplete because no table has been created. | G   |
| IMPLICIT    | CHAR(1)<br>NOT NULL  | Whether the table space was created implicitly:<br><br>N No<br>Y Yes                                                                                                                                                                                                                                                                                       | G   |
| NTABLES     | SMALLINT<br>NOT NULL | Number of tables defined in the table space.                                                                                                                                                                                                                                                                                                               | G   |
| NACTIVE     | INTEGER<br>NOT NULL  | Number of active pages in the table space. A page is termed active if it is formatted for rows, even if it currently contains none. The value is 0 if statistics have not been gathered. This is an updatable column.                                                                                                                                      | S   |
|             | CHAR(8)<br>NOT NULL  | Not used                                                                                                                                                                                                                                                                                                                                                   | N   |
| CLOSERULE   | CHAR(1)<br>NOT NULL  | Whether the data sets are candidates for closure when the limit on the number of open data sets is reached.<br><br>N No<br>Y Yes                                                                                                                                                                                                                           | G   |

## SYSIBM.SYSTABLESPACE

| Column name | Data type                             | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                | Use |
|-------------|---------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| SPACE       | INTEGER<br>NOT NULL                   | Number of kilobytes of DASD storage allocated to the table space, as determined by the last execution of the STOSPACE utility. The value is 0 if the table space is not related to a storage group, or if STOSPACE has not been run. If the table space is partitioned, the value is the total kilobytes of DASD storage allocated to all partitions that are storage group defined.                                                                                                                                                                                                                                                                       | G   |
| IBMREQD     | CHAR(1)<br>NOT NULL                   | A value of Y indicates that the row came from the basic machine-readable material (MRM) tape. For all other values, see "Release dependency indicators" on page 1005.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      | G   |
|             | VARCHAR(18)<br>NOT NULL               | Internal use only                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          | I   |
|             | CHAR(8)<br>NOT NULL                   | Internal use only                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          | I   |
| SEGSIZE     | SMALLINT<br>NOT NULL WITH<br>DEFAULT  | Number of pages in each segment of a segmented table space. The value is 0 if the table space is not segmented.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            | G   |
| CREATEDBY   | CHAR(8)<br>NOT NULL WITH<br>DEFAULT   | Primary authorization ID of the user who created the table space.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          | G   |
| STATSTIME   | TIMESTAMP<br>NOT NULL WITH<br>DEFAULT | If RUNSTATS updated the statistics, the date and time when the last invocation of RUNSTATS updated the statistics. The default value is '0001-01-01-00.00.00.000000'. This is an updatable column.                                                                                                                                                                                                                                                                                                                                                                                                                                                         | G   |
| LOCKMAX     | INTEGER                               | The maximum number of locks per user to acquire for the table or table space before escalating to the next locking level.<br><br><b>0</b> Lock escalation does not occur.<br><b>n</b> n, where n > 0, is the maximum number of locks (row, page, or LOB locks for the table or table space) an application process can acquire before lock escalation occurs.<br><b>-1</b> Represents LOCKMAX SYSTEM. The value of field LOCKS PER TABLE(SPACE) on installation panel DSNTIPJ determines lock escalation. If the value of the field is 0, lock escalation does not occur. If the value is n, where n > 0, lock escalation occurs as it does for LOCKMAX n. | G   |
| TYPE        | CHAR(1)<br>NOT NULL WITH<br>DEFAULT   | The type of table space:<br><br><b>blank</b> The table space was created without any of the following options: DSSIZE, LARGE, LOB, and MEMBER CLUSTER.<br><b>I</b> The table space was defined with the MEMBER CLUSTER option and is not greater than 64 gigabytes.<br><b>K</b> The table space was defined with the MEMBER CLUSTER option and can be greater than 64 gigabytes.<br><b>L</b> The table space can be greater than 64 gigabytes.<br><b>O</b> The table space was defined with the LOB option (the table space is a LOB table space).                                                                                                         | G   |
| CREATEDTS   | TIMESTAMP<br>NOT NULL WITH<br>DEFAULT | Time when the CREATE statement was executed for the table space. If the table space was created in a DB2 release prior to Version 5, the value is '0001-01-01-00.00.000000'.                                                                                                                                                                                                                                                                                                                                                                                                                                                                               | G   |
| ALTEREDTS   | TIMESTAMP<br>NOT NULL WITH<br>DEFAULT | Time when the most recent ALTER TABLESPACE statement was executed for the table space. If no ALTER TABLESPACE statement has been applied, ALTEREDTS has the value of CREATEDTS. If the index was created in a DB2 release prior to Version 5, the value is '0001-01-01-00.00.000000'.                                                                                                                                                                                                                                                                                                                                                                      | G   |

## SYSIBM.SYSTABLESPACE

| Column name     | Data type                               | Description                                                                                                                                                                                                                                                                                                                                                                                                                          | Use |
|-----------------|-----------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| ENCODING_SCHEME | CHAR(1)<br>NOT NULL WITH<br>DEFAULT 'E' | Default encoding scheme for the table space:<br><b>E</b> EBCDIC<br><b>A</b> ASCII<br><b>U</b> UNICODE<br><b>blank</b> For tables spaces in a work file database or a TEMP database (a database that was created AS TEMP, which is for declared temporary tables.)<br>The value is 'E' for tables in non work file databases and blank for tables in work file databases created prior to Version 5 or the default database, DSNDB04. | G   |
| SBCS_CCSID      | INTEGER<br>NOT NULL WITH<br>DEFAULT     | Default SBCS CCSID for the table space. For a table space in a TEMP database or a database created in a DB2 release prior to Version 5, the value is 0.                                                                                                                                                                                                                                                                              | G   |
| DBCS_CCSID      | INTEGER<br>NOT NULL WITH<br>DEFAULT     | Default DBCS CCSID for the table space. For a table space in a TEMP database or a database created in a DB2 release prior to Version 5, the value is 0.                                                                                                                                                                                                                                                                              | G   |
| MIXED_CCSID     | INTEGER<br>NOT NULL WITH<br>DEFAULT     | Default mixed CCSID for the table space. For a table space in a TEMP database or ar database created in a DB2 release prior to Version 5, the value is 0.                                                                                                                                                                                                                                                                            | G   |
| MAXROWS         | SMALLINT<br>NOT NULL<br>DEFAULT 255     | The maximum number of rows that DB2 will place on a data page. The default value is 255. For a LOB table space, the value is 0 to indicate that the column is not applicable.                                                                                                                                                                                                                                                        | G   |
| LOCKPART        | CHAR(1)<br>NOT NULL WITH<br>DEFAULT     | <b>Y</b> LOCKPART YES is specified for the table space.<br><b>blank</b> LOCKPART NO is specified, or LOCKPART is not specified or not a partitioned table space.                                                                                                                                                                                                                                                                     | G   |
| LOG             | CHAR(1)<br>NOT NULL WITH<br>DEFAULT 'Y' | Whether the changes to a table space are to be logged.<br><b>N</b> No, only applies to LOB table spaces<br><b>Y</b> Yes                                                                                                                                                                                                                                                                                                              | G   |
| NACTIVEF        | FLOAT<br>NOT NULL WITH<br>DEFAULT -1    | Number of active pages in the table space. A page is termed active if it is formatted for rows, even if it currently contains none. The value is -1 if statistics have not been gathered. This is an updatable column.                                                                                                                                                                                                               | S   |
| DSSIZE          | INTEGER<br>NOT NULL WITH<br>DEFAULT     | Maximum size of a data set in kilobytes.                                                                                                                                                                                                                                                                                                                                                                                             | G   |

## SYSIBM.SYSTABSTATS

### SYSIBM.SYSTABSTATS table

Contains one row for each partition of a partitioned table space. Rows in this table can be inserted, updated, and deleted.

| Column name | Data type                            | Description                                                                                                                                                           | Use |
|-------------|--------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| CARD        | INTEGER<br>NOT NULL                  | Total number of rows in the partition.                                                                                                                                | S   |
| NPAGES      | INTEGER<br>NOT NULL                  | Total number of pages on which rows of the partition appear.                                                                                                          | S   |
| PCTPAGES    | SMALLINT<br>NOT NULL                 | Percentage of total active pages in the partition that contain rows of the table.                                                                                     | S   |
| NACTIVE     | INTEGER<br>NOT NULL                  | Number of active pages in the partition.                                                                                                                              | S   |
| PCTROWCOMP  | SMALLINT<br>NOT NULL                 | Percentage of rows compressed within the total number of active rows in the partition. This includes any row in a table space that is defined with COMPRESS YES.      | S   |
| STATSTIME   | TIMESTAMP<br>NOT NULL                | If RUNSTATS updated the statistics, the date and time when the last invocation of RUNSTATS updated the statistics.                                                    | G   |
| IBMREQD     | CHAR(1)<br>NOT NULL                  | A value of Y indicates that the row came from the basic machine-readable material (MRM) tape. For all other values, see "Release dependency indicators" on page 1005. | G   |
| DBNAME      | CHAR(8)<br>NOT NULL                  | Database that contains the table space named in TSNAME.                                                                                                               | G   |
| TSNAME      | CHAR(8)<br>NOT NULL                  | Table space that contains the table.                                                                                                                                  | G   |
| PARTITION   | SMALLINT<br>NOT NULL                 | Partition number of the table space that contains the table.                                                                                                          | G   |
| OWNER       | CHAR(8)<br>NOT NULL                  | Authorization ID of the owner of the table.                                                                                                                           | G   |
| NAME        | VARCHAR(18)<br>NOT NULL              | Name of the table.                                                                                                                                                    | G   |
| CARDF       | FLOAT<br>NOT NULL WITH<br>DEFAULT -1 | Total number of rows in the partition.                                                                                                                                | S   |

---

## SYSIBM.SYSTABSTATS\_HIST table

# Rows are added or changed in this table when RUNSTATS collects history statistics. Rows in this table can also be inserted, updated, and deleted.

| Column name | Data type                               | Description                                                                                                        | Use |
|-------------|-----------------------------------------|--------------------------------------------------------------------------------------------------------------------|-----|
| NPAGES      | INTEGER<br>NOT NULL                     | Total number of pages on which rows of the partition appear.                                                       | S   |
| STATSTIME   | TIMESTAMP<br>NOT NULL                   | If RUNSTATS updated the statistics, the date and time when the last invocation of RUNSTATS updated the statistics. | G   |
| DBNAME      | CHAR(8)<br>NOT NULL                     | Database that contains the table space named in TSNAME.                                                            | G   |
| TSNAME      | CHAR(8)<br>NOT NULL                     | Table space that contains the table.                                                                               | G   |
| PARTITION   | SMALLINT<br>NOT NULL                    | Partition number of the table space that contains the table.                                                       | G   |
| OWNER       | CHAR(8)<br>NOT NULL                     | Authorization ID of the owner of the table.                                                                        | G   |
| NAME        | VARCHAR(18)<br>NOT NULL                 | Name of the table.                                                                                                 | G   |
| CARDF       | FLOAT(8)<br>NOT NULL WITH<br>DEFAULT -1 | Total number of rows in the partition. The value is -1 if statistics have not been gathered.                       | S   |
| IBMREQD     | CHAR(1)<br>NOT NULL WITH<br>DEFAULT 'N' |                                                                                                                    | G   |

---

## SYSIBM.SYSTRIGGERS

### SYSIBM.SYSTRIGGERS table

Contains one row for each trigger.

| Column name | Data type                 | Description                                                                                                                                                                                                               | Use |
|-------------|---------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| NAME        | CHAR(8)<br>NOT NULL       | Name of the trigger and trigger package.                                                                                                                                                                                  | G   |
| SCHEMA      | CHAR(8)<br>NOT NULL       | Schema of the trigger. This implicit or explicit qualifier for the trigger name is also used for the collection ID of the trigger package.                                                                                | G   |
| SEQNO       | SMALLINT<br>NOT NULL      | Sequence number of this row; the first portion of the trigger definition is in row 1, and successive rows have increasing SEQNO values.                                                                                   | G   |
| DBID        | SMALLINT<br>NOT NULL      | Internal identifier of the database for the trigger.                                                                                                                                                                      | G   |
| OBID        | SMALLINT<br>NOT NULL      | Internal identifier of the trigger.                                                                                                                                                                                       | G   |
| OWNER       | CHAR(8)<br>NOT NULL       | Authorization ID of the owner of the trigger. The value is set to the current authorization ID (the plan or package owner for static CREATE TRIGGER statement; the current SQLID for a dynamic CREATE TRIGGER statement). | G   |
| CREATEDBY   | CHAR(8)<br>NOT NULL       | Authorization ID of the owner of the trigger. The value is set to the current authorization ID (the plan or package owner for static CREATE TRIGGER statement; the current SQLID for a dynamic CREATE TRIGGER statement). | G   |
| TBNAME      | VARCHAR(18)<br>NOT NULL   | Name of the table to which this trigger applies.                                                                                                                                                                          | G   |
| TBOWNER     | CHAR(8)<br>NOT NULL       | Qualifier of the name of the table to which this trigger applies.                                                                                                                                                         | G   |
| TRIGTIME    | CHAR(1)<br>NOT NULL       | Time when triggered actions are applied to the base table, relative to the event that activated the trigger:<br><b>B</b> Trigger is applied before the event.<br><b>A</b> Trigger is applied after the event.             | G   |
| TRIGEVENT   | CHAR(1)<br>NOT NULL       | Operation that activates the trigger:<br><b>I</b> Insert<br><b>D</b> Delete<br><b>U</b> Update                                                                                                                            | G   |
| GRANULARITY | CHAR(1)<br>NOT NULL       | Trigger is executed once per:<br><b>S</b> Statement<br><b>R</b> Row                                                                                                                                                       | G   |
| CREATEDTS   | TIMESTAMP<br>NOT NULL     | Time when the CREATE statement was executed for this trigger. The time value is used in resolving functions, distinct types, and stored procedures. It is also used to order the execution of multiple triggers.          | G   |
| IBMREQD     | CHAR(1)<br>NOT NULL       | A value of Y indicates that the row came from the basic machine-readable material (MRM) tape. For all other values, see “Release dependency indicators” on page 1005.                                                     | G   |
| TEXT        | VARCHAR(3460)<br>NOT NULL | Full text of the CREATE TRIGGER statement.                                                                                                                                                                                | G   |
| REMARKS     | VARCHAR(254)<br>NOT NULL  | A character string provided by the user with the COMMENT ON statement.                                                                                                                                                    | G   |
| TRIGNAME    | VARCHAR(30)<br>NOT NULL   | Unused                                                                                                                                                                                                                    | G   |

## SYSIBM.SYSUSERAUTH table

Records the system privileges that are held by users.

| Column name   | Data type            | Description                                                                                                                                                                                                                                                                              | Use |
|---------------|----------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| GRANTOR       | CHAR(8)<br>NOT NULL  | Authorization ID of the user who granted the privileges.                                                                                                                                                                                                                                 | G   |
| GRANTEE       | CHAR(8)<br>NOT NULL  | Authorization ID of the user that holds the privilege. Could also be PUBLIC for a grant to PUBLIC.                                                                                                                                                                                       | G   |
|               | CHAR(12)<br>NOT NULL | Internal use only                                                                                                                                                                                                                                                                        | I   |
|               | CHAR(6)<br>NOT NULL  | Not used.                                                                                                                                                                                                                                                                                | N   |
|               | CHAR(8)<br>NOT NULL  | Not used.                                                                                                                                                                                                                                                                                | N   |
|               | CHAR(1)<br>NOT NULL  | Not used                                                                                                                                                                                                                                                                                 | N   |
| AUTHHOWGOT    | CHAR(1)<br>NOT NULL  | Authorization level of the user from whom the privileges were received. This authorization level is not necessarily the highest authorization level of the grantor.<br><br>blank Not applicable<br>C DBCTL<br>D DBADM<br>L SYSCTRL<br>M DBMAINT<br>O SYSOPR<br>S SYSADM                  | G   |
|               | CHAR(1)<br>NOT NULL  | Not used                                                                                                                                                                                                                                                                                 | N   |
| BINDADDAUTH   | CHAR(1)<br>NOT NULL  | Whether the GRANTEE can use the BIND subcommand with the ADD option:<br><br>blank Privilege is not held<br>G Privilege is held with the GRANT option<br>Y Privilege is held without the GRANT option                                                                                     | G   |
| BSDSAUTH      | CHAR(1)<br>NOT NULL  | Whether the GRANTEE can issue the RECOVER BSDS command:<br><br>blank Privilege is not held<br>G Privilege is held with the GRANT option<br>Y Privilege is held without the GRANT option                                                                                                  | G   |
| CREATEDBAAUTH | CHAR(1)<br>NOT NULL  | Whether the GRANTEE can create databases and automatically receive DBADM authority over the new databases:<br><br>blank Privilege is not held<br>G Privilege is held with the GRANT option<br>Y Privilege is held without the GRANT option                                               | G   |
| CREATEDBCAUTH | CHAR(1)<br>NOT NULL  | Whether the GRANTEE can execute the CREATE DATABASE statement to create new databases and automatically receive DBCTRL authority over the new databases:<br><br>blank Privilege is not held<br>G Privilege is held with the GRANT option<br>Y Privilege is held without the GRANT option | G   |
| CREATESGAUTH  | CHAR(1)<br>NOT NULL  | Whether the GRANTEE can execute the CREATE STOGROUP statement to create new storage groups:<br><br>blank Privilege is not held<br>G Privilege is held with the GRANT option<br>Y Privilege is held without the GRANT option                                                              | G   |
| DISPLAYAUTH   | CHAR(1)<br>NOT NULL  | Whether the GRANTEE can use the DISPLAY commands:<br><br>blank Privilege is not held<br>G Privilege is held with the GRANT option<br>Y Privilege is held without the GRANT option                                                                                                        | G   |

## SYSIBM.SYSUSERAUTH

| Column name     | Data type                           | Description                                                                                                                                                                                                             | Use |
|-----------------|-------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| RECOVERAUTH     | CHAR(1)<br>NOT NULL                 | Whether the GRANTEE can use the RECOVER INDOUBT command:<br><b>blank</b> Privilege is not held<br><b>G</b> Privilege is held with the GRANT option<br><b>Y</b> Privilege is held without the GRANT option               | G   |
| STOPALLAUTH     | CHAR(1)<br>NOT NULL                 | Whether the GRANTEE can use the STOP command:<br><b>blank</b> Privilege is not held<br><b>G</b> Privilege is held with the GRANT option<br><b>Y</b> Privilege is held without the GRANT option                          | G   |
| STOSPACEAUTH    | CHAR(1)<br>NOT NULL                 | Whether the GRANTEE can use the STOSPACE utility:<br><b>blank</b> Privilege is not held<br><b>G</b> Privilege is held with the GRANT option<br><b>Y</b> Privilege is held without the GRANT option                      | G   |
| SYSADMAUTH      | CHAR(1)<br>NOT NULL                 | Whether the GRANTEE has system administration authority:<br><b>blank</b> Privilege is not held<br><b>G</b> Privilege was granted with the GRANT option<br><b>Y</b> Privilege was granted without the GRANT option       | G   |
| SYSOPRAUTH      | CHAR(1)<br>NOT NULL                 | GRANTEE has the privilege with the GRANT option for a value of either Y or G.                                                                                                                                           | G   |
| TRACEAUTH       | CHAR(1)<br>NOT NULL                 | Whether the GRANTEE can issue the START TRACE and STOP TRACE commands:<br><b>blank</b> Privilege is not held<br><b>G</b> Privilege is held with the GRANT option<br><b>Y</b> Privilege is held without the GRANT option | G   |
| IBMREQD         | CHAR(1)<br>NOT NULL                 | A value of Y indicates that the row came from the basic machine-readable material (MRM) tape. For all other values, see "Release dependency indicators" on page 1005.                                                   | G   |
| MON1AUTH        | CHAR(1)<br>NOT NULL WITH<br>DEFAULT | Whether the GRANTEE can obtain IFC serviceability data:<br><b>blank</b> Privilege is not held<br><b>G</b> Privilege is held with the GRANT option<br><b>Y</b> Privilege is held without the GRANT option                | G   |
| MON2AUTH        | CHAR(1)<br>NOT NULL WITH<br>DEFAULT | Whether the GRANTEE can obtain IFC data:<br><b>blank</b> Privilege is not held<br><b>G</b> Privilege is held with the GRANT option<br><b>Y</b> Privilege is held without the GRANT option                               | G   |
| CREATEALIASAUTH | CHAR(1)<br>NOT NULL WITH<br>DEFAULT | Whether the GRANTEE can execute the CREATE ALIAS statement:<br><b>blank</b> Privilege is not held<br><b>G</b> Privilege held with the GRANT option<br><b>Y</b> Privilege held without the GRANT option                  | G   |
| SYSCTRLAUTH     | CHAR(1)<br>NOT NULL WITH<br>DEFAULT | Whether the GRANTEE has SYSCTRL authority:<br><b>blank</b> Privilege is not held<br><b>G</b> Privilege is held with the GRANT option<br><b>Y</b> Privilege is held without the GRANT option                             | G   |
| BINDAGENTAUTH   | CHAR(1)<br>NOT NULL WITH<br>DEFAULT | Whether the GRANTEE has BINDAGENT privilege:<br><b>blank</b> Privilege is not held<br><b>G</b> Privilege is held with the GRANT option<br><b>Y</b> Privilege is held without the GRANT option                           | G   |
|                 |                                     | See "GRANT (system privileges)" on page 821 for a description of the BINDAGENT privilege.                                                                                                                               |     |

## SYSIBM.SYSUSERAUTH

| Column name     | Data type                             | Description                                                                                                                                                                                                        | Use |
|-----------------|---------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| ARCHIVEAUTH     | CHAR(1)<br>NOT NULL WITH<br>DEFAULT   | Whether the GRANTEE is privileged to use the ARCHIVE LOG command:<br><b>blank</b> Privilege is not held<br><b>G</b> Privilege is held with the GRANT option<br><b>Y</b> Privilege is held without the GRANT option | G   |
|                 | CHAR(1)<br>NOT NULL WITH<br>DEFAULT   | Not used                                                                                                                                                                                                           | N   |
|                 | CHAR(1)<br>NOT NULL WITH<br>DEFAULT   | Not used                                                                                                                                                                                                           | N   |
| GRANTEDTS       | TIMESTAMP<br>NOT NULL WITH<br>DEFAULT | Time when the GRANT statement was executed. The value is '1985-04-01-00.00.00.000000' for the one installation row.                                                                                                | G   |
| CREATETMTABAUTH | CHAR(1)<br>NOT NULL WITH<br>DEFAULT   | Whether the GRANTEE has CREATETMTABAUTH privilege:<br><b>blank</b> Privilege is not held<br><b>G</b> Privilege is held with the GRANT option<br><b>Y</b> Privilege is held without the GRANT option                | G   |

**SYSIBM.SYSVIEWDEP table**

Records the dependencies of views on tables, functions, and other views.

| Column name | Data type                               | Description                                                                                                                                                           | Use |
|-------------|-----------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| BNAME       | VARCHAR(18)<br>NOT NULL                 | Name of the object on which the view is dependent. If the object type is a function (BTTYPE='F'), the name is the specific name of the function.                      | G   |
| BCREATOR    | CHAR(8)<br>NOT NULL                     | Authorization ID of the owner of BNAME. For functions, it is the schema name of the BNAME.                                                                            | G   |
| BTTYPE      | CHAR(1)<br>NOT NULL                     | Type of object:<br><b>F</b> Function<br><b>G</b> Temporary table<br><b>T</b> Table<br><b>V</b> View                                                                   | G   |
| DNAME       | VARCHAR(18)<br>NOT NULL                 | Name of the view.                                                                                                                                                     | G   |
| DCREATOR    | CHAR(8)<br>NOT NULL                     | Authorization ID of the owner of the view.                                                                                                                            | G   |
| IBMREQD     | CHAR(1)<br>NOT NULL                     | A value of Y indicates that the row came from the basic machine-readable material (MRM) tape. For all other values, see "Release dependency indicators" on page 1005. | G   |
| BSHEMA      | CHAR(8)<br>NOT NULL WITH<br>DEFAULT     | Schema of BNAME.                                                                                                                                                      | G   |
| # DTYPE     | CHAR(1)<br>NOT NULL WITH<br>DEFAULT 'V' | Type of table:<br><b>F</b> SQL function<br><b>V</b> View                                                                                                              | G   |

## SYSIBM.SYSVIEWS table

Contains one or more rows for each view.

| Column name | Data type                                | Description                                                                                                                                                                                                                            | Use |
|-------------|------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| NAME        | VARCHAR(18)<br>NOT NULL                  | Name of the view.                                                                                                                                                                                                                      | G   |
| CREATOR     | CHAR(8)<br>NOT NULL                      | Authorization ID of the owner of the view.                                                                                                                                                                                             | G   |
| SEQNO       | SMALLINT<br>NOT NULL                     | Sequence number of this row; the first portion of the view is on row one and successive rows have increasing values of SEQNO.                                                                                                          | G   |
| CHECK       | CHAR(1)<br>NOT NULL                      | Whether the WITH CHECK OPTION clause was specified in the CREATE VIEW statement:<br><br>N No<br>C Yes with the <i>cascaded</i> semantic<br>Y Yes with the <i>local</i> semantic<br><br>The value is N if the view has no WHERE clause. | G   |
| IBMREQD     | CHAR(1)<br>NOT NULL                      | A value of Y indicates that the row came from the basic machine-readable material (MRM) tape. For all other values, see "Release dependency indicators" on page 1005.                                                                  | G   |
| TEXT        | VARCHAR(254)<br>NOT NULL                 | Text or portion of the text of the CREATE VIEW statement.                                                                                                                                                                              | G   |
| PATHSCHEMAS | VARCHAR(254)<br>NOT NULL WITH<br>DEFAULT | SQL path at the time the view was defined. The path is used to resolve unqualified data type and function names used in the view definition.                                                                                           | G   |
| RELCREATED  | CHAR(1)<br>NOT NULL WITH<br>DEFAULT      | Release of DB2 that was used to create the object:<br><br>blank Created prior to Version 7.<br>K Created on Version 7                                                                                                                  | G   |
| # TYPE      | CHAR(1)<br>NOT NULL WITH<br>DEFAULT 'V'  | Type of table:<br><br>F SQL function<br>V View                                                                                                                                                                                         | G   |

## SYSIBM.SYSVOLUMES

### SYSIBM.SYSVOLUMES table

Contains one row for each volume of each storage group.

| Column name | Data type           | Description                                                                                                                                                           | Use |
|-------------|---------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| SGNAME      | CHAR(8)<br>NOT NULL | Name of the storage group.                                                                                                                                            | G   |
| SGCREATOR   | CHAR(8)<br>NOT NULL | Authorization ID of the owner of the storage group.                                                                                                                   | G   |
| VOLID       | CHAR(6)<br>NOT NULL | Serial number of the volume or * if SMS-managed.                                                                                                                      | G   |
| IBMREQD     | CHAR(1)<br>NOT NULL | A value of Y indicates that the row came from the basic machine-readable material (MRM) tape. For all other values, see “Release dependency indicators” on page 1005. | G   |

---

## SYSIBM.USERNAMES table

Each row in the table is used to carry out one of the following operations:

- Outbound ID translation
- Inbound ID translation and “come from” checking

Rows in this table can be inserted, updated, and deleted.

| Column name | Data type                               | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                | Use |
|-------------|-----------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| TYPE        | CHAR(1)<br>NOT NULL                     | How the row is to be used:<br><br>O For outbound translation.<br>I For inbound translation and “come from” checking.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       | G   |
| AUTHID      | CHAR(8)<br>NOT NULL WITH<br>DEFAULT     | Authorization ID to be translated. Applies to any authorization ID if blank.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               | G   |
| LINKNAME    | CHAR(8)<br>NOT NULL                     | Identifies the VTAM or TCP/IP network locations associated with this row. A blank value in this column indicates this name translation rule applies to any TCP/IP or SNA partner.<br><br>If a nonblank LINKNAME is specified, one or both of the following statements must be true: <ul style="list-style-type: none"> <li>• A row exists in SYSIBM.LUNAMES whose LUNAME matches the value specified in the SYSIBM.USERNAMES LINKNAME column. This row specifies the VTAM site associated with this name translation rule.</li> <li>• A row exists in SYSIBM.IPNAMES whose LINKNAME matches the value specified in the SYSIBM.USERNAMES LINKNAME column. This row specifies the TCP/IP host associated with this name translation rule.</li> </ul> Inbound name translation and “come from” checking are not performed for TCP/IP clients. | G   |
| NEWAUTHID   | CHAR(8)<br>NOT NULL WITH<br>DEFAULT     | Translated value of AUTHID. Blank specifies no translation.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                | G   |
| PASSWORD    | CHAR(8)<br>NOT NULL WITH<br>DEFAULT     | Password to accompany an outbound request, if passwords are not encrypted. If passwords are encrypted, or the row is for inbound requests, the column is not used.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         | G   |
| IBMREQD     | CHAR(1)<br>NOT NULL WITH<br>DEFAULT 'N' | A value of Y indicates that the row came from the basic machine-readable material (MRM) tape. For all other values, see “Release dependency indicators” on page 1005.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      | G   |

## SYSIBM.USERNAMES

---

## Appendix E. Using the catalog in database design

The information in this chapter is General-use Programming Interface and Associated Guidance Information, as defined in Appendix H, "Notices," on page 1173.

Retrieving information from the catalog, using SQL statements, can be helpful in designing your relational database. Appendix D of *DB2 SQL Reference* lists all the DB2 catalog tables and the information stored in them.

The information in the catalog is vital to normal DB2 operation. As the examples in this chapter show, you can *retrieve* catalog information, but *changing* it can have serious consequences. Therefore you cannot execute INSERT or DELETE statements that affect the catalog, and only a limited number of columns exist that you can update. Exceptions to these restrictions are the SYSIBM.SYSSTRINGS, SYSIBM.SYSPROCEDURES, SYSIBM.SYSCOLDIST, and SYSIBM.SYSCOLDISTSTATS catalog tables, into which you can insert rows and proceed to update and delete rows.

To execute the following examples, you need at least the SELECT privilege on the appropriate catalog tables. Be careful when querying the DB2 catalog because some catalog queries can result in long table space scans.

---

### Retrieving catalog information about DB2 storage groups

SYSIBM.SYSSTOGROUP and SYSIBM.SYSVOLUMES contain information about DB2 storage groups and the volumes in those storage groups. The following query shows what volumes are in a DB2 storage group, how much space is used, and when that space was last calculated.

```
SELECT SGNAME, VOLID, SPACE, SPCDATE
 FROM SYSIBM.SYSVOLUMES, SYSIBM.SYSSTOGROUP
 WHERE SGNAME=NAME
 ORDER BY SGNAME;
```

---

### Retrieving catalog information about a table

SYSIBM.SYSTABLES contains a row for every table, view, and alias in your DB2 system. Each row tells you whether the object is a table, a view, or an alias, its name, who created it, what database it belongs to, what table space it belongs to, and other information. SYSTABLES also has a REMARKS column in which you can store your own information about the table in question. See "Adding and retrieving comments" on page 1151 for more information about how to do this.

This statement displays all the information for the project activity sample table:

```
SELECT *
 FROM SYSIBM.SYSTABLES
 WHERE NAME = 'PROJECT'
 AND CREATOR = 'DSN8710';
```

---

### Retrieving catalog information about aliases

SYSIBM.SYSTABLES describes the aliases you create. It has three columns used only for aliases:

- LOCATION contains your subsystem's location name for the remote system, if the object on which the alias is defined resides at a remote subsystem.

- TBCREATOR contains the owner of the table or view.
- TBNAME contains the name of the table or the view.

These sample user-defined functions make it easy to find information about aliases. See *DB2 SQL Reference* for more information.

- TABLE\_NAME returns the name of a table, view, or undefined object found after resolving aliases for a user-specified object.
- TABLE\_SCHEMA returns the schema name of a table, view, or undefined object found after resolving aliases for a user-specified object.
- TABLE\_LOCATION returns the location name of a table, view, or undefined object found after resolving aliases for a user-specified object.

The NAME and CREATOR columns of SYSTABLES contain the name and owner of the alias, and three other columns contain the following information for aliases:

- TYPE is A.
- DBNAME is DSNDDB06.
- TSNAME is SYSDBAUT.

If similar tables at different locations have names with the same second and third parts, you can retrieve the aliases for them with a query like this one:

```
SELECT LOCATION, CREATOR, NAME
 FROM SYSIBM.SYSTABLES
 WHERE TBCREATOR='DSN8710' AND TBNAME='EMP'
 AND TYPE='A';
```

## **Retrieving catalog information about columns**

SYSIBM.SYSCOLUMNS has one row for each column of every table and view. Query it, for example, if you cannot remember the column names of a table or view.

This statement retrieves information about columns in the sample department table:

```
SELECT NAME, TBNAME, COLTYPE, LENGTH, NULLS, DEFAULT
 FROM SYSIBM.SYSCOLUMNS
 WHERE TBNAME='DEPT'
 AND TBCREATOR = 'DSN8710';
```

The result is shown below; for each column, the following information about each column is given:

- The column name
- The name of the table that contains it
- Its data type
- Its length attribute
- Whether it allows nulls
- Whether it allows default values

| NAME     | TBNAME | COLTYPE | LENGTH | NULLS | DEFAULT |
|----------|--------|---------|--------|-------|---------|
| DEPTNO   | DEPT   | CHAR    | 3      | N     | N       |
| DEPTNAME | DEPT   | VARCHAR | 36     | N     | N       |
| MGRNO    | DEPT   | CHAR    | 6      | Y     | N       |
| ADMRDEPT | DEPT   | CHAR    | 3      | N     | N       |

For LOB columns, the LENGTH column shows the length of the pointer to the LOB. For an example of a query showing the actual LOB length, see “Retrieving catalog information about LOBs” on 1149.

---

## Retrieving catalog information about indexes

SYSIBM.SYSINDEXES contains a row for every index, including indexes on catalog tables. This example retrieves a row about an index named XEMPL2.

```
SELECT *
 FROM SYSIBM.SYSINDEXES
 WHERE NAME = 'XEMPL2'
 AND CREATOR = 'DSN8710';
```

A table can have more than one index. To display information about all the indexes of a table, enter a statement like this one:

```
SELECT *
 FROM SYSIBM.SYSINDEXES
 WHERE TBNAME = 'EMP'
 AND TBCREATOR = 'DSN8710';
```

---

## Retrieving catalog information about views

For every view you create, DB2 stores descriptive information in several catalog tables. The following actions occur in the catalog after the execution of CREATE VIEW:

- A row is inserted into SYSIBM.SYSTABLES.
- A row is inserted into SYSIBM.SYSTABAUTH to record the owner's privileges on the view.
- For each column of the view, a row is inserted into SYSIBM.SYSCOLUMNS.
- One or more rows are inserted into the SYSIBM.SYSVIEWS table to record the text of the CREATE VIEW statement.
- For each table or view on which the view is dependent, a row is inserted into SYSIBM.SYSVIEWDEP to record the dependency.
- A row is inserted into SYSIBM.SYSVTREE, and possibly into SYSIBM.SYSLVTREE, to record the parse tree of the view (an internal representation of its logic).

Users might want a view of one or more of those tables, containing information about their own tables and views.

---

## Retrieving catalog information about authorizations

SYSIBM.SYSTABAUTH contains information about the privileges held by authorization IDs over tables and views. Query it to learn who can access your data. The following query retrieves the names of all users who have been granted access to the DSN8710.DEPT table.

```
SELECT GRANTEE
 FROM SYSIBM.SYSTABAUTH
 WHERE TTNAME = 'DEPT'
 AND GRANTEETYPE <> 'P'
 AND TCREATOR = 'DSN8710';
```

GRANTEE is the name of the column that contains authorization IDs for users of tables. TTNAME and TCREATOR specify the DSN8710.DEPT table. The clause GRANTEETYPE <> 'P' ensures that you retrieve the names only of users (not application plans or packages) that have authority to access the table.

---

## Retrieving catalog information about parent keys

SYSIBM.SYSCOLUMNS identifies columns of a parent key in column KEYSEQ; a nonzero value indicates the place of a column in the parent key. To retrieve the creator, database, and names of the columns in the parent key of the sample project activity table using SQL statements, execute:

```
SELECT TBCREATOR, TBNAME, NAME, KEYSEQ
 FROM SYSIBM.SYSCOLUMNS
 WHERE TBCREATOR = 'DSN8710'
 AND TBNAME = 'PROJECT'
 AND KEYSEQ > 0
 ORDER BY KEYSEQ;
```

SYSIBM.SYSINDEXES identifies the primary index of a table by the value P in column UNIQUERULE. To find the name, creator, database, and index space of the primary index on the project activity table, execute:

```
SELECT TBCREATOR, TBNAME, NAME, CREATOR, DBNAME, INDEXSPACE
 FROM SYSIBM.SYSINDEXES
 WHERE TBCREATOR = 'DSN8710'
 AND TBNAME = 'PROJECT'
 AND UNIQUERULE = 'P' ;
```

**Note:** It is not always possible to retrieve information about unique keys created before Version 7. Information can be retrieved for unique keys created in Version 7 and for unique keys created before Version 7 if they are not involved in referential integrity.

---

## Retrieving catalog information about foreign keys

SYSIBM.SYSRELS contains information about referential constraints, and each constraint is uniquely identified by the creator and name of the dependent table and the constraint name (RELNAME). SYSIBM.SYSFOREIGNKEYS contains information about the columns of the foreign key that defines the constraint. To retrieve the constraint name, column names, and parent table names for every relationship in which the project table is a dependent, execute:

```
SELECT A.CREATOR, A.TBNAME, A.RELNAME, B.COLNAME, B.COLSEQ,
 A.REFTBCREATOR, A.REFTBNAME
 FROM SYSIBM.SYSRELS A, SYSIBM.SYSFOREIGNKEYS B
 WHERE A.CREATOR = 'DSN8710'
 AND B.CREATOR = 'DSN8710'
 AND A.TBNAME = 'PROJ'
 AND B.TBNAME = 'PROJ'
 AND A.RELNAME = B.RELNAME
 ORDER BY A.RELNAME, B.COLSEQ;
```

You can use the same tables to find information about the foreign keys of tables to which the project table is a parent, as follows:

```
SELECT A.RELNAME, A.CREATOR, A.TBNAME, B.COLNAME, B.COLNO
 FROM SYSIBM.SYSRELS A, SYSIBM.SYSFOREIGNKEYS B
 WHERE A.REFTBCREATOR = 'DSN8710'
 AND A.REFTBNAME = 'PROJ'
 AND A.RELNAME = B.RELNAME
 ORDER BY A.RELNAME, B.COLNO;
```

---

## Retrieving catalog information about check pending

SYSIBM.SYSTABLESPACE indicates that a table space is in check-pending status by a value in column STATUS: P if the entire table space has that status, S if the status has a scope of less than the entire space. To list all table spaces whose use is restricted for *any* reason, issue this command:

```
-DISPLAY DATABASE (*) SPACENAM(*) RESTRICT
```

To retrieve the names of table spaces in check-pending status only, with the names of the tables they contain, execute:

```
SELECT A.DBNAME, A.NAME, B.CREATOR, B.NAME
 FROM SYSIBM.SYSTABLESPACE A, SYSIBM.SYSTABLES B
 WHERE A.DBNAME = B.DBNAME
 AND A.NAME = B.TSNAME
 AND (A.STATUS = 'P' OR A.STATUS = 'S')
 ORDER BY 1, 2, 3, 4;
```

---

## Retrieving catalog information about table check constraints

Information about check constraints is stored in the DB2 catalog in:

- SYSIBM.SYSCHECKS, which contains one row for each check constraint defined on a table
- SYSIBM.SYSCHECKDEP, which contains one row for each reference to a column in a check constraint

The following query shows all table check constraints on all tables named SIMPDEPT and SIMPEMPL in order by column name within table owner. It shows the name, authorization ID of the creator, and text for each constraint. A constraint that uses more than one column name appears more than once in the result.

```
CREATE TABLE SIMPDEPT
 (DEPTNO CHAR(3) NOT NULL,
 DEPTNAME VARCHAR(12) CONSTRAINT CC1 CHECK (DEPTNAME IS NOT NULL),
 MGRNO CHAR(6),
 MGRNAME CHAR(6));
SELECT A.TBOWNER, A.TBNAME, B.COLNAME,
 A.CHECKNAME, A.CREATOR, A.CHECKCONDITION
 FROM SYSIBM.SYSCHECKS A, SYSIBM.SYSCHECKDEP B
 WHERE A.TBOWNER = B.TBOWNER
 AND A.TBNAME = B.TBNAME
 AND B.TBNAME = 'SIMPDEPT'
 AND A.CHECKNAME = B.CHECKNAME
 ORDER BY TBOWNER, TBNAME, COLNAME;
```

---

## Retrieving catalog information about LOBs

SYSIBM.SYSAUXRELS contains information about the relationship between a base table and an auxiliary table. For example, this query returns information about the name of the LOB columns for the employee table and its associated auxiliary table owner and name:

```
SELECT COLNAME, PARTITION, AUXTBOWNER, AUXTBNAME
 FROM SYSIBM.SYSAUXRELS
 WHERE TBNAME = 'EMP' AND TBOWNER = 'DSN8710';
```

Information about the length of a LOB is in the LENGTH2 column of SYSCOLUMNS. You can query information about the length of the column as it is returned to an application with this query:

```
SELECT NAME, TBNAME, COLTYPE, LENGTH2, NULLS, DEFAULT
 FROM SYSIBM.SYSCOLUMNS
 WHERE TBNAME='DEPT'
 AND TBCREATOR = 'DSN8710';
```

---

## Retrieving catalog information about user-defined functions and stored procedures

SYSIBM.SYSROUTINES describes user-defined functions and stored procedures. You can use this example to find packages with stored procedure that were created prior to Version 6 and then migrated to SYSIBM.SYSROUTINES:

```
SELECT SCHEMA, NAME FROM SYSIBM.SYSROUTINES
 WHERE ROUTINETYPE = 'P';
```

You can use this query to retrieve information about user-defined functions:

```
SELECT SCHEME, NAME, FUNCTION_TYPE, PARM_COUNT FROM SYSIBM.SYSROUTINES
 WHERE ROUTINETYPE='F';
```

Stored procedures created before Version 6 have different authorization requirements than those created in Version 6. See *DB2 Application Programming and SQL Guide* for more information.

---

## Retrieving catalog information about triggers

SYSIBM.SYSTRIGGERS contains information about the triggers defined in your databases. To find all the triggers defined on a particular table, their characteristics, and to determine the order they are fired in, issue this query:

```
SELECT DISTINCT SCHEMA, NAME, TRIGTIME, TRIGEVENT, GRANULARITY, CREADEDTS
 FROM SYSIBM.SYSTRIGGERS
 WHERE TBNAME = 'EMP' AND TBOWNER = 'DSN8710';
```

Issue this query to retrieve the text of a particular trigger:

```
SELECT TEXT, SEQNO
 FROM SYSIBM.SYSTRIGGERS
 WHERE SCHEMA = schema_name
 AND NAME = trigger_name
 ORDER BY SEQNO;
```

Or to determine triggers that must be rebound because they are invalidated after objects are dropped or altered, issue this query:

```
SELECT COLLID, NAME
 FROM SYSIBM.SYSPACKAGE
 WHERE TYPE = 'T'
 AND (VALID = 'N' OR OPERATIVE = 'N');
```

---

## Retrieving catalog information about distinct types

Information about distinct types is in SYSIBM.SYSDATATYPES and SYSIBM.SYSRESAUTH. This query returns the source of a distinct type:

```
SELECT SOURCESCHEMA, SOURCETYPE
 FROM SYSIBM.SYSDATATYPES
 WHERE NAME = 'MONEY' AND SCHEMA = 'USER1B';
```

Issue this query to determine privileges on distinct types for user USER1B:

```
SELECT GRANTOR, NAME, DATEGRANTED
 FROM SYSIBM.SYSRESAUTH
 WHERE OBTYPE = 'D' AND GRANTEE = 'USER1B';
```

---

## Adding and retrieving comments

After you create a table, view, index, or alias, you can provide explanatory information about it for future reference—information such as the purpose of the table, who uses it, and anything unusual about it. You can store a comment about the table or the view as a whole, and you can also include a comment for *each column*. You can also create comments on user-defined functions, stored procedures, and triggers. A comment must not exceed 254 bytes.

A comment is especially useful if your names do not clearly indicate the contents of columns or tables. In that case, use a comment to describe the specific contents of the column or table.

Below are two examples of the COMMENT statement:

```
COMMENT ON TABLE DSN8710.EMP IS
 'Employee table. Each row in this table represents one
 employee of the company.';
COMMENT ON COLUMN DSN8710.PROJ.PRSTDATE IS
 'Estimated project start date. The format is DATE.';
```

After you execute a COMMENT statement, your comments are stored in the REMARKS column of SYSIBM.SYSTABLES or SYSIBM.SYSCOLUMNS. (Any comment that is already present in the row is replaced by the new one.) The next two examples retrieve the comments that are added by the previous COMMENT statements.

```
SELECT REMARKS
 FROM SYSIBM.SYSTABLES
 WHERE NAME = 'EMP'
 AND CREATOR = 'DSN8710';
SELECT REMARKS
 FROM SYSIBM.SYSCOLUMNS
 WHERE NAME = 'PRSTDATE' AND TBNAME = 'PROJ'
 AND TBCREATOR = 'DSN8710';
```

---

## Verifying the accuracy of the database definition

You can use the catalog to verify the accuracy of your database definition. After you have created the objects in your database, display selected information from the catalog to check that no errors are in your CREATE statements. By examining the catalog tables, you can verify that your tables are in the correct table space, your table spaces are in the correct storage group, and so on.



---

## Appendix F. SQL reserved words

Table 84 on page 1154 lists the words that cannot be used as ordinary identifiers in some contexts because they might be interpreted as SQL keywords. For example, ALL cannot be a column name in a SELECT statement. Each word, however, can be used as a delimited identifier in contexts where it otherwise cannot be used as an ordinary identifier. For example, if the quotation mark ("") is the escape character that begins and ends delimited identifiers, "ALL" can appear as a column name in a SELECT statement. In addition, some sections of this book might indicate words that cannot be used in the specific context that is being described.

## SQL reserved words

Table 84. SQL reserved words

|              |                       |              |              |            |
|--------------|-----------------------|--------------|--------------|------------|
| ADD          | CURRENT_TIMESTAMP     | GOTO         | NULL         | SECQTY     |
| AFTER        | CURSOR                | GRANT        | NULLS        | SECURITY   |
| ALL          | DATA                  | GROUP        | NUMPARTS     | SELECT     |
| ALLOCATE     | DATABASE              | HANDLER      | OBID         | SENSITIVE  |
| ALLOW        | DAY                   | HAVING       | OF           | SET        |
| ALTER        | DAYS                  | HOUR         | ON           | SIMPLE     |
| AND          | DBINFO                | HOURS        | OPEN         | SOME       |
| ANY          | DB2SQL                | IF           | OPTIMIZATION | SOURCE     |
| AS           | DECLARE               | IMMEDIATE    | OPTIMIZE     | SPECIFIC   |
| ASSOCIATE    | DEFAULT               | IN           | OR           | STANDARD   |
| ASUTIME      | DELETE                | INDEX        | ORDER        | STATIC     |
| AUDIT        | DESCRIPTOR            | INHERIT      | OUT          | STAY       |
| AUX          | DETERMINISTIC         | INNER        | OUTER        | STOGROUP   |
| AUXILIARY    | DISALLOW              | INOUT        | PACKAGE      | STORES     |
| BEFORE       | DISTINCT              | INSENSITIVE  | PARAMETER    | STYLE      |
| BEGIN        | DO                    | INSERT       | PART         | SUBPAGES   |
| BETWEEN      | DOUBLE                | INTO         | PATH         | SYNONYM    |
| BUFFERPOOL   | DROP                  | IS           | PIECESIZE    | SYSFUN     |
| BY           | DSNHATTR              | ISOBID       | PLAN         | SYSIBM     |
| CALL         | DSIZE                 | JAR          | PRECISION    | SYSPROC    |
| CAPTURE      | DYNAMIC               | JAVA         | PREPARE      | SYSTEM     |
| CASCADED     | EDITPROC              | JOIN         | PRIQTY       | TABLE      |
| CASE         | ELSE                  | KEY          | PRIVILEGES   | TABLESPACE |
| CAST         | ELSEIF                | LABEL        | PROCEDURE    | THEN       |
| CCSID        | ENCODING              | LANGUAGE     | PROGRAM      | TO         |
| CHAR         | END                   | LC_CTYPE     | PSID         | TRIGGER    |
| CHARACTER    | END-EXEC <sup>1</sup> | LEAVE        | QUERYNO      | UNDO       |
| CHECK        | ERASE                 | LEFT         | READS        | UNION      |
| CLOSE        | ESCAPE                | LIKE         | REFERENCES   | UNIQUE     |
| CLUSTER      | EXCEPT                | LOCAL        | RELEASE      | UNTIL      |
| COLLECTION   | EXECUTE               | LOCALE       | RENAME       | UPDATE     |
| COLLID       | EXISTS                | LOCATOR      | REPEAT       | USER       |
| COLUMN       | EXIT                  | LOCATORS     | RESTRICT     | USING      |
| COMMENT      | EXTERNAL              | LOCK         | RESULT       | VALIDPROC  |
| COMMIT       | FENCED                | LOCKMAX      | RESULT_SET_  | VALUES     |
| CONCAT       | FETCH                 | LOCKSIZE     | LOCATOR      | VARIANT    |
| CONDITION    | FIELDPROC             | LONG         | RETURN       | VCAT       |
| CONNECT      | FINAL                 | LOOP         | RETURNS      | VIEW       |
| CONNECTION   | FOR                   | MICROSECOND  | REVOKE       | VOLUMES    |
| CONSTRAINT   | FROM                  | MICROSECONDS | RIGHT        | WHEN       |
| CONTAINS     | FULL                  | MINUTE       | ROLLBACK     | WHERE      |
| CONTINUE     | FUNCTION              | MINUTES      | RUN          | WHILE      |
| CREATE       | GENERAL               | MODIFIES     | SAVEPOINT    | WITH       |
| CURRENT      | GENERATED             | MONTH        | SCHEMA       | WLM        |
| CURRENT_DATE | GET                   | MONTHS       | SCRATCHPAD   | YEAR       |
| CURRENT_LC_  | GLOBAL                | NO           | SECOND       | YEARS      |
| CTYPE        | GO                    | NOT          | SECONDS      |            |
| CURRENT_PATH |                       |              |              |            |
| CURRENT_TIME |                       |              |              |            |

Note: <sup>1</sup>COBOL only

#

APPLICATION, NAME, and TYPE are no longer reserved words.

IBM SQL has additional reserved words that DB2 for OS/390 and z/OS does not enforce. Therefore, it would be best to not use these additional reserved words as ordinary identifiers in names that have a continuing use. See *IBM SQL Reference* for a list of the words.

## Appendix G. Sample user-defined functions

This appendix describes the sample user-defined functions that are provided with DB2. You can use the functions in the following ways:

- In your applications just as you would use other user-defined functions. Use the functions only if installation job DSNTEJ2U, which prepares the functions for use, has been run. Because the external programs that implement the logic of the sample functions are written in C and C++, the installation job requires that your site has IBM C/C++ for OS/390. For information on installation job DSNTEJ2U, see *DB2 Installation Guide*.
- As examples to help you define and implement your own user-defined functions. Data set *prefix.SDSNSAMP* contains the code for the sample functions.

Table 85 lists the sample user-defined functions. The detailed descriptions of the functions that follow the table include their external program names and specific names. The functions are in schema DSN8.

Table 85. DB2 sample user-defined functions

| Function Name  | Description                                                                                               | Page |
|----------------|-----------------------------------------------------------------------------------------------------------|------|
| ALTDATE        | Returns the current date or a user-specified date in a user-specified format                              | 1156 |
| ALTTIME        | Returns the current time or a user-specified time in a user-specified format                              | 1159 |
| CURRENCY       | Returns a floating-point number as a currency value                                                       | 1161 |
| DAYNAME        | Returns the name of the day of the week on which a date in ISO format falls                               | 1163 |
| MONTHNAME      | Returns the name of the month in which a date in ISO format falls                                         | 1164 |
| TABLE_LOCATION | Returns the location name of a table or view after resolving any aliases                                  | 1165 |
| TABLE_NAME     | Returns the unqualified name of a table or view after resolving any aliases                               | 1167 |
| TABLE_SCHEMA   | Returns the schema name of a table or view after resolving any aliases                                    | 1169 |
| WEATHER        | Shows how to use a user-defined table function to make non-relational data available for SQL manipulation | 1171 |

**ALTDAT**

```
►►ALTDAT([input date, input format,] output format)►►
```

The schema is DSN8.

The ALTDAT function returns the current date in one of the following formats or converts a user-specified date from one format to another:

| D MONTH YY | D MONTH YYYY | DD MONTH YY | DD MONTH YYYY |
|------------|--------------|-------------|---------------|
| D.M.YY     | D.M.YYYY     | DD.MM.YY    | DD.MM.YYYY    |
| D-M-YY     | D-M-YYYY     | DD-MM-YY    | DD-MM-YYYY    |
| D/M/YY     | D/M/YYYY     | DD/MM/YY    | DD/MM/YYYY    |
| M/D/YY     | M/D/YYYY     | MM/DD/YY    | MM/DD/YYYY    |
| YY/M/D     | YYYY/M/D     | YY/MM/DD    | YYYY/MM/DD    |
| YY.M.D     | YYYY.M.D     | YY.MM.DD    | YYYY.MM.DD    |
|            | YYYY-M-D     |             | YYYY-MM-DD    |
|            | YYYY-D-XX    |             | YYYY-DD-XX    |
|            | YYYY-XX-D    |             | YYYY-XX-DD    |

where:

D: Suppress leading zero if the day is less than 10  
DD: Retain leading zero if the day is less than 10  
M: Suppress leading zero if the month is less than 10  
MM: Retain leading zero if the month is less than 10  
MONTH: Use English-language name of month  
XX: Use a capital Roman numeral for month  
YY: Use a year format without century  
YYYY: Use a year format with century

The ALTDAT function demonstrates how you can create an overloaded function—a function name for which there are multiple function instances. Each instance supports a different parameter list enabling you to group related but distinct functions in a single user-defined function. The ALTDAT function has two forms.

**Form 1: ALTDAT(*output format*)**

This form of the function converts the current date into the specified format.

*output format*

A character string that matches one of the 34 date formats that are shown above. The character string must have a data type of VARCHAR and an actual length that is not greater than 13 bytes.

The result of the function has a VARCHAR data type and an actual length that is not greater than 17 bytes.

**Form 2: ALTDAT(*input date*, *input format*, *output format*)**

This form of the function converts a date (*input date*) in one user-specified format (*input format*) into another format (*output format*).

*input date*

The argument must be a date or a character string representation of a date in the format specified by *input format*. The character string must have a data type of VARCHAR and an actual length that is not greater than 17 bytes.

*input format*

A character string that matches one of the 34 date formats that are shown above. The character string must have a data type of VARCHAR and an actual length that is not greater than 13 bytes.

*output format*

A character string that matches one of the 34 date formats that are shown above. The character string must have a data type of VARCHAR and an actual length that is not greater than 13 bytes.

The result of the function has a VARCHAR data type and an actual length that is not greater than 17 bytes.

Table 86 shows the external and specific names for the two forms of the function, which are based on the input to the function.

*Table 86. External program and specific names for ALTDAT*

| Conversion type     | Input arguments                                                                                | External name | Specific name    |
|---------------------|------------------------------------------------------------------------------------------------|---------------|------------------|
| Current date        | <i>Output format</i> (VARCHAR)                                                                 | DSN8DUAD      | DSN8.DSN8DUADV   |
| User-specified date | <i>Input date</i> (VARCHAR)<br><i>Input format</i> (VARCHAR)<br><i>Output format</i> (VARCHAR) | DSN8DUCD      | DSN8.DSN8DUCDVVV |
|                     | <i>Input date</i> (DATE)<br><i>Input format</i> (VARCHAR)<br><i>Output format</i> (VARCHAR)    | DSN8DUCD      | DSN8.DSN8DUCDDVV |

*Example 1:* Convert the current date into format 'DD MONTH YY', a format that will include any leading zero for the month, the name of the month in English, and the year without the two digits for the century.

```
VALUES DSN8.ALTDAT('DD MONTH YY');
```

*Example 2:* Convert the current date into format 'D.M.YYYY', a format that will suppress any leading zero for the day or month and include the year with the century.

```
VALUES DSN8.ALTDAT('D.M.YYYY');
```

*Example 3:* Convert the current date into format 'YYYY-XX-DD', a format that will include the century, the month of the year as a roman numeral, and the day of the month with any leading zero.

```
VALUES DSN8.ALTDAT('YYYY-XX-DD');
```

*Example 4:* Convert a date in the format of 'DD MONTH YYYY' to a date in the format of 'YYYY/MM/DD'.

```
VALUES DSN8.ALTDAT('11 November 1918',
 'DD MONTH YYYY',
 'YYYY/MM/DD');
```

The result of the above example is 1918/11/18.

*Example 5:* Convert the date that employee 000130 was hired, a date in ISO format, into the format of 'D.M.YY'.

```
SELECT FIRSTNAME || ' '
 || LASTNAME || ' was hired on '
```

## ALTDAT

```
|| DSN8.ALTDAT(HIREDATE,
 'YYYY-MM-DD',
 'D.M.YY')
FROM EMP
WHERE EMPNO = '000130';
```

Assuming that the HIREDATE is 1971-07-28, the above example returns: DELORES QUINTANA was hired on 28.7.71.

## ALTTIME

```
►►—ALTTIME([input time, input format,] output format)—►►
```

The schema is DSN8.

The ALTTIME function returns the current time in one of the following formats or converts a user-specified time from one of the formats to another:

|                |             |
|----------------|-------------|
| H:MM AM/PM     | HH:MM AM/PM |
| HH:MM:SS AM/PM | HH:MM:SS    |
| H.MM           | HH.MM       |
| H.MM.SS        | HH.MM.SS    |

where:

|        |                                                     |
|--------|-----------------------------------------------------|
| H:     | Suppress leading zero if the hour is less than 10   |
| HH:    | Retain leading zero if the hour is less than 10     |
| M:     | Suppress leading zero if the minute is less than 10 |
| MM:    | Retain leading zero if the minute is less than 10   |
| AM/PM: | Return time in 12-hour clock format, else 24-hour   |

The ALTTIME function demonstrates how you can create an overloaded function—a function name for which there are multiple function instances. Each instance supports a different parameter list enabling you to group related but distinct functions in a single user-defined function. The ALTIME function has two forms.

### Form 1: ALTTIME(*output format*)

This form of the function converts the current time into the specified format.

#### *output format*

A character string that matches one of the 8 time formats that are shown above. The character string must have a data type of VARCHAR and an actual length that is not greater than 14 bytes.

The result of the function has a VARCHAR data type and an actual length that is not greater than 11 bytes.

### Form 2: ALTTIME(*input time, input format, output format*)

This form of the function converts a time (*input date*) in one user-specified format (*input format*) into another format (*output format*).

#### *input time*

The argument must be a time or a character string representation of a time in the format specified by *input format*. A character string argument must have a data type of VARCHAR and an actual length that is not greater than 11 bytes.

#### *input format*

A character string that matches one of the 8 time formats that are shown above. The character string must have a data type of VARCHAR and an actual length that is not greater than 14 bytes.

#### *output format*

A character string that matches one of the 8 time formats that are shown above. The character string must have a data type of VARCHAR and an actual length that is not greater than 14 bytes.

## ALTTIME

The result of the function has a VARCHAR data type and an actual length that is not greater than 11 bytes.

Table 87 shows the external program and specific names for the two forms of the function, which are based on the input to the function.

*Table 87. External and specific names for ALTTIME*

| Conversion type     | Input arguments                                                                                | External name | Specific name    |
|---------------------|------------------------------------------------------------------------------------------------|---------------|------------------|
| Current time        | <i>Output format</i> (VARCHAR)                                                                 | DSN8DUAT      | DSN8.DSN8DUATV   |
| User-specified time | <i>Input time</i> (VARCHAR)<br><i>Input format</i> (VARCHAR)<br><i>Output format</i> (VARCHAR) | DSN8DUCT      | DSN8.DSN8DUCTVVV |
|                     | <i>Input date</i> (TIME)<br><i>Input format</i> (VARCHAR)<br><i>Output format</i> (VARCHAR)    | DSN8DUCT      | DSN8.DSN8DUCTTVV |

*Example 1:* Convert the current time into a 12-hour clock format without seconds, 'H.MM AM/PM'.

```
VALUES DSN8.ALTTIME('H:MM AM/PM');
```

*Example 2:* Convert the current time into a 24-hour clock format without seconds, 'HH.MM'.

```
VALUES DSN8.ALTTIME('HH.MM');
```

*Example 3:* Convert the current time into a 24-hour clock format with seconds, 'HH.MM.SS'.

```
VALUES DSN8.ALTTIME('HH.MM.SS');
```

*Example 4:* Convert '00:00:00', a time in 24-hour clock format with seconds, to a time in 12-hour clock format without seconds.

```
VALUES DSN8.ALTTIME('00:00:00', 'HH:MM:SS', 'HH:MM AM/PM');
```

The function returns 12:00 AM.

*Example 5:* Convert '00:00:00', a time in 24-hour clock format with seconds, to a time in 12-hour clock format without seconds and without any leading zero on the hour.

```
VALUES DSN8.ALTTIME('06.42.37', 'HH.MM.SS', 'H:MM AM/PM');
```

The function returns 6:42 AM.

## CURRENCY

```
►►—CURRENCY(—input amount, currency symbol
 [, credit/debit indicator])—►►
```

The schema is DSN8.

The CURRENCY function returns a value that is formatted as an amount with a user-specified currency symbol and, if specified, one of three symbols that indicate debit or credit.

*input amount*

An expression that specifies the value to be formatted. The expression must be a floating-point value.

*currency symbol*

A character string that specifies the currency symbol. The string must have a data type of VARCHAR and an actual length that is not greater than 2 bytes.

*credit/debit indicator*

A character string that specifies the symbol that is included with the result to indicate whether the value is negative or positive. The string must have a data type of VARCHAR and an actual length that is not greater than 5 bytes. If *credit/debit indicator* is not specified or is the value null, the result is formatted without an indicator symbol. You can specify the following symbols:

**CR/DB**

*Bank style*. Negative input values are appended with "DB"; positive input values are appended with "CR".

**+/-** *Arithmetic style*. Negative input values are prefixed with a minus sign "-"; positive values are formatted without symbols.

**(/)** *Accounting style*. Negative input values are enclosed in parentheses "()" ; positive values are formatted without symbols.

The result of the function is a character string with a data type of VARCHAR and an actual length that is not greater than 19 bytes.

The CURRENCY function uses the C language functions *strfmon* to facilitate formatting of money amounts and *setlocale* to initialize strfmon for local conventions. If setlocale fails, the CURRENCY function returns an error.

Table 88 shows the external program and specific names for CURRENCY. The specific names differ depending on the input to the function.

*Table 88. External program and specific names for CURRENCY*

| Input arguments                                                                | External name | Specific name    |
|--------------------------------------------------------------------------------|---------------|------------------|
| <i>input amount</i><br><i>currency symbol</i>                                  | DSN8DUCY      | DSN8.DSN8DUCYFV  |
| <i>input amount</i><br><i>currency symbol</i><br><i>debit/credit indicator</i> | DSN8DUCY      | DSN8.DSN8DUCYFVV |

## CURRENCY

*Example 1:* Express -1234.56 as an amount in US dollars, using the bank style debit/credit indicator to indicate whether the value is negative or positive.

```
VALUES DSN8.CURRENCY(-1234.56,'$', 'CR/DB');
```

The result of the function is \$1,234.56 DB.

*Example 2:* Express -1234.56 as an amount in Deutsche marks, using the accounting style debit/credit indicator to indicate whether the value is negative or positive.

```
VALUES DSN8.CURRENCY(-1234.56,'DM','(/)');
```

The result of the function is (DM 1,234.56).

*Example 3:* Express -1234.56 as an amount in Canadian dollars, using the accounting style debit/credit indicator to indicate whether the value is negative or positive.

```
VALUES DSN8.CURRENCY(-1234.56,'CD','+/-');
```

The result of the function is -CD 1,234.56.

## DAYNAME

►►—DAYNAME(*input date*)—►►

The schema is DSN8.

The DAYNAME function returns the name of the weekday on which a given date falls. The name is returned in English.

*input date*

A valid date or valid character string representation of a date. A character string representation. The string must have a data type of VARCHAR and an actual length that is not greater than 10 bytes. The date must be in ISO format.

The result of the function is a character string with a data type of VARCHAR and an actual length that is not greater than 9 bytes.

The DAYNAME function uses the IBM C++ class *IDate*.

Table 89 shows the external and specific names for DAYNAME. The specific names differ depending on the data type of the input argument.

Table 89. External and specific names for DAYNAME

| Input arguments             | External name | Specific name  |
|-----------------------------|---------------|----------------|
| <i>input date</i> (VARCHAR) | DSN8EUDN      | DSN8.DSN8EUDNV |
| <i>input date</i> (DATE)    | DSN8EUDN      | DSN8.DSN8EUDND |

*Example 1:* For the current date, find the day of the week.

```
VALUES DSN8.DAYNAME(CURRENT DATE);
```

*Example 2:* Find the day of the week on which leap year falls in the year 2000.

```
VALUES DSN8.DAYNAME('2000-02-29');
```

The result of the function is Tuesday.

*Example 3:* Find the day of the week on which Delores Quintana, employee number 000130, was hired.

```
SELECT FIRSTNME || ' '
 || LASTNAME || ' was hired on '
 || DSN8.DAYNAME(HIREDATE) || ', '
 || CHAR(HIREDATE)
 FROM EMP
 WHERE EMPNO = '000130';
```

The result of the function is DELORES QUINTANA was hired on Wednesday, 1971-07-28.

## MONTHNAME

### MONTHNAME

```
►►MONTHNAME(input date)————►►
```

The schema is DSN8.

The MONTHNAME function returns the calendar name of the month in which a given date falls. The name is returned in English.

#### *input date*

A valid date or valid character string representation of a date. A character string representation must have a data type of VARCHAR and an actual length that is no greater than 10 bytes. The date must be in ISO format.

The result of the function is a character string with a data type of VARCHAR and an actual length that is not greater than 9 bytes.

The MONTHNAME function uses the IBM C++ class *IDate*.

Table 90 shows the external and specific names for MONTHNAME. The specific names differ depending on the data type of the input argument.

*Table 90. External and specific names for MONTHNAME*

| Input arguments             | External name | Specific name  |
|-----------------------------|---------------|----------------|
| <i>input date</i> (VARCHAR) | DSN8EUMN      | DSN8.DSN8EUMNV |
| <i>input date</i> (DATE)    | DSN8EUMN      | DSN8.DSN8EUMND |

*Example 1:* For the current date, find the name of the month.

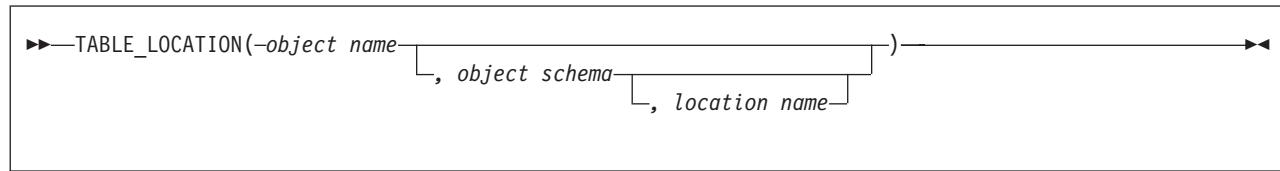
```
VALUES DSN8.MONTHNAME(CURRENT DATE);
```

*Example 2:* Find the month of the year in which Delores Quintana, employee number 000130, was hired.

```
SELECT FIRSTNME || ' '
 || LASTNAME || ' was hired in the month of '
 || DSN8.MONTHNAME(HIREDATE)
 || CHAR(HIREDATE)
 FROM EMP
 WHERE EMPNO = '000130';
```

The result of the function is DELORES QUINTANA was hired in the month of July.

## TABLE\_LOCATION



The schema is DSN8.

The TABLE\_LOCATION function searches for an object and returns the location name of the object after any alias chains have been resolved. The starting point of the resolution is the object that is specified by *object name* and, if specified, *object schema* and *location name*. If the starting point does not refer to an alias, the location name of the starting point is returned. The resulting name can be of a table, view, or undefined object. The function returns a blank if there is no location name.

### *object name*

A character expression that specifies the unqualified name to be resolved. The unqualified name is usually of an existing alias. *object name* must have a data type of VARCHAR and an actual length that is no greater than 18 bytes.

### *object schema*

A character expression that represents the schema that is used to qualify the value specified in *object name* before resolution. *object schema* must have a data type of VARCHAR and an actual length that is no greater than 8 bytes.

If *object schema* is not specified or is null, the default schema is used for the qualifier.

### *object location*

A character expression that represents the location that is used to qualify the value specified in *object name* before resolution. *object location* must have a data type of VARCHAR and an actual length that is no greater than 16 bytes.

If *object location* is not specified or is null, the location name is equivalent to "any".

The result of the function has a data type of VARCHAR and an actual length that is no greater than 16 bytes. If *object name* can be null, the result can be null; if *object name* is null, the result is the null value.

Table 91 shows the external and specific names for TABLE\_LOCATION. The specific names differ depending on the number of input arguments to the function.

Table 91. External and specific names for TABLE\_LOCATION

| Input arguments                                                                                | External name | Specific name    |
|------------------------------------------------------------------------------------------------|---------------|------------------|
| <i>object name</i> (VARCHAR)                                                                   | DSN8DUTI      | DSN8.DSN8DUTILV  |
| <i>object name</i> (VARCHAR)<br><i>schema name</i> (VARCHAR)                                   | DSN8DUTI      | DSN8.DSN8DUTILVV |
| <i>object name</i> (VARCHAR)<br><i>schema name</i> (VARCHAR)<br><i>location name</i> (VARCHAR) | DSN8DUTI      | DSN8.DSN8DUTILVV |

## TABLE\_LOCATION

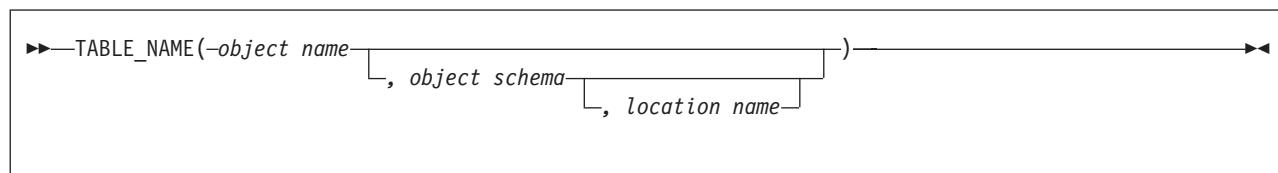
*Example:* Assume that:

- DSN8.ALIAS\_RS\_SYSTABLES is an alias of SYSIBM.SYSTABLES at location name REMOTE\_SITE.
- The current SQLID is DSN8.

Use TABLE\_LOCATION to find the location name where the base object for ALIAS\_OF\_SYSTABLES resides.

```
VALUES DSN8.TABLE_LOCATION('ALIAS_RS_SYSTABLES');
```

The result of the function is REMOTE\_SITE.

**TABLE\_NAME**

The schema is DSN8.

The TABLE\_NAME function searches for an object and returns the unqualified name of the object after any alias chains have been resolved. The starting point of the resolution is the object that is specified by *object name* and, if specified, *object schema* and *location name*. If the starting point does not refer to an alias, the unqualified name of the starting point is returned. The resulting name can be of a table, view, or undefined object.

*object name*

A character expression that specifies the unqualified name to be resolved. The unqualified name is usually of an existing alias. *object name* must have a data type of VARCHAR and an actual length that is no greater than 18 bytes.

*object schema*

A character expression that represents the schema that is used to qualify the value specified in *object name* before resolution. *object schema* must have a data type of VARCHAR and an actual length that is no greater than 8 bytes.

If *object schema* is not specified or is null, the default schema is used for the qualifier.

*object location*

A character expression that represents the location that is used to qualify the value specified in *object name* before resolution. *object location* must have a data type of VARCHAR and an actual length that is no greater than 16 bytes.

If *object location* is not specified or is null, the location name is equivalent to "any".

The result of the function has a data type of VARCHAR and an actual length that is no greater than 18 bytes. If *object name* can be null, the result can be null; if *object name* is null, the result is the null value.

Table 92 shows the external and specific names for TABLE\_NAME. The specific names differ depending on the number of input arguments to the function.

Table 92. External and specific names for TABLE\_NAME

| Input arguments                                                                                | External name | Specific name     |
|------------------------------------------------------------------------------------------------|---------------|-------------------|
| <i>object name</i> (VARCHAR)                                                                   | DSN8DUTI      | DSN8.DSN8DUTINV   |
| <i>object name</i> (VARCHAR)<br><i>schema name</i> (VARCHAR)                                   | DSN8DUTI      | DSN8.DSN8DUTINVV  |
| <i>object name</i> (VARCHAR)<br><i>schema name</i> (VARCHAR)<br><i>location name</i> (VARCHAR) | DSN8DUTI      | DSN8.DSN8DUTINVVV |

## **TABLE\_NAME**

*Example:* Assume that:

- DSN8.VIEW\_OF\_SYSTABLES is a view of SYSIBM.SYSTABLES.
- DSN8.ALIAS\_OF\_VIEW is an alias of DSN8.VIEW\_OF\_SYSTABLES.
- The current SQLID is DSN8.

Use TABLE\_NAME to find the name of the base object for ALIAS\_OF\_VIEW.

```
VALUES DSN8.TABLE_NAME('ALIAS_OF_SYSVIEW');
```

The result of the function is VIEW\_OF\_SYSTABLES.

**TABLE\_SCHEMA**

```
►►—TABLE_SCHEMA(—object name—
 , object schema—
 , location name—)————►►
```

The schema is DSN8.

The TABLE\_SCHEMA function searches for an object and returns the schema name of the object after any synonyms or alias chains have been resolved. The starting point of the resolution is the object that is specified by *objectname* and *objectschema*. If the starting point does not refer to an alias or synonym, the schema name of the starting point is returned. The resulting schema name can be of a table, view, or undefined object.

*object name*

A character expression that specifies the unqualified name to be resolved. The unqualified name is usually of an existing alias. *object name* must have a data type of VARCHAR and an actual length that is no greater than 18 bytes.

*object schema*

A character expression that represents the schema that is used to qualify the value specified in *object name* before resolution. *object schema* must have a data type of VARCHAR and an actual length that is no greater than 8 bytes.

If *object schema* is not specified or is null, the default schema is used for the qualifier.

*object location*

A character expression that represents the location that is used to qualify the value specified in *object name* before resolution. *object location* must have a data type of VARCHAR (and an actual length that is no greater than 16 bytes).

If *object location* is not specified or is null, the location name is equivalent to “any”.

The result of the function has a data type of VARCHAR and an actual length that is no greater than 8 bytes. If *object name* can be null, the result can be null; if *object name* is null, the result is the null value.

Table 93 shows the external and specific names for TABLE\_SCHEMA. The specific names differ depending on the number of input arguments.

Table 93. External and specific names for function TABLE\_SCHEMA

| Input arguments                                                                                | External name | Specific name     |
|------------------------------------------------------------------------------------------------|---------------|-------------------|
| <i>object name</i> (VARCHAR)                                                                   | DSN8DUTI      | DSN8.DSN8DUTISV   |
| <i>object name</i> (VARCHAR)<br><i>schema name</i> (VARCHAR)                                   | DSN8DUTI      | DSN8.DSN8DUTISVV  |
| <i>object name</i> (VARCHAR)<br><i>schema name</i> (VARCHAR)<br><i>location name</i> (VARCHAR) | DSN8DUTI      | DSN8.DSN8DUTISVVV |

## TABLE\_SCHEMA

*Example:* Assume that:

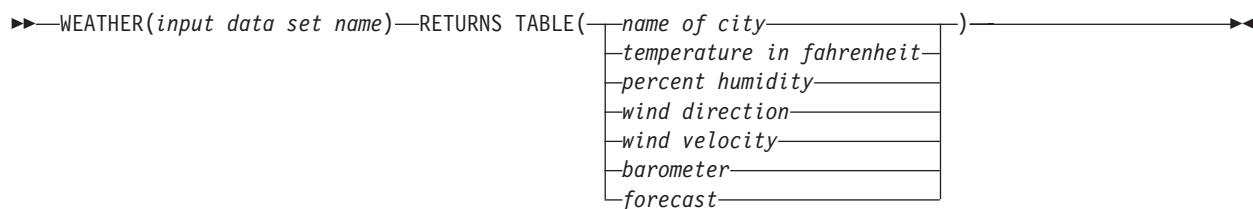
- DSN8.ALIAS\_OF\_SYSTABLES is an alias of SYSIBM.SYSTABLES.
- The current SQLID is DSN8.

Find the name of the schema of the base table for ALIAS\_OF\_SYSTABLES.

```
VALUES DSN8.TABLE_SCHEMA('ALIAS_OF_SYSTABLES');
```

The result of the function is SYSIBM.

## WEATHER



The schema is DSN8.

Unlike the other sample user-defined functions, which are scalar functions, WEATHER is a table function. WEATHER shows how to use a table function to make non-relational data available to a client for manipulation by SQL. The WEATHER function is provided primarily to help you design and implement table functions.

The WEATHER function returns information from a TSO data set as a DB2 table. The TSO data set contains sample weather statistics for various cities in the United States. The statistics are returned to the client with a row for each city and a column for each statistic.

*input data set name*

The name of the TSO data set that contains sample weather statistics. The name is a character string with a data type of VARCHAR and an actual length that is not greater than 44 bytes.

The result of the function is a DB2 table with the following columns. Each column can be null.

|                                         |             |
|-----------------------------------------|-------------|
| <b><i>name of city</i></b>              | VARCHAR(30) |
| <b><i>temperature in Fahrenheit</i></b> | INTEGER     |
| <b><i>percent humidity</i></b>          | INTEGER     |
| <b><i>wind direction</i></b>            | VARCHAR(5)  |
| <b><i>wind velocity</i></b>             | INTEGER     |
| <b><i>barometer</i></b>                 | FLOAT       |
| <b><i>forecast</i></b>                  | VARCHAR(25) |

The external program name for the function is DSN8DUWF, and the specific name is DSN8.DSN8DUWF.

*Example:* Find the name of and the forecast for the cities that have a temperature less than 25 degrees.

```

 SELECT CITY, FORECAST
 FROM TABLE(DSN8.WEATHER('prefix.SDSNIVPD(DSN8LWC)') AS W
 WHERE TEMP_IN_F < 25
 ORDER BY CITY;

```

This example returns:

|              |                       |
|--------------|-----------------------|
| Bessemer, MI | Slight chance of snow |
| Cheyenne, WY | Continued cooling     |
| Helena, MT   | Heavy snow            |
| Pierre, SD   | Continued cold        |

## WEATHER

---

## Appendix H. Notices

This information was developed for products and services offered in the U.S.A. IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing  
IBM Corporation  
North Castle Drive  
Armonk, NY 10504-1785  
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation  
Licensing  
2-31 Roppongi 3-chome, Minato-ku  
Tokyo 106-0032, Japan

**The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:**

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs

and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation  
J46A/G4  
555 Bailey Avenue  
San Jose, CA 95141-1003  
U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this information and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, or any equivalent agreement between us.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

#### COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

---

## Programming interface information

This book is intended to help you to code SQL statements. This book primarily documents General-use Programming Interface and Associated Guidance Information provided by IBM DATABASE 2 Universal Database Server for OS/390 and z/OS (DB2 for OS/390 and z/OS).

General-use programming interfaces allow the customer to write programs that obtain the services of DB2 for OS/390 and z/OS.

However, this book also documents Product-sensitive Programming Interface and Associated Guidance Information.

Product-sensitive programming interfaces allow the customer installation to perform tasks such as diagnosing, modifying, monitoring, repairing, tailoring, or tuning of this IBM software product. Use of such interfaces creates dependencies on the detailed design or implementation of the IBM software product. Product-sensitive programming interfaces should be used only for these specialized purposes. Because of their dependencies on detailed design and implementation, it is to be expected that programs written to such interfaces may need to be changed to run with new product releases or versions, or as a result of service.

Product-sensitive Programming Interface and Associated Guidance Information is identified where it occurs by an introductory statement to a chapter or section or by the following marking:

**Product-sensitive Programming Interface**

Product-sensitive Programming Interface and Associated Guidance Information ...

**End of Product-sensitive Programming Interface**

---

## Trademarks

The following terms are trademarks of the International Business Machines Corporation in the United States, or other countries, or both:

|                                                 |                           |
|-------------------------------------------------|---------------------------|
| AD/Cycle                                        | Enterprise Storage Server |
| AIX                                             | Enterprise System/3090    |
| APL2                                            | Enterprise System/9000    |
| AS/400                                          | IBM                       |
| BookManager                                     | IMS                       |
| CICS                                            | IMS/ESA                   |
| CICS/ESA                                        | Language Environment      |
| CICS/MVS                                        | MVS/DFP                   |
| COBOL/370                                       | MVS/ESA                   |
| C/370                                           | MVS/XA                    |
| DATABASE 2                                      | Net.Data                  |
| DataHub                                         | Operating System/390      |
| DataPropagator                                  | OS/2                      |
| DB2                                             | OS/390                    |
| DB2 Connect                                     | OS/400                    |
| DB2 Extenders                                   | Parallel Sysplex          |
| DB2 Universal Database                          | QMF                       |
| DFSMSdfp                                        | RACF                      |
| DFSMSdss                                        | RAMAC                     |
| DFSMShsm                                        | SAA                       |
| DFSMS/MVS                                       | SecureWay                 |
| DFSORT                                          | SQL/DS                    |
| Distributed Relational<br>Database Architecture | System/370                |
| DRDA                                            | System/390                |
|                                                 | VTAM                      |
|                                                 | WebSphere                 |
|                                                 | z/OS                      |

Lotus and Notes are trademarks of Lotus Development Corporation in the United States, or other countries, or both

Tivoli and NetView are trademarks of Tivoli Systems, Inc. in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Other company, product, and service names may be trademarks or service marks of others.

# Glossary

The following terms and abbreviations are defined as they are used in the DB2 library.

## A

**abend.** Abnormal end of task.

**abend reason code.** A 4-byte hexadecimal code that uniquely identifies a problem with DB2. A complete list of DB2 abend reason codes and their explanations is contained in *DB2 Messages and Codes*.

**abnormal end of task (abend).** Termination of a task, job, or subsystem because of an error condition that recovery facilities cannot resolve during execution.

**access method services.** The facility that is used to define and reproduce VSAM key-sequenced data sets.

**access path.** The path that is used to locate data that is specified in SQL statements. An access path can be indexed or sequential.

**active log.** The portion of the DB2 log to which log records are written as they are generated. The active log always contains the most recent log records, whereas the archive log holds those records that are older and no longer fit on the active log.

**active member state.** A state of a member of a data sharing group. An active member is identified with a group by XCF, which associates the member with a particular task, address space, and MVS system. A member that is not active has either a failed member state or a quiesced member state.

**after trigger.** A trigger that is defined with the trigger activation time AFTER.

**agent.** As used in DB2, the structure that associates all processes that are involved in a DB2 unit of work. An *allied agent* is generally synonymous with an *allied thread*. *System agents* are units of work that process independently of the allied agent, such as prefetch processing, deferred writes, and service tasks.

**alias.** An alternative name that can be used in SQL statements to refer to a table or view in the same or a remote DB2 subsystem.

**allied thread.** A thread that originates at the local DB2 subsystem and that can access data at a remote DB2 subsystem.

**allocated cursor.** A cursor that is defined for stored procedure result sets by using the SQL ALLOCATE CURSOR statement.

**already verified.** An LU 6.2 security option that allows DB2 to provide the user's verified authorization ID when allocating a conversation. The user is not validated by the partner DB2 subsystem.

**ambiguous cursor.** A database cursor that is not defined with the FOR FETCH ONLY clause or the FOR UPDATE OF clause, is not defined on a read-only result table, is not the target of a WHERE CURRENT clause on an SQL UPDATE or DELETE statement, and is in a plan or package that contains either PREPARE or EXECUTE IMMEDIATE SQL statements.

**American National Standards Institute (ANSI).** An organization consisting of producers, consumers, and general interest groups, that establishes the procedures by which accredited organizations create and maintain voluntary industry standards in the United States.

**ANSI.** American National Standards Institute.

**API.** Application programming interface.

**APPL.** A VTAM® network definition statement that is used to define DB2 to VTAM as an application program that uses SNA LU 6.2 protocols.

**application.** A program or set of programs that performs a task; for example, a payroll application.

**application-directed connection.** A connection that an application manages using the SQL CONNECT statement.

**application plan.** The control structure that is produced during the bind process. DB2 uses the application plan to process SQL statements that it encounters during statement execution.

**application process.** The unit to which resources and locks are allocated. An application process involves the execution of one or more programs.

**application programming interface (API).** A functional interface that is supplied by the operating system or by a separately orderable licensed program that allows an application program that is written in a high-level language to use specific data or functions of the operating system or licensed program.

**application requester.** The component on a remote system that generates DRDA requests for data on behalf of an application. An application requester accesses a DB2 database server using the DRDA application-directed protocol.

**application server.** The target of a request from a remote application. In the DB2 environment, the

## archive log • call level interface (CLI)

| application server function is provided by the distributed data facility and is used to access DB2 data from remote applications.

**archive log.** The portion of the DB2 log that contains log records that have been copied from the active log.

| **ASCII.** An encoding scheme that is used to represent strings in many environments, typically on PCs and workstations. Contrast with *EBCDIC* and *Unicode*.

**attachment facility.** An interface between DB2 and TSO, IMS, CICS, or batch address spaces. An attachment facility allows application programs to access DB2.

**attribute.** A characteristic of an entity. For example, in database design, the phone number of an employee is one of that employee's attributes.

**authorization ID.** A string that can be verified for connection to DB2 and to which a set of privileges is allowed. It can represent an individual, an organizational group, or a function, but DB2 does not determine this representation.

**auxiliary index.** An index on an auxiliary table in which each index entry refers to a LOB.

**auxiliary table.** A table that stores columns outside the table in which they are defined. Contrast with *base table*.

## B

**base table.** (1) A table that is created by the SQL CREATE TABLE statement and that holds persistent data. Contrast with *result table* and *temporary table*.  
(2) A table containing a LOB column definition. The actual LOB column data is not stored with the base table. The base table contains a row identifier for each row and an indicator column for each of its LOB columns. Contrast with *auxiliary table*.

**base table space.** A table space that contains base tables.

**basic predicate.** A predicate that compares two values.

**before trigger.** A trigger that is defined with the trigger activation time BEFORE.

**binary integer.** A basic data type that can be further classified as small integer or large integer.

**binary large object (BLOB).** A sequence of bytes where the size of the value ranges from 0 bytes to 2 GB-1. Such a string does not have an associated CCSID.

**binary string.** A sequence of bytes that is not associated with a CCSID. For example, the BLOB data type is a binary string.

**bind.** The process by which the output from the SQL precompiler is converted to a usable control structure, often called an access plan, application plan, or package. During this process, access paths to the data are selected and some authorization checking is performed. The types of bind are:

**automatic bind.** (More correctly, *automatic rebind*) A process by which SQL statements are bound automatically (without a user issuing a BIND command) when an application process begins execution and the bound application plan or package it requires is not valid.

**dynamic bind.** A process by which SQL statements are bound as they are entered.

**incremental bind.** A process by which SQL statements are bound during the execution of an application process, because they could not be bound during the bind process, and VALIDATE(RUN) was specified.

**static bind.** A process by which SQL statements are bound after they have been precompiled. All static SQL statements are prepared for execution at the same time.

**bit data.** Data that is character type CHAR or VARCHAR and is not associated with a coded character set.

**BLOB.** Binary large object.

**BMP.** Batch Message Processing (IMS).

**bootstrap data set (BSDS).** A VSAM data set that contains name and status information for DB2, as well as RBA range specifications, for all active and archive log data sets. It also contains passwords for the DB2 directory and catalog, and lists of conditional restart and checkpoint records.

**BSDS.** Bootstrap data set.

**buffer pool.** Main storage that is reserved to satisfy the buffering requirements for one or more table spaces or indexes.

**built-in function.** A function that DB2 supplies. Contrast with *user-defined function*.

## C

**cache structure.** A coupling facility structure that stores data that can be available to all members of a Sysplex. A DB2 data sharing group uses cache structures as group buffer pools.

**call level interface (CLI).** A callable application programming interface (API) for database access, which is an alternative to using embedded SQL. In contrast to

embedded SQL, DB2 ODBC (which is based on the CLI architecture) does not require the user to precompile or bind applications, but instead provides a standard set of functions to process SQL statements and related services at run time.

**cascade delete.** The way in which DB2 enforces referential constraints when it deletes all descendant rows of a deleted parent row.

**CASE expression.** Allows an expression to be selected based on the evaluation of one or more conditions.

**cast function.** A function that is used to convert instances of a (source) data type into instances of a different (target) data type. In general, a cast function has the name of the target data type. It has one single argument whose type is the source data type; its return type is the target data type.

**castout.** The DB2 process of writing changed pages from a group buffer pool to DASD.

**catalog.** In DB2, a collection of tables that contains descriptions of objects such as tables, views, and indexes.

**catalog table.** Any table in the DB2 catalog.

**CCSID.** Coded character set identifier.

**CDB.** Communications database.

**CDRA.** Character data representation architecture.

**Character Data Representation Architecture (CDRA).** An architecture that is used to achieve consistent representation, processing, and interchange of string data.

**character large object (CLOB).** A sequence of bytes representing single-byte characters or a mixture of single- and double-byte characters where the size of the value can be up to 2 GB–1. In general, character large object values are used whenever a character string might exceed the limits of the VARCHAR type.

**character set.** A defined set of characters.

**character string.** A sequence of bytes that represent bit data, single-byte characters, or a mixture of single-byte and multibyte characters.

**check constraint.** See *table check constraint*.

**check integrity.** The condition that exists when each row in a table conforms to the table check constraints that are defined on that table. Maintaining check integrity requires DB2 to enforce table check constraints on operations that add or change data.

**check pending.** A state of a table space or partition that prevents its use by some utilities and some SQL

statements because of rows that violate referential constraints, table check constraints, or both.

**checkpoint.** A point at which DB2 records internal status information on the DB2 log; the recovery process uses this information if DB2 abnormally terminates.

**CICS.** Represents (in this publication) one of the following products:

**CICS Transaction Server for OS/390:** Customer Information Control System Transaction Server for OS/390

**CICS/ESA:** Customer Information Control System/Enterprise Systems Architecture

**CICS/MVS:** Customer Information Control System/Multiple Virtual Storage

**CICS attachment facility.** A DB2 subcomponent that uses the MVS subsystem interface (SSI) and cross storage linkage to process requests from CICS to DB2 and to coordinate resource commitment.

**clause.** In SQL, a distinct part of a statement, such as a SELECT clause or a WHERE clause.

**CLI.** Call level interface.

**client.** See *requester*.

**CLOB.** Character large object.

**clustering index.** An index that determines how rows are physically ordered in a table space.

**coded character set.** A set of unambiguous rules that establish a character set and the one-to-one relationships between the characters of the set and their coded representations.

**coded character set identifier (CCSID).** A 16-bit number that uniquely identifies a coded representation of graphic characters. It designates an encoding scheme identifier and one or more pairs consisting of a character set identifier and an associated code page identifier.

**code page.** A set of assignments of characters to code points. In EBCDIC, for example, the character 'A' is assigned code point X'C1', and character 'B' is assigned code point X'C2'. Within a code page, each code point has only one specific meaning.

**code point.** In CDRA, a unique bit pattern that represents a character in a code page.

**collection.** A group of packages that have the same qualifier.

**column.** The vertical component of a table. A column has a name and a particular data type (for example, character, decimal, or integer).

## column function • DASD

**column function.** An operation that derives its result by using values from one or more rows. Contrast with *scalar function*.

**"come from" checking.** An LU 6.2 security option that defines a list of authorization IDs that are allowed to connect to DB2 from a partner LU.

**command.** A DB2 operator command or a DSN subcommand. A command is distinct from an SQL statement.

**commit.** The operation that ends a unit of work by releasing locks so that the database changes that are made by that unit of work can be perceived by other processes.

**commit point.** A point in time when data is considered consistent.

**committed phase.** The second phase of the multisite update process that requests all participants to commit the effects of the logical unit of work.

**communications database (CDB).** A set of tables in the DB2 catalog that are used to establish conversations with remote database management systems.

**comparison operator.** A token (such as `=`, `>`, `<`) that is used to specify a relationship between two values.

**composite key.** An ordered set of key columns of the same table.

**concurrency.** The shared use of resources by more than one application process at the same time.

**connection.** In SNA, the existence of a communication path between two partner LUs that allows information to be exchanged (for example, two DB2 subsystems that are connected and communicating by way of a conversation).

**consistency token.** A timestamp that is used to generate the version identifier for an application. See also *version*.

**constant.** A language element that specifies an unchanging value. Constants are classified as string constants or numeric constants. Contrast with *variable*.

**constraint.** A rule that limits the values that can be inserted, deleted, or updated in a table. See *referential constraint*, *table check constraint*, and *uniqueness constraint*.

**conversation.** Communication, which is based on LU 6.2 or Advanced Program-to-Program Communication (APPC), between an application and a remote transaction program over an SNA logical unit-to-logical unit (LU-LU) session that allows communication while processing a transaction.

**correlated subquery.** A subquery (part of a WHERE or HAVING clause) that is applied to a row or group of rows of a table or view that is named in an outer subselect statement.

**correlation ID.** An identifier that is associated with a specific thread. In TSO, it is either an authorization ID or the job name.

**correlation name.** An identifier that designates a table, a view, or individual rows of a table or view within a single SQL statement. It can be defined in any FROM clause or in the first clause of an UPDATE or DELETE statement.

**cost category.** A category into which DB2 places cost estimates for SQL statements at the time the statement is bound. A cost estimate can be placed in either of the following cost categories:

- A: Indicates that DB2 had enough information to make a cost estimate without using default values.
- B: Indicates that some condition exists for which DB2 was forced to use default values for its estimate.

The cost category is externalized in the COST\_CATEGORY column of the DSN\_STATEMENT\_TABLE when a statement is explained.

**created temporary table.** A table that holds temporary data and is defined with the SQL statement CREATE GLOBAL TEMPORARY TABLE. Information about created temporary tables is stored in the DB2 catalog, so this kind of table is persistent and can be shared across application processes. Contrast with *declared temporary table*. See also *temporary table*.

**CS.** Cursor stability.

**current data.** Data within a host structure that is current with (identical to) the data within the base table.

**cursor.** A named control structure that an application program uses to point to a row of interest within some set of rows, and to retrieve rows from the set, possibly making updates or deletions.

**cursor stability (CS).** The isolation level that provides maximum concurrency without the ability to read uncommitted data. With cursor stability, a unit of work holds locks only on its uncommitted changes and on the current row of each of its cursors.

**cycle.** A set of tables that can be ordered so that each table is a descendent of the one before it, and the first table is a descendent of the last table. A self-referencing table is a cycle with a single member.

## D

**DASD.** Direct access storage device.

**database.** A collection of tables, or a collection of table spaces and index spaces.

**database access thread.** A thread that accesses data at the local subsystem on behalf of a remote subsystem.

**database administrator (DBA).** An individual who is responsible for designing, developing, operating, safeguarding, maintaining, and using a database.

**database descriptor (DBD).** An internal representation of a DB2 database definition, which reflects the data definition that is in the DB2 catalog. The objects that are defined in a database descriptor are table spaces, tables, indexes, index spaces, and relationships.

**database management system (DBMS).** A software system that controls the creation, organization, and modification of a database and the access to the data stored within it.

**database request module (DBRM).** A data set member that is created by the DB2 precompiler and that contains information about SQL statements. DBRMs are used in the bind process.

**database server.** The target of a request from a local application or an intermediate database server. In the DB2 environment, the database server function is provided by the distributed data facility to access DB2 data from local applications, or from a remote database server that acts as an intermediate database server.

**DATABASE 2 Interactive (DB2I).** The DB2 facility that provides for the execution of SQL statements, DB2 (operator) commands, programmer commands, and utility invocation.

**data currency.** The state in which data that is retrieved into a host variable in your program is a copy of data in the base table.

**data sharing.** The ability of two or more DB2 subsystems to directly access and change a single set of data.

**data sharing group.** A collection of one or more DB2 subsystems that directly access and change the same data while maintaining data integrity.

**data sharing member.** A DB2 subsystem that is assigned by XCF services to a data sharing group.

**data type.** An attribute of columns, literals, host variables, special registers, and the results of functions and expressions.

**date.** A three-part value that designates a day, month, and year.

**date duration.** A decimal integer that represents a number of years, months, and days.

**datetime value.** A value of the data type DATE, TIME, or TIMESTAMP.

**DBA.** Database administrator.

**DBCLOB.** Double-byte character large object.

**DBCS.** Double-byte character set.

**DBD.** Database descriptor.

**DBID.** Database identifier.

**DBMS.** Database management system.

**DBRM.** Database request module.

**DB2 catalog.** Tables that are maintained by DB2 and contain descriptions of DB2 objects, such as tables, views, and indexes.

**DB2 command.** An instruction to the DB2 subsystem allowing a user to start or stop DB2, to display information on current users, to start or stop databases, to display information on the status of databases, and so on.

**DB2 for VSE & VM.** The IBM DB2 relational database management system for the VSE and VM operating systems.

**DB2I.** DATABASE 2 Interactive.

**DCLGEN.** Declarations generator.

**DDF.** Distributed data facility.

**declarations generator (DCLGEN).** A subcomponent of DB2 that generates SQL table declarations and COBOL, C, or PL/I data structure declarations that conform to the table. The declarations are generated from DB2 system catalog information. DCLGEN is also a DSN subcommand.

**declared temporary table.** A table that holds temporary data and is defined with the SQL statement DECLARE GLOBAL TEMPORARY TABLE. Information about declared temporary tables is not stored in the DB2 catalog, so this kind of table is not persistent and can only be used by the application process that issued the DECLARE statement. Contrast with *created temporary table*. See also *temporary table*.

**default value.** A predetermined value, attribute, or option that is assumed when no other is explicitly specified.

**deferred embedded SQL.** SQL statements that are neither fully static nor fully dynamic. Like static statements, they are embedded within an application, but like dynamic statements, they are prepared during the execution of the application.

## delete-connected • DRDA access

**delete-connected.** A table that is a dependent of table P or a dependent of a table to which delete operations from table P cascade.

**delete rule.** The rule that tells DB2 what to do to a dependent row when a parent row is deleted. For each relationship, the rule might be CASCADE, RESTRICT, SET NULL, or NO ACTION.

**delete trigger.** A trigger that is defined with the triggering SQL operation DELETE.

**delimited identifier.** A sequence of characters that are enclosed within double quotation marks (""). The sequence must consist of a letter followed by zero or more characters, each of which is a letter, digit, or the underscore character (\_).

**delimiter token.** A string constant, a delimited identifier, an operator symbol, or any of the special characters that are shown in syntax diagrams.

**dependent.** An object (row, table, or table space) that has at least one parent. The object is also said to be a dependent (row, table, or table space) of its parent. See *parent row*, *parent table*, *parent table space*.

**dependent row.** A row that contains a foreign key that matches the value of a primary key in the parent row.

**dependent table.** A table that is a dependent in at least one referential constraint.

**descendent.** An object that is a dependent of an object or is the dependent of a descendent of an object.

**descendent row.** A row that is dependent on another row, or a row that is a descendent of a dependent row.

**descendent table.** A table that is a dependent of another table, or a table that is a descendent of a dependent table.

**deterministic function.** A user-defined function whose result is dependent on the values of the input arguments. That is, successive invocations with the same input values produce the same answer. Sometimes referred to as a *not-variant* function. Contrast this with an *not-deterministic function* (sometimes called a *variant function*), which might not always produce the same result for the same inputs.

**dimension.** A data category such as time, products, or markets. The elements of a dimension are referred to as members. Dimensions offer a very concise, intuitive way of organizing and selecting data for retrieval, exploration, and analysis. See also *dimension table*.

**dimension table.** The representation of a dimension in a star schema. Each row in a dimension table represents all of the attributes for a particular member of the dimension. See also *dimension*, *star schema*, and *star join*.

**direct access storage device (DASD).** A device in which access time is independent of the location of the data.

**directory.** The DB2 system database that contains internal objects such as database descriptors and skeleton cursor tables.

**distinct type.** A user-defined data type that is internally represented as an existing type (its source type), but is considered to be a separate and incompatible type for semantic purposes.

**distributed data facility (DDF).** A set of DB2 components through which DB2 communicates with another RDBMS.

**Distributed Relational Database Architecture (DRDA).** A connection protocol for distributed relational database processing that is used by IBM's relational database products. DRDA includes protocols for communication between an application and a remote relational database management system, and for communication between relational database management systems.

**DNS.** Domain name server.

**domain name.** The name by which TCP/IP applications refer to a TCP/IP host within a TCP/IP network.

**domain name server (DNS).** A special TCP/IP network server that manages a distributed directory that is used to map TCP/IP host names to IP addresses.

**double-byte character large object (DBCLOB).** A sequence of bytes representing double-byte characters where the size of the values can be up to 2 GB. In general, double-byte character large object values are used whenever a double-byte character string might exceed the limits of the VARGRAPHIC type.

**double-byte character set (DBCS).** A set of characters, which are used by national languages such as Japanese and Chinese, that have more symbols than can be represented by a single byte. Each character is 2 bytes in length. Contrast with *single-byte character set* and *multibyte character set*.

**double-precision floating point number.** A 64-bit approximate representation of a real number.

**DRDA.** Distributed Relational Database Architecture.

**DRDA access.** An open method of accessing distributed data that you can use to connect to another database server to execute packages that were previously bound at the server location. You use the SQL CONNECT statement or an SQL statement with a three-part name to identify the server. Contrast with *private protocol access*.

**DSN.** (1) The default DB2 subsystem name. (2) The name of the TSO command processor of DB2. (3) The first three characters of DB2 module and macro names.

**duration.** A number that represents an interval of time. See *date duration*, *labeled duration*, and *time duration*.

**dynamic SQL.** SQL statements that are prepared and executed within an application program while the program is executing. In dynamic SQL, the SQL source is contained in host language variables rather than being coded into the application program. The SQL statement can change several times during the application program's execution.

## E

**EA-enabled table space.** A table space or index space that is enabled for extended addressability and that contains individual partitions (or pieces, for LOB table spaces) that are greater than 4 GB.

| **EBCDIC.** Extended binary coded decimal interchange code. An encoding scheme that is used to represent character data in the OS/390, MVS, VM, VSE, and OS/400® environments. Contrast with *ASCII* and *Unicode*.

**EDM pool.** A pool of main storage that is used for database descriptors, application plans, authorization cache, application packages, and dynamic statement caching.

**embedded SQL.** SQL statements that are coded within an application program. See *static SQL*.

| **encoding scheme.** A set of rules to represent character data (ASCII, EBCDIC, or Unicode).

**equijoin.** A join operation in which the join-condition has the form *expression = expression*.

**escape character.** The symbol that is used to enclose an SQL delimited identifier. The escape character is the double quotation mark ("), except in COBOL applications, where the user assigns the symbol, which is either a double quotation mark or an apostrophe (').

**EUR.** IBM European Standards.

**exclusive lock.** A lock that prevents concurrently executing application processes from reading or changing data. Contrast with *share lock*.

**executable statement.** An SQL statement that can be embedded in an application program, dynamically prepared and executed, or issued interactively.

**exit routine.** A user-written (or IBM-provided default) program that receives control from DB2 to perform specific functions. Exit routines run as extensions of DB2.

**exposed name.** A correlation name or a table or view name for which a correlation name is not specified. Names specified in a FROM clause are exposed or non-exposed.

**expression.** An operand or a collection of operators and operands that yields a single value.

**external function.** A function for which the body is written in a programming language that takes scalar argument values and produces a scalar result for each invocation. Contrast with *sourced function*, *built-in function*, and *SQL function*.

**external procedure.** An application program written by a user that can be invoked with the SQL CALL statement written in a programming language. Contrast with *SQL procedure*.

**external routine.** A user-defined function or stored procedure that is based on code that is written in an external programming language.

## F

**failed member state.** A state of a member of a data sharing group. When a member fails, the XCF permanently records the failed member state. This state usually means that the member's task, address space, or MVS system terminated before the state changed from active to quiesced.

**fallback.** The process of returning to a previous release of DB2 after attempting or completing migration to a current release.

**field procedure.** A user-written exit routine that is designed to receive a single value and transform (encode or decode) it in any way the user can specify.

**filter factor.** A number between zero and one that estimates the proportion of rows in a table for which a predicate is true.

**fixed-length string.** A character or graphic string whose length is specified and cannot be changed. Contrast with *varying-length string*.

**foreign key.** A column or set of columns in a dependent table of a constraint relationship. The key must have the same number of columns, with the same descriptions, as the primary key of the parent table. Each foreign key value must either match a parent key value in the related parent table or be null.

**free space.** The total amount of unused space in a page; that is, the space that is not used to store records or control information is free space.

**full outer join.** The result of a join operation that includes the matched rows of both tables that are being joined and preserves the unmatched rows of both tables. See also *join*.

## fullselect • index partition

**fullselect.** A subselect, a values-clause, or a number of both that are combined by set operators. *Fullselect* specifies a result table. If UNION is not used, the result of the fullselect is the result of the specified subselect.

**function.** A mapping, embodied as a program (the function body), invocable by means of zero or more input values (arguments), to a single value (the result). See also *column function* and *scalar function*.

Functions can be user-defined, built-in, or generated by DB2. (See *built-in function*, *cast function*, *external function*, *sourced function*, *SQL function*, and *user-defined function*.)

**function definer.** The authorization ID of the owner of the schema of the function that is specified in the CREATE FUNCTION statement.

**function implementer.** The authorization ID of the owner of the function program and function package.

**function package.** A package that results from binding the DBRM for a function program.

**function resolution.** The process, internal to the DBMS, by which a function invocation is bound to a particular function instance. This process uses the function name, the data types of the arguments, and a list of the applicable schema names (called the *SQL path*) to make the selection. This process is sometimes called *function selection*.

**function selection.** See *function resolution*.

**function signature.** The logical concatenation of a fully qualified function name with the data types of all of its parameters.

## G

**GB.** Gigabyte (1 073 741 824 bytes).

**GBP.** Group buffer pool.

**GBP-dependent.** The status of a page set or page set partition that is dependent on the group buffer pool. Either read/write interest is active among DB2 subsystems for this page set, or the page set has changed pages in the group buffer pool that have not yet been cast out to DASD.

**graphic string.** A sequence of DBCS characters.

**gross lock.** The *shared*, *update*, or *exclusive* mode locks on a table, partition, or table space.

**group buffer pool (GBP).** A coupling facility cache structure that is used by a data sharing group to cache data and to ensure that the data is consistent for all members.

**group name.** The MVS XCF identifier for a data sharing group.

**group restart.** A restart of at least one member of a data sharing group after the loss of either locks or the shared communications area.

## H

**host.** The set of programs and resources that are available on a given TCP/IP instance.

**host identifier.** A name that is declared in the host program.

**host language.** A programming language in which you can embed SQL statements.

**host program.** An application program that is written in a host language and that contains embedded SQL statements.

**host structure.** In an application program, a structure that is referenced by embedded SQL statements.

**host variable.** In an application program, an application variable that is referenced by embedded SQL statements.

## I

**ICF.** Integrated catalog facility.

**identity column.** A column that provides a way for DB2 to automatically generate a numeric value for each row. The generated values are unique if cycling is not used. Identity columns are defined with the AS IDENTITY clause. Uniqueness of values can be ensured by defining a single-column unique index using the identity column. A table can have no more than one identity column.

**image copy.** An exact reproduction of all or part of a table space. DB2 provides utility programs to make full image copies (to copy the entire table space) or incremental image copies (to copy only those pages that have been modified since the last image copy).

**IMS.** Information Management System.

**independent.** An object (row, table, or table space) that is neither a parent nor a dependent of another object.

**index.** A set of pointers that are logically ordered by the values of a key. Indexes can provide faster access to data and can enforce uniqueness on the rows in a table.

**index key.** The set of columns in a table that is used to determine the order of index entries.

**index partition.** A VSAM data set that is contained within a partitioning index space.

**index space.** A page set that is used to store the entries of one index.

**indicator column.** A 4-byte value that is stored in a base table in place of a LOB column.

**indicator variable.** A variable that is used to represent the null value in an application program. If the value for the selected column is null, a negative value is placed in the indicator variable.

**indoubt.** A status of a unit of recovery. If DB2 fails after it has finished its phase 1 commit processing and before it has started phase 2, only the commit coordinator knows if an individual unit of recovery is to be committed or rolled back. At emergency restart, if DB2 lacks the information it needs to make this decision, the status of the unit of recovery is *indoubt* until DB2 obtains this information from the coordinator. More than one unit of recovery can be indoubt at restart.

**indoubt resolution.** The process of resolving the status of an indoubt logical unit of work to either the committed or the rollback state.

**inner join.** The result of a join operation that includes only the matched rows of both tables being joined. See also *join*.

**inoperative package.** A package that cannot be used because one or more user-defined functions or procedures that the package depends on were dropped. Such a package must be explicitly rebound. Contrast with *invalid package*.

**insensitive cursor.** A cursor that is not sensitive to inserts, updates, or deletes that are made to the underlying rows of a result table after the result table has materialized.

**insert trigger.** A trigger that is defined with the triggering SQL operation INSERT.

**inter-DB2 R/W interest.** A property of data in a table space, index, or partition that has been opened by more than one member of a data sharing group and that has been opened for writing by at least one of those members.

| **intermediate database server.** The target of a request from a local application or a remote application requester that is forwarded to another database server.  
| In the DB2 environment, the remote request is forwarded transparently to another database server if the object that is referenced by a three-part name does not reference the local location.

**internal resource lock manager (IRLM).** An MVS subsystem that DB2 uses to control communication and database locking.

**invalid package.** A package that depends on an object (other than a user-defined function) that is dropped. Such a package is implicitly rebound on invocation. Contrast with *inoperative package*.

**invariant character set.** (1) A character set, such as the syntactic character set, whose code point assignments do not change from code page to code page. (2) A minimum set of characters that is available as part of all character sets.

**IP address.** A 4-byte value that uniquely identifies a TCP/IP host.

**IRLM.** Internal resource lock manager.

**ISO.** International Standards Organization.

**isolation level.** The degree to which a unit of work is isolated from the updating operations of other units of work. See also *cursor stability*, *read stability*, *repeatable read*, and *uncommitted read*.

## J

**Japanese Industrial Standards Committee (JISC).** An organization that issues standards for coding character sets.

**Java® Archive (JAR).** A file format that is used for aggregating many files into a single file.

**JIS.** Japanese Industrial Standard.

**join.** A relational operation that allows retrieval of data from two or more tables based on matching column values. See also *equijoin*, *full outer join*, *inner join*, *left outer join*, *outer join*, and *right outer join*.

## K

**KB.** Kilobyte (1024 bytes).

**key.** A column or an ordered collection of columns identified in the description of a table, index, or referential constraint.

**keyword.** In SQL, a name that identifies an option used in an SQL statement.

## L

**labeled duration.** A number that represents a duration of years, months, days, hours, minutes, seconds, or microseconds.

**large object (LOB).** A sequence of bytes representing bit data, single-byte characters, double-byte characters, or a mixture of single- and double-byte characters. A LOB can be up to 2 GB-1 byte in length. See also *BLOB*, *CLOB*, and *DBCLOB*.

## leaf page • migration

**leaf page.** A page that contains pairs of keys and RIDs and that points to actual data. Contrast with *nonleaf page*.

**left outer join.** The result of a join operation that includes the matched rows of both tables that are being joined, and that preserves the unmatched rows of the first table. See also *join*.

**L-lock.** Logical lock.

**LOB.** Large object.

**LOB locator.** A mechanism that allows an application program to manipulate a large object value in the database system. A LOB locator is a fullword integer value that represents a single LOB value. An application program retrieves a LOB locator into a host variable and can then apply SQL operations to the associated LOB value using the locator.

**LOB table space.** A table space that contains all the data for a particular LOB column in the related base table.

**local.** A way of referring to any object that the local DB2 subsystem maintains. A *local table*, for example, is a table that is maintained by the local DB2 subsystem. Contrast with *remote*.

**local subsystem.** The unique RDBMS to which the user or application program is directly connected (in the case of DB2, by one of the DB2 attachment facilities).

| **location.** The unique name of a database server. An application uses the location name to access a DB2 database server.

**lock.** A means of controlling concurrent events or access to data. DB2 locking is performed by the IRLM.

**lock duration.** The interval over which a DB2 lock is held.

**lock escalation.** The promotion of a lock from a row, page, or LOB lock to a table space lock because the number of page locks that are concurrently held on a given resource exceeds a preset limit.

**locking.** The process by which the integrity of data is ensured. Locking prevents concurrent users from accessing inconsistent data.

**lock mode.** A representation for the type of access that concurrently running programs can have to a resource that a DB2 lock is holding.

**lock object.** The resource that is controlled by a DB2 lock.

**lock promotion.** The process of changing the size or mode of a DB2 lock to a higher level.

**lock size.** The amount of data controlled by a DB2 lock on table data; the value can be a row, a page, a LOB, a partition, a table, or a table space.

**log.** A collection of records that describe the events that occur during DB2 execution and that indicate their sequence. The information thus recorded is used for recovery in the event of a failure during DB2 execution.

**logical index partition.** The set of all keys that reference the same data partition.

**logical lock (L-lock).** The lock type that transactions use to control intra- and inter-DB2 data concurrency between transactions. Contrast with *physical lock (P-lock)*.

**logical unit.** An access point through which an application program accesses the SNA network in order to communicate with another application program.

**logical unit of work (LUW).** The processing that a program performs between synchronization points.

**log initialization.** The first phase of restart processing during which DB2 attempts to locate the current end of the log.

**log record sequence number (LRSN).** A number that DB2 generates and associates with each log record. DB2 also uses the LRSN for page versioning. The LRSNs that a particular DB2 data sharing group generates form a strictly increasing sequence for each DB2 log and a strictly increasing sequence for each page across the DB2 group.

**log truncation.** A process by which an explicit starting RBA is established. This RBA is the point at which the next byte of log data is to be written.

**long string.** A string whose actual length, or a varying-length string whose maximum length, is greater than 255 bytes or 127 double-byte characters. Any LOB column, LOB host variable, or expression that evaluates to a LOB is considered a long string.

**LRSN.** Log record sequence number.

**LU name.** Logical unit name, which is the name by which VTAM refers to a node in a network. Contrast with *location name*.

**LUW.** Logical unit of work.

## M

**member name.** The MVS XCF identifier for a particular DB2 subsystem in a data sharing group.

**migration.** The process of converting a DB2 subsystem with a previous release of DB2 to an updated or current release. In this process, you can

acquire the functions of the updated or current release without losing the data you created on the previous release.

**mixed data string.** A character string that can contain both single-byte and double-byte characters.

**mode name.** A VTAM name for the collection of physical and logical characteristics and attributes of a session.

**multisite update.** Distributed relational database processing in which data is updated in more than one location within a single unit of work.

**MVS.** Multiple Virtual Storage.

**MVS/ESA™.** Multiple Virtual Storage/Enterprise Systems Architecture.

**MVS/XA™.** Multiple Virtual Storage/Extended Architecture.

## N

**nested table expression.** A fullselect in a FROM clause (surrounded by parentheses).

**nonleaf page.** A page that contains keys and page numbers of other pages in the index (either leaf or nonleaf pages). Nonleaf pages never point to actual data.

**nonpartitioning index.** Any index that is not a partitioning index.

**not-deterministic function.** A user-defined function whose result is not solely dependent on the values of the input arguments. That is, successive invocations with the same argument values can produce a different answer. This type of function is sometimes called a *variant* function. Contrast this with a *deterministic function* (sometimes called a *not-variant function*), which always produces the same result for the same inputs.

**not-variant function.** See *deterministic function*.

**NUL.** In C, a single character that denotes the end of the string.

**null.** A special value that indicates the absence of information.

**NULLIF.** A scalar function that evaluates two passed expressions, returning either NULL if the arguments are equal or the value of the first argument if they are not.

**NUL-terminated host variable.** A varying-length host variable in which the end of the data is indicated by the presence of a NUL terminator.

**NUL terminator.** In C, the value that indicates the end of a string. For character strings, the NUL terminator is X'00'.

## O

**ODBC.** Open Database Connectivity.

**OBID.** Data object identifier.

**Open Database Connectivity (ODBC).** A Microsoft® database application programming interface (API) for C that allows access to database management systems by using callable SQL. ODBC does not require the use of an SQL preprocessor. In addition, ODBC provides an architecture that lets users add modules called *database drivers*, which link the application to their choice of database management systems at run time. This means that applications no longer need to be directly linked to the modules of all the database management systems that are supported.

**ordinary identifier.** An uppercase letter followed by zero or more characters, each of which is an uppercase letter, a digit, or the underscore character. An ordinary identifier must not be a reserved word.

**ordinary token.** A numeric constant, an ordinary identifier, a host identifier, or a keyword.

**OS/390.** Operating System/390®.

**outer join.** The result of a join operation that includes the matched rows of both tables that are being joined and preserves some or all of the unmatched rows of the tables that are being joined. See also *join*.

**overloaded function.** A function name for which multiple function instances exist.

## P

**package.** An object containing a set of SQL statements that have been statically bound and that is available for processing. A package is sometimes also called an *application package*.

**package list.** An ordered list of package names that may be used to extend an application plan.

**package name.** The name of an object that is created by a BIND PACKAGE or REBIND PACKAGE command. The object is a bound version of a database request module (DBRM). The name consists of a location name, a collection ID, a package ID, and a version ID.

**page.** A unit of storage within a table space (4 KB, 8 KB, 16 KB, or 32 KB) or index space (4 KB). In a table space, a page contains one or more rows of a table. In a LOB table space, a LOB value can span more than one page, but no more than one LOB value is stored on a page.

## page set • privilege

**page set.** Another way to refer to a table space or index space. Each page set consists of a collection of VSAM data sets.

**parallel I/O processing.** A form of I/O processing in which DB2 initiates multiple concurrent requests for a single user query and performs I/O processing concurrently (in *parallel*) on multiple data partitions.

**parameter marker.** A question mark (?) that appears in a statement string of a dynamic SQL statement. The question mark can appear where a host variable could appear if the statement string were a static SQL statement.

**parameter-name.** A long identifier that names a parameter that can be referenced in a procedure or user-defined function.

**parent key.** A primary key or unique key in the parent table of a referential constraint. The values of a parent key determine the valid values of the foreign key in the referential constraint.

**parent row.** A row whose primary key value is the foreign key value of a dependent row.

**parent table.** A table whose primary key is referenced by the foreign key of a dependent table.

**parent table space.** A table space that contains a parent table. A table space containing a dependent of that table is a dependent table space.

**partition.** A portion of a page set. Each partition corresponds to a single, independently extendable data set. Partitions can be extended to a maximum size of 1, 2, or 4 GB, depending on the number of partitions in the partitioned page set. All partitions of a given page set have the same maximum size.

**partitioned data set (PDS).** A data set in direct access storage that is divided into partitions, which are called members. Each partition can contain a program, part of a program, or data. The term partitioned data set is synonymous with program library.

**partitioned table space.** A table space that is subdivided into parts (based on index key range), each of which can be processed independently by utilities.

**path.** See *SQL path*.

**PDS.** Partitioned data set.

**piece.** A data set of a nonpartitioned page set.

**plan.** See *application plan*.

**plan allocation.** The process of allocating DB2 resources to a plan in preparation for execution.

**plan name.** The name of an application plan.

**point of consistency.** A time when all recoverable data that an application accesses is consistent with other data. The term point of consistency is synonymous with *sync point* or *commit point*.

**precompilation.** A processing of application programs containing SQL statements that takes place before compilation. SQL statements are replaced with statements that are recognized by the host language compiler. Output from this precompilation includes source code that can be submitted to the compiler and the database request module (DBRM) that is input to the bind process.

**predicate.** An element of a search condition that expresses or implies a comparison operation.

**prefix.** A code at the beginning of a message or record.

**prepare.** The first phase of a two-phase commit process in which all participants are requested to prepare for commit.

**prepared SQL statement.** A named object that is the executable form of an SQL statement that has been processed by the PREPARE statement.

**primary authorization ID.** The authorization ID used to identify the application process to DB2.

**primary index.** An index that enforces the uniqueness of a primary key.

**primary key.** In a relational database, a unique, nonnull key that is part of the definition of a table. A table cannot be defined as a parent unless it has a unique key or primary key.

**private connection.** A communications connection that is specific to DB2.

**private protocol access.** A method of accessing distributed data by which you can direct a query to another DB2 system. Contrast with *DRDA access*.

**private protocol connection.** A DB2 private connection of the application process. See also *private connection*.

**privilege.** The capability of performing a specific function, sometimes on a specific object. The term includes:

**explicit privileges**, which have names and are held as the result of SQL GRANT and REVOKE statements. For example, the SELECT privilege.

**implicit privileges**, which accompany the ownership of an object, such as the privilege to drop a synonym one owns, or the holding of an authority, such as the privilege of SYSADM authority to terminate any utility job.

**privilege set.** For the installation SYSADM ID, the set of all possible privileges. For any other authorization ID, the set of all privileges that are recorded for that ID in the DB2 catalog.

**process.** In DB2, the unit to which DB2 allocates resources and locks. Sometimes called an *application process*, a process involves the execution of one or more programs. The execution of an SQL statement is always associated with some process. The means of initiating and terminating a process are dependent on the environment.

**program.** A single compilable collection of executable statements in a programming language.

**protected conversation.** A VTAM conversation that supports two-phase commit flows.

## **Q**

**QMF™.** Query Management Facility.

**query.** A component of certain SQL statements that specifies a result table.

**query block.** The part of a query that is represented by one of the FROM clauses. Each FROM clause can have multiple query blocks, depending on DB2's internal processing of the query.

**quiesced member state.** A state of a member of a data sharing group. An active member becomes quiesced when a STOP DB2 command takes effect without a failure. If the member's task, address space, or MVS system fails before the command takes effect, the member state is failed.

## **R**

**RACF.** Resource Access Control Facility, which is a component of the SecureWay Security Server for OS/390.

**RDB.** Relational database.

**RDBMS.** Relational database management system.

**RDBNAM.** Relational database name.

**read stability (RS).** An isolation level that is similar to repeatable read but does not completely isolate an application process from all other concurrently executing application processes. Under level RS, an application that issues the same query more than once might read additional rows that were inserted and committed by a concurrently executing application process.

**rebind.** The creation of a new application plan for an application program that has been bound previously. If, for example, you have added an index for a table that

your application accesses, you must rebind the application in order to take advantage of that index.

**record.** The storage representation of a row or other data.

**recovery.** The process of rebuilding databases after a system failure.

**recovery log.** A collection of records that describes the events that occur during DB2 execution and indicates their sequence. The recorded information is used for recovery in the event of a failure during DB2 execution.

**referential constraint.** The requirement that nonnull values of a designated foreign key are valid only if they equal values of the primary key of a designated table.

**referential integrity.** The state of a database in which all values of all foreign keys are valid. Maintaining referential integrity requires the enforcement of referential constraints on all operations that change the data in a table upon which the referential constraints are defined.

**referential structure.** A set of tables and relationships that includes at least one table and, for every table in the set, all the relationships in which that table participates and all the tables to which it is related.

**relational database (RDB).** A database that can be perceived as a set of tables and manipulated in accordance with the relational model of data.

**relational database management system (RDBMS).** A collection of hardware and software that organizes and provides access to a relational database.

**relational database name (RDBNAM).** A unique identifier for an RDBMS within a network. In DB2, this must be the value in the LOCATION column of table SYSIBM.LOCATIONS in the CDB. DB2 publications refer to the name of another RDBMS as a LOCATION value or a location name.

**relationship.** A defined connection between the rows of a table or the rows of two tables. A relationship is the internal representation of a referential constraint.

**remote.** Any object that is maintained by a remote DB2 subsystem (that is, by a DB2 subsystem other than the local one). A *remote view*, for example, is a view that is maintained by a remote DB2 subsystem. Contrast with *local*.

**remote subsystem.** Any RDBMS, except the *local subsystem*, with which the user or application can communicate. The subsystem need not be remote in any physical sense, and might even operate on the same processor under the same MVS system.

## reoptimization • sequential prefetch

**reoptimization.** The DB2 process of reconsidering the access path of an SQL statement at run time; during reoptimization, DB2 uses the values of host variables, parameter markers, or special registers.

**repeatable read (RR).** The isolation level that provides maximum protection from other executing application programs. When an application program executes with repeatable read protection, rows referenced by the program cannot be changed by other programs until the program reaches a commit point.

**request commit.** The vote that is submitted to the prepare phase if the participant has modified data and is prepared to commit or roll back.

| **requester.** The source of a request to access data at a remote server. In the DB2 environment, the requester function is provided by the distributed data facility.

**resource.** The object of a lock or claim, which could be a table space, an index space, a data partition, an index partition, or a logical partition.

**resource limit facility (RLF).** A portion of DB2 code that prevents dynamic manipulative SQL statements from exceeding specified time limits. The resource limit facility is sometimes called the governor.

**result set.** The set of rows that a stored procedure returns to a client application.

**result set locator.** A 4-byte value that DB2 uses to uniquely identify a query result set that a stored procedure returns.

**result table.** The set of rows that are specified by a SELECT statement.

**right outer join.** The result of a join operation that includes the matched rows of both tables that are being joined and preserves the unmatched rows of the second join operand. See also *join*.

**RLF.** Resource limit facility.

**rollback.** The process of restoring data changed by SQL statements to the state at its last commit point. All locks are freed. Contrast with *commit*.

**routine.** A term that refers to either a user-defined function or a stored procedure.

**row.** The horizontal component of a table. A row consists of a sequence of values, one for each column of the table.

**ROWID.** Row identifier.

**row identifier (ROWID).** A value that uniquely identifies a row. This value is stored with the row and never changes.

**row trigger.** A trigger that is defined with the trigger granularity FOR EACH ROW.

**row-value-expression.** A comma-separated list of value expressions enclosed in parentheses.

**RS.** Read stability.

## S

| **savepoint.** A named entity that represents the state of data and schemas at a particular point in time within a unit of work. SQL statements exist to set a savepoint, release a savepoint, and restore data and schemas to the state that the savepoint represents. The restoration of data and schemas to a savepoint is usually referred to as *rolling back to a savepoint*.

**SBCS.** Single-byte character set.

**scalar function.** An SQL operation that produces a single value from another value and is expressed as a function name, followed by a list of arguments that are enclosed in parentheses. Contrast with *column function*.

**schema.** A logical grouping for user-defined functions, distinct types, triggers, and stored procedures. When an object of one of these types is created, it is assigned to one schema, which is determined by the name of the object. For example, the following statement creates a distinct type T in schema C:

CREATE DISTINCT TYPE C.T ...

**search condition.** A criterion for selecting rows from a table. A search condition consists of one or more predicates.

**secondary authorization ID.** An authorization ID that has been associated with a primary authorization ID by an authorization exit routine.

**segmented table space.** A table space that is divided into equal-sized groups of pages called segments. Segments are assigned to tables so that rows of different tables are never stored in the same segment.

**self-referencing constraint.** A referential constraint that defines a relationship in which a table is a dependent of itself.

**self-referencing table.** A table with a self-referencing constraint.

**sensitive cursor.** A cursor that is sensitive to changes made to the database after the result table has materialized.

**sequential prefetch.** A mechanism that triggers consecutive asynchronous I/O operations. Pages are fetched before they are required, and several pages are read with a single I/O operation.

**serial cursor.** A cursor that can be moved only in a forward direction.

| **server.** The target of a request from a remote requester. In the DB2 environment, the server function is provided by the distributed data facility, which is used to access DB2 data from remote applications.

**share lock.** A lock that prevents concurrently executing application processes from changing data, but not from reading data. Contrast with *exclusive lock*.

**shift-in character.** A special control character (X'0F') that is used in EBCDIC systems to denote that the subsequent bytes represent SBCS characters. See also *shift-out character*.

**shift-out character.** A special control character (X'0E') that is used in EBCDIC systems to denote that the subsequent bytes, up to the next shift-in control character, represent DBCS characters. See also *shift-in character*.

**short string.** A string whose actual length, or a varying-length string whose maximum length, is 255 bytes (or 127 double-byte characters) or less. Regardless of length, a LOB string is not a short string.

**sign-on.** A request that is made on behalf of an individual CICS or IMS application process by an attachment facility to enable DB2 to verify that it is authorized to use DB2 resources.

**simple table space.** A table space that is neither partitioned nor segmented.

| **single-byte character set (SBCS).** A set of characters in which each character is represented by a single byte. Contrast with *double-byte character set* or *multibyte character set*.

**single-precision floating point number.** A 32-bit approximate representation of a real number.

**SMF.** System management facility.

**SMS.** Storage Management Subsystem.

**socket.** A callable TCP/IP programming interface that is used by TCP/IP network applications to communicate with remote TCP/IP partners.

**sourced function.** A function that is implemented by another built-in or user-defined function that is already known to the database manager. This function can be a scalar function or a column (aggregating) function; it returns a single value from a set of values (for example, MAX or AVG). Contrast with *built-in function*, *external function*, and *SQL function*.

**source program.** A set of host language statements and SQL statements that is processed by an SQL precompiler.

**source type.** An existing type that is used to internally represent a distinct type.

**space.** A sequence of one or more blank characters.

**special register.** A storage area that DB2 defines for an application process to use for storing information that can be referenced in SQL statements. Examples of special registers are USER and CURRENT DATE.

**specific function name.** A particular user-defined function that is known to the database manager by its specific name. Many specific user-defined functions can have the same function name. When a user-defined function is defined to the database, every function is assigned a specific name that is unique within its schema. Either the user can provide this name, or a default name is used.

**SPUFI.** SQL Processor Using File Input.

**SQL.** Structured Query Language.

**SQL authorization ID (SQL ID).** The authorization ID that is used for checking dynamic SQL statements in some situations.

**SQLCA.** SQL communication area.

**SQL communication area (SQLCA).** A structure that is used to provide an application program with information about the execution of its SQL statements.

**SQLDA.** SQL descriptor area.

**SQL descriptor area (SQLDA).** A structure that describes input variables, output variables, or the columns of a result table.

**SQL/DS™.** Structured Query Language/Data System. This product is now obsolete and has been replaced by DB2 for VSE & VM.

**SQL escape character.** The symbol that is used to enclose an SQL delimited identifier. This symbol is the double quotation mark ("). See also *escape character*.

**SQL function.** A user-defined function in which the CREATE FUNCTION statement contains the source code. The source code is a single SQL expression that evaluates to a single value. The SQL user-defined function can return only one parameter.

**SQL ID.** SQL authorization ID.

**SQL path.** An ordered list of schema names that are used in the resolution of unqualified references to user-defined functions, distinct types, and stored procedures. In dynamic SQL, the current path is found in the CURRENT PATH special register. In static SQL, it is defined in the PATH bind option.

## SQL procedure • table

**SQL procedure.** A user-written program that can be invoked with the SQL CALL statement. Contrast with *external procedure*.

**SQL Processor Using File Input (SPUFI).** SQL Processor Using File Input. A facility of the TSO attachment subcomponent that enables the DB2I user to execute SQL statements without embedding them in an application program.

**SQL return code.** Either SQLCODE or SQLSTATE.

**SQL routine.** A user-defined function or stored procedure that is based on code that is written in SQL.

**SQL string delimiter.** A symbol that is used to enclose an SQL string constant. The SQL string delimiter is the apostrophe ('), except in COBOL applications, where the user assigns the symbol, which is either an apostrophe or a double quotation mark (").

**SSI.** Subsystem interface (in MVS).

**star join.** A method of joining a dimension column of a fact table to the key column of the corresponding dimension table. See also *join*, *dimension*, and *star schema*.

**star schema.** The combination of a fact table (which contains most of the data) and a number of dimension tables. See also *star join*, *dimension*, and *dimension table*.

**statement string.** For a dynamic SQL statement, the character string form of the statement.

**statement trigger.** A trigger that is defined with the trigger granularity FOR EACH STATEMENT.

**static SQL.** SQL statements, embedded within a program, that are prepared during the program preparation process (before the program is executed). After being prepared, the SQL statement does not change (although values of host variables that are specified by the statement might change).

**storage group.** A named set of disks on which DB2 data can be stored.

**stored procedure.** A user-written application program that can be invoked through the use of the SQL CALL statement.

**string.** See *character string* or *graphic string*.

**strong typing.** A process that guarantees that only user-defined functions and operations that are defined on a distinct type can be applied to that type. For example, you cannot directly compare two currency types, such as Canadian dollars and U.S. dollars. But you can provide a user-defined function to convert one currency to the other and then do the comparison.

**Structured Query Language (SQL).** A standardized language for defining and manipulating data in a relational database.

**subject table.** The table for which a trigger is created. When the defined triggering event occurs on this table, the trigger is activated.

**subpage.** The unit into which a physical index page can be divided.

**subquery.** A SELECT statement within the WHERE or HAVING clause of another SQL statement; a nested SQL statement.

**subselect.** That form of a query that does not include ORDER BY clause, UPDATE clause, or UNION operators.

**substitution character.** A unique character that is substituted during character conversion for any characters in the source program that do not have a match in the target coding representation.

**subsystem.** A distinct instance of a relational database management system (RDBMS).

**sync point.** See *commit point*.

**synonym.** In SQL, an alternative name for a table or view. Synonyms can be used only to refer to objects at the subsystem in which the synonym is defined.

**syntactic character set.** A set of 81 graphic characters that are registered in the IBM registry as character set 00640. This set was originally recommended to the programming language community to be used for syntactic purposes toward maximizing portability and interchangeability across systems and country boundaries. It is contained in most of the primary registered character sets, with a few exceptions. See also *invariant character set*.

**system administrator.** The person at a computer installation who designs, controls, and manages the use of the computer system.

**system conversation.** The conversation that two DB2 subsystems must establish to process system messages before any distributed processing can begin.

**system-directed connection.** A connection that an RDBMS manages by processing SQL statements with three-part names.

## T

**table.** A named data object consisting of a specific number of columns and some number of unordered rows. See also *base table* or *temporary table*.

## table check constraint • trigger granularity

**table check constraint.** A user-defined constraint that specifies the values that specific columns of a base table can contain.

**table function.** A function that receives a set of arguments and returns a table to the SQL statement that references the function. A table function can be referenced only in the FROM clause of a subselect.

**table locator.** A mechanism that allows access to trigger transition tables in the FROM clause of SELECT statements, the subselect of INSERT statements, or from within user-defined functions. A table locator is a fullword integer value that represents a transition table.

**table space.** A page set that is used to store the records in one or more tables.

**TB.** Terabyte (1 099 511 627 776 bytes).

**TCP/IP.** A network communication protocol that computer systems use to exchange information across telecommunication links.

**TCP/IP port.** A 2-byte value that identifies an end user or a TCP/IP network application within a TCP/IP host.

**temporary table.** A table that holds temporary data; for example, temporary tables are useful for holding or sorting intermediate results from queries that contain a large number of rows. The two kinds of temporary table, which are created by different SQL statements, are the created temporary table and the declared temporary table. Contrast with *result table*. See also *created temporary table* and *declared temporary table*.

**thread.** The DB2 structure that describes an application's connection, traces its progress, processes resource functions, and delimits its accessibility to DB2 resources and services. Most DB2 functions execute under a thread structure. See also *allied thread* and *database access thread*.

**three-part name.** The full name of a table, view, or alias. It consists of a location name, authorization ID, and an object name, separated by a period.

**time.** A three-part value that designates a time of day in hours, minutes, and seconds.

**time duration.** A decimal integer that represents a number of hours, minutes, and seconds.

**Time-Sharing Option (TSO).** An option in MVS that provides interactive time sharing from remote terminals.

**timestamp.** A seven-part value that consists of a date and time. The timestamp is expressed in years, months, days, hours, minutes, seconds, and microseconds.

**trace.** A DB2 facility that provides the ability to monitor and collect DB2 monitoring, auditing, performance, accounting, statistics, and serviceability (global) data.

**transaction program name.** In SNA LU 6.2 conversations, the name of the program at the remote logical unit that is to be the other half of the conversation.

**transition table.** A temporary table that contains all the affected rows of the subject table in their state before or after the triggering event occurs. Triggered SQL statements in the trigger definition can reference the table of changed rows in the old state or the new state.

**transition variable.** A variable that contains a column value of the affected row of the subject table in its state before or after the triggering event occurs. Triggered SQL statements in the trigger definition can reference the set of old values or the set of new values.

**trigger.** A set of SQL statements that are stored in a DB2 database and executed when a certain event occurs in a DB2 table.

**trigger activation.** The process that occurs when the trigger event that is defined in a trigger definition is executed. Trigger activation consists of the evaluation of the triggered action condition and conditional execution of the triggered SQL statements.

**trigger activation time.** An indication in the trigger definition of whether the trigger should be activated before or after the triggered event.

**trigger body.** The set of SQL statements that is executed when a trigger is activated and its triggered action condition evaluates to true.

**trigger cascading.** The process that occurs when the triggered action of a trigger causes the activation of another trigger.

**triggered action.** The SQL logic that is performed when a trigger is activated. The triggered action consists of an optional triggered action condition and a set of triggered SQL statements that are executed only if the condition evaluates to true.

**triggered action condition.** An optional part of the triggered action. This Boolean condition appears as a WHEN clause and specifies a condition that DB2 evaluates to determine if the triggered SQL statements should be executed.

**triggered SQL statements.** The set of SQL statements that is executed when a trigger is activated and its triggered action condition evaluates to true. Triggered SQL statements are also called the *trigger body*.

**trigger granularity.** A characteristic of a trigger, which determines whether the trigger is activated:

- Only once for the triggering SQL statement
- Once for each row that the SQL statement modifies

## triggering event • varying-length string

**triggering event.** The specified operation in a trigger definition that causes the activation of that trigger. The triggering event is comprised of a triggering operation (INSERT, UPDATE, or DELETE) and a subject table on which the operation is performed.

**triggering SQL operation.** The SQL operation that causes a trigger to be activated when performed on the subject table.

**trigger package.** A package that is created when a CREATE TRIGGER statement is executed. The package is executed when the trigger is activated.

**TSO.** Time-Sharing Option.

**typed parameter marker.** A parameter marker that is specified along with its target data type. It has the general form:

CAST(?) AS data-type)

**type 1 indexes.** Indexes that were created by a release of DB2 before DB2 Version 4 or that are specified as type 1 indexes in Version 4. Contrast with **type 2 indexes**. As of Version 7, type 1 indexes are no longer supported.

**type 2 indexes.** Indexes that are created on a release of DB2 after Version 6 or that are specified as type 2 indexes in Version 4 or later.

## U

**UCS-2.** Universal Character Set, coded in 2 octets, which means that characters are represented in 16-bits per character.

**UDF.** User-defined function.

**UDT.** User-defined data type. In DB2 for OS/390 and z/OS, the term *distinct type* is used instead of user-defined data type. See *distinct type*.

**uncommitted read (UR).** The isolation level that allows an application to read uncommitted data.

**underlying view.** The view on which another view is directly or indirectly defined.

**Unicode.** A standard that parallels the ISO-10646 standard. Several implementations of the Unicode standard exist, all of which have the ability to represent a large percentage of the characters contained in the many scripts that are used throughout the world.

**union.** An SQL operation that combines the results of two select statements. Unions are often used to merge lists of values that are obtained from several tables.

**unique constraint.** An SQL rule that no two values in a primary key, or in the key of a unique index, can be the same.

**unique index.** An index which ensures that no identical key values are stored in a table.

**unit of recovery.** A recoverable sequence of operations within a single resource manager, such as an instance of DB2. Contrast with *unit of work*.

**unit of work.** A recoverable sequence of operations within an application process. At any time, an application process is a single unit of work, but the life of an application process can involve many units of work as a result of commit or rollback operations. In a *multisite update* operation, a single unit of work can include several *units of recovery*. Contrast with *unit of recovery*.

**untyped parameter marker.** A parameter marker that is specified without its target data type. It has the form of a single question mark (?).

**update trigger.** A trigger that is defined with the triggering SQL operation UPDATE.

**UR.** Uncommitted read.

**user-defined data type (UDT).** See *distinct type*.

**user-defined function (UDF).** A function that is defined to DB2 by using the CREATE FUNCTION statement and that can be referenced thereafter in SQL statements. A user-defined function can be an *external function*, a *sourced function*, or an *SQL function*. Contrast with *built-in function*.

**UT.** Utility-only access.

**UTF-8.** Unicode Transformation Format, 8-bit encoding form, which is designed for ease of use with existing ASCII-based systems. The CCSID value for data in UTF-8 format is 1208. DB2 for OS/390 and z/OS supports UTF-8 in mixed data fields.

**UTF-16.** Unicode Transformation Format, 16-bit encoding form, which is designed to provide code values for over a million characters and a superset of UCS-2. The CCSID value for data in UTF-16 format is 1200. DB2 for OS/390 and z/OS supports UTF-16 in graphic data fields.

## V

**value.** The smallest unit of data that is manipulated in SQL.

**variable.** A data element that specifies a value that can be changed. A COBOL elementary data item is an example of a variable. Contrast with *constant*.

**variant function.** See *not-deterministic function*.

**varying-length string.** A character or graphic string whose length varies within set limits. Contrast with *fixed-length string*.

**version.** A member of a set of similar programs, DBRMs, packages, or LOBs.

**A version of a program** is the source code that is produced by precompiling the program. The program version is identified by the program name and a timestamp (consistency token).

**A version of a DBRM** is the DBRM that is produced by precompiling a program. The DBRM version is identified by the same program name and timestamp as a corresponding program version.

**A version of a package** is the result of binding a DBRM within a particular database system. The package version is identified by the same program name and consistency token as the DBRM.

**A version of a LOB** is a copy of a LOB value at a point in time. The version number for a LOB is stored in the auxiliary index entry for the LOB.

**view.** An alternative representation of data from one or more tables. A view can include all or some of the columns that are contained in tables on which it is defined.

**view check option.** An option that specifies whether every row that is inserted or updated through a view must conform to the definition of that view. A view check option can be specified with the WITH CASCADED CHECK OPTION, WITH CHECK OPTION, or WITH LOCAL CHECK OPTION clauses of the CREATE VIEW statement.

**Virtual Storage Access Method (VSAM).** An access method for direct or sequential processing of fixed- and varying-length records on direct access devices. The records in a VSAM data set or file can be organized in logical sequence by a key field (key sequence), in the physical sequence in which they are written on the data set or file (entry-sequence), or by relative-record number.

**Virtual Telecommunications Access Method (VTAM).** An IBM licensed program that controls communication and the flow of data in an SNA network.

**VSAM.** Virtual storage access method.

**VTAM.** Virtual Telecommunication Access Method (in MVS).

## Z

**z/OS.** An operating system for the eServer product line that supports 64-bit real storage.



# Bibliography

## DB2 Universal Database Server for OS/390 and z/OS Version 7 product libraries:

### DB2 for OS/390 and z/OS

- *An Introduction to DB2 for OS/390, SC26-9937*
- *DB2 Administration Guide, SC26-9931*
- *DB2 Application Programming and SQL Guide, SC26-9933*
- *DB2 Application Programming Guide and Reference for Java, SC26-9932*
- *DB2 Command Reference, SC26-9934*
- *DB2 Data Sharing: Planning and Administration, SC26-9935*
- *DB2 Data Sharing Quick Reference Card, SX26-3846*
- *DB2 Diagnosis Guide and Reference, LY37-3740*
- *DB2 Diagnostic Quick Reference Card, LY37-3741*
- *DB2 Image, Audio, and Video Extenders Administration and Programming, SC26-9947*
- *DB2 Installation Guide, GC26-9936*
- *DB2 Licensed Program Specifications, GC26-9938*
- *DB2 Master Index, SC26-9939*
- *DB2 Messages and Codes, GC26-9940*
- *DB2 ODBC Guide and Reference, SC26-9941*
- *DB2 Reference for Remote DRDA Requesters and Servers, SC26-9942*
- *DB2 Reference Summary, SX26-3847*
- *DB2 Release Planning Guide, SC26-9943*
- *DB2 SQL Reference, SC26-9944*
- *DB2 Text Extender Administration and Programming, SC26-9948*
- *DB2 Utility Guide and Reference, SC26-9945*
- *DB2 What's New? GC26-9946*
- *DB2 XML Extender for OS/390 and z/OS Administration and Programming, SC27-9949*
- *DB2 Program Directory, GI10-8182*

### DB2 Administration Tool

- *DB2 Administration Tool for OS/390 and z/OS User's Guide, SC26-9847*

### DB2 Buffer Pool Tool

- *DB2 Buffer Pool Tool for OS/390 and z/OS User's Guide and Reference, SC26-9306*

### DB2 DataPropagator™

- *DB2 UDB Replication Guide and Reference, SC26-9920*

### Net.Data®

The following books are available at this Web site:  
[www.ibm.com/software/net.data/](http://www.ibm.com/software/net.data/)

- *Net.Data Library: Administration and Programming Guide for OS/390 and z/OS*
- *Net.Data Library: Language Environment Interface Reference*
- *Net.Data Library: Messages and Codes*
- *Net.Data Library: Reference*

### DB2 PM for OS/390

- *DB2 PM for OS/390 Batch User's Guide, SC27-0857*
- *DB2 PM for OS/390 Command Reference, SC27-0855*
- *DB2 PM for OS/390 Data Collector Application Programming Interface Guide, SC27-0861*
- *DB2 PM for OS/390 General Information, GC27-0852*
- *DB2 PM for OS/390 Installation and Customization, SC27-0860*
- *DB2 PM for OS/390 Messages, SC27-0856*
- *DB2 PM for OS/390 Online Monitor User's Guide, SC27-0858*
- *DB2 PM for OS/390 Report Reference Volume 1, SC27-0853*
- *DB2 PM for OS/390 Report Reference Volume 2, SC27-0854*
- *DB2 PM for OS/390 Using the Workstation Online Monitor, SC27-0859*
- *DB2 PM for OS/390 Program Directory, GI10-8223*

### Query Management Facility (QMF)

- *Query Management Facility: Developing QMF Applications, SC26-9579*
- *Query Management Facility: Getting Started with QMF on Windows, SC26-9582*
- *Query Management Facility: High Performance Option User's Guide for OS/390 and z/OS, SC26-9581*

## Bibliography

- *Query Management Facility: Installing and Managing QMF on OS/390 and z/OS, GC26-9575*
- *Query Management Facility: Installing and Managing QMF on Windows, GC26-9583*
- *Query Management Facility: Introducing QMF, GC26-9576*
- *Query Management Facility: Messages and Codes, GC26-9580*
- *Query Management Facility: Reference, SC26-9577*
- *Query Management Facility: Using QMF, SC26-9578*

### Ada/370

- *IBM Ada/370 Language Reference, SC09-1297*
- *IBM Ada/370 Programmer's Guide, SC09-1414*
- *IBM Ada/370 SQL Module Processor for DB2 Database Manager User's Guide, SC09-1450*

### APL2®

- *APL2 Programming Guide, SH21-1072*
- *APL2 Programming: Language Reference, SH21-1061*
- *APL2 Programming: Using Structured Query Language (SQL), SH21-1057*

### AS/400®

The following books are available at this Web site:  
[www.as400.ibm.com/infocenter](http://www.as400.ibm.com/infocenter)

- *DB2 Universal Database for AS/400 Database Programming*
- *DB2 Universal Database for AS/400 Performance and Query Optimization*
- *DB2 Universal Database for AS/400 Distributed Data Management*
- *DB2 Universal Database for AS/400 Distributed Data Programming*
- *DB2 Universal Database for AS/400 SQL Programming Concepts*
- *DB2 Universal Database for AS/400 SQL Programming with Host Languages*
- *DB2 Universal Database for AS/400 SQL Reference*

### BASIC

- *IBM BASIC/MVS Language Reference, GC26-4026*
- *IBM BASIC/MVS Programming Guide, SC26-4027*

### BookManager® READ/MVS

- *BookManager READ/MVS V1R3: Installation Planning & Customization, SC38-2035*

### SAA® AD/Cycle® C/370™

- *IBM SAA AD/Cycle C/370 Programming Guide, SC09-1841*
- *IBM SAA AD/Cycle C/370 Programming Guide for Language Environment/370, SC09-1840*
- *IBM SAA AD/Cycle C/370 User's Guide, SC09-1763*
- *SAA CPI C Reference, SC09-1308*

### Character Data Representation Architecture

- *Character Data Representation Architecture Overview, GC09-2207*
- *Character Data Representation Architecture Reference and Registry, SC09-2190*

### CICS/ESA

- *CICS/ESA Application Programming Guide, SC33-1169*
- *CICS External Interfaces Guide, SC33-1944*
- *CICS for MVS/ESA Application Programming Reference, SC33-1170*
- *CICS for MVS/ESA CICS-RACF Security Guide, SC33-1185*
- *CICS for MVS/ESA CICS-Supplied Transactions, SC33-1168*
- *CICS for MVS/ESA Customization Guide, SC33-1165*
- *CICS for MVS/ESA Data Areas, LY33-6083*
- *CICS for MVS/ESA Installation Guide, SC33-1163*
- *CICS for MVS/ESA Intercommunication Guide, SC33-1181*
- *CICS for MVS/ESA Messages and Codes, GC33-1177*
- *CICS for MVS/ESA Operations and Utilities Guide, SC33-1167*
- *CICS/ESA Performance Guide, SC33-1183*
- *CICS/ESA Problem Determination Guide, SC33-1176*
- *CICS for MVS/ESA Resource Definition Guide, SC33-1166*
- *CICS for MVS/ESA System Definition Guide, SC33-1164*
- *CICS for MVS/ESA System Programming Reference, GC33-1171*

### CICS Transaction Server for OS/390

- *CICS Application Programming Guide, SC33-1687*
- *CICS External Interfaces Guide, SC33-1703*
- *CICS DB2 Guide, SC33-1939*
- *CICS Resource Definition Guide, SC33-1684*

### IBM C/C++ for MVS/ESA

- *IBM C/C++ for MVS/ESA Library Reference, SC09-1995*
- *IBM C/C++ for MVS/ESA Programming Guide, SC09-1994*

### IBM COBOL

- *IBM COBOL Language Reference, SC26-4769*
- *IBM COBOL for MVS & VM Programming Guide, SC26-9049*

### Conversion Guide

- *IMS-DB and DB2 Migration and Coexistence Guide, GH21-1083*

### Cooperative Development Environment

- *CoOperative Development Environment/370: Debug Tool, SC09-1623*

### DataPropagator NonRelational

- *DataPropagator NonRelational MVS/ESA Administration Guide, SH19-5036*
- *DataPropagator NonRelational MVS/ESA Reference, SH19-5039*

### Data Facility Data Set Services

- *Data Facility Data Set Services: User's Guide and Reference, SC26-4388*

### Database Design

- *DB2 Design and Development Guide* by Gabrielle Wiorkowski and David Kull, Addison Wesley, ISBN 0-20158-049-7
- *Handbook of Relational Database Design* by C. Fleming and B. Von Halle, Addison Wesley, ISBN 0-20111-434-8

### DataHub®

- *IBM DataHub General Information, GC26-4874*

### Data Refresher

- *Data Refresher Relational Extract Manager for MVS GI10-9927*

### DB2 Connect®

- *DB2 Connect Enterprise Edition for OS/2 and Windows: Quick Beginnings, GC09-2953*
- *DB2 Connect Enterprise Edition for UNIX: Quick Beginnings, GC09-2952*
- *DB2 Connect Personal Edition Quick Beginnings, GC09-2967*
- *DB2 Connect User's Guide, SC09-2954*

### DB2 Red Books

- *DB2 UDB Server for OS/390 Version 6 Technical Update, SG24-6108-00*

### DB2 Server for VSE & VM

- *DB2 Server for VM: DBS Utility, SC09-2394*
- *DB2 Server for VSE: DBS Utility, SC09-2395*

### DB2 Universal Database for UNIX®, Windows®, OS/2®

- *DB2 UDB Administration Guide: Planning, SC09-2946*
- *DB2 UDB Administration Guide: Implementation, SC09-2944*
- *DB2 UDB Administration Guide: Performance, SC09-2945*
- *DB2 UDB Administrative API Reference, SC09-2947*
- *DB2 UDB Application Development Guide, Volume 3, SC09-2948*
- *DB2 UDB Application Development Guide, SC09-2949*
- *DB2 UDB CLI Guide and Reference, SC09-2950*
- *DB2 UDB Command Reference, SC09-2951*
- *DB2 UDB SQL Getting Started, SC09-2973*
- *DB2 UDB SQL Reference Volume 1, SC09-2974*
- *DB2 UDB SQL Reference Volume 2, SC09-2975*

### Device Support Facilities

- *Device Support Facilities User's Guide and Reference, GC35-0033*

### DFSMS

These books provide information about a variety of components of DFSMS, including DFSMS/MVS®, DFSMSdfp™, DFSMSdss™, DFSMShsm™, and MVS/DFP™.

- *DFSMS/MVS: Access Method Services for the Integrated Catalog, SC26-4906*
- *DFSMS/MVS: Access Method Services for VSAM Catalogs, SC26-4905*
- *DFSMS/MVS: Administration Reference for DFSMSdss, SC26-4929*
- *DFSMS/MVS: DFSMShsm Managing Your Own Data, SH21-1077*
- *DFSMS/MVS: Diagnosis Reference for DFSMSdfp, LY27-9606*
- *DFSMS/MVS Storage Management Library: Implementing System-Managed Storage, SC26-3123*
- *DFSMS/MVS: Macro Instructions for Data Sets, SC26-4913*
- *DFSMS/MVS: Managing Catalogs, SC26-4914*
- *z/OS MVS: Program Management User's Guide and Reference, SA22-7643*

## Bibliography

- *DFSMS/MVS: Storage Administration Reference for DFSMSdfp, SC26-4920*
- *DFSMS/MVS: Using Advanced Services, SC26-4921*
- *DFSMS/MVS: Utilities, SC26-4926*
- *OS/390 DFSMS: Using Data Sets, SC26-4749*

### DFSORT™

- *DFSORT Application Programming: Guide, SC33-4035*

### Distributed Relational Database Architecture™

- *Data Stream and OPA Reference, SC31-6806*
- *IBM SQL Reference, SC26-8416*
- *Open Group Technical Standard*

The Open Group presently makes the following DRDA® books available through its Web site at: [www.opengroup.org](http://www.opengroup.org)

- *DRDA Version 2 Vol. 1: Distributed Relational Database Architecture (DRDA)*
- *DRDA Version 2 Vol. 2: Formatted Data Object Content Architecture*
- *DRDA Version 2 Vol. 3: Distributed Data Management Architecture*

### Domain Name System

- *DNS and BIND, Third Edition, Paul Albitz and Cricket Liu, O'Reilly, ISBN 1-56592-512-2*

### Education

- *IBM Dictionary of Computing, McGraw-Hill, ISBN 0-07031-489-6*
- *1999 IBM All-in-One Education and Training Catalog, GR23-8105*

### Enterprise System/9000® and Enterprise System/3090™

- *Enterprise System/9000 and Enterprise System/3090 Processor Resource/System Manager Planning Guide, GA22-7123*

### High Level Assembler

- *High Level Assembler for MVS and VM and VSE Language Reference, SC26-4940*
- *High Level Assembler for MVS and VM and VSE Programmer's Guide, SC26-4941*

### Parallel Sysplex® Library

- *OS/390 Parallel Sysplex Application Migration, GC28-1863*
- *System/390 MVS Sysplex Hardware and Software Migration, GC28-1862*
- *OS/390 Parallel Sysplex Overview: An Introduction to Data Sharing and Parallelism, GC28-1860*

- *OS/390 Parallel Sysplex Systems Management, GC28-1861*
- *OS/390 Parallel Sysplex Test Report, GC28-1963*
- *System/390 9672/9674 System Overview, GA22-7148*

### ICSF/MVS

- *ICSF/MVS General Information, GC23-0093*

### IMS

- *IMS Batch Terminal Simulator General Information, GH20-5522*
- *IMS Administration Guide: System, SC26-9420*
- *IMS Administration Guide: Transaction Manager, SC26-9421*
- *IMS Application Programming: Database Manager, SC26-9422*
- *IMS Application Programming: Design Guide, SC26-9423*
- *IMS Application Programming: Transaction Manager, SC26-9425*
- *IMS Command Reference, SC26-9426*
- *IMS Customization Guide, SC26-9427*
- *IMS Install Volume 1: Installation and Verification, GC26-9429*
- *IMS Install Volume 2: System Definition and Tailoring, GC26-9430*
- *IMS Messages and Codes, GC27-1120*
- *IMS Open Transaction Manager Access Guide and Reference, SC18-7829*
- *IMS Utilities Reference: System, SC26-9441*

### ISPF

- *ISPF V4 Dialog Developer's Guide and Reference, SC34-4486*
- *ISPF V4 Messages and Codes, SC34-4450*
- *ISPF V4 Planning and Customizing, SC34-4443*
- *ISPF V4 User's Guide, SC34-4484*

### Language Environment®

- *Debug Tool User's Guide and Reference, SC09-2137*

### MQSeries

- *MQSeries Application Messaging Interface, SC34-5604*
- *MQSeries for OS/390 Concepts and Planning Guide, GC34-5650*
- *MQSeries for OS/390 System Setup Guide, SC34-5651*

### National Language Support

- *IBM National Language Support Reference Manual Volume 2, SE09-8002*

### NetView®

- *NetView Installation and Administration Guide, SC31-8043*
- *NetView User's Guide, SC31-8056*

### Microsoft ODBC

- *Microsoft ODBC 3.0 Software Development Kit and Programmer's Reference, Microsoft Press, ISBN 1-57231-516-4*

### OS/390 and z/OS

- *OS/390 C/C++ Programming Guide, SC09-2362*
- *OS/390 C/C++ Run-Time Library Reference, SC28-1663*
- *OS/390 C/C++ User's Guide, SC09-2361*
- *OS/390 eNetwork Communications Server: IP Configuration, SC31-8513*
- *OS/390 Hardware Configuration Definition Planning, GC28-1750*
- *OS/390 Information Roadmap, GC28-1727*
- *OS/390 Introduction and Release Guide, GC28-1725*
- *OS/390 JES2 Initialization and Tuning Guide, SC28-1791*
- *OS/390 JES3 Initialization and Tuning Guide, SC28-1802*
- *OS/390 Language Environment for OS/390 & VM Concepts Guide, GC28-1945*
- *OS/390 Language Environment for OS/390 & VM Customization, SC28-1941*
- *OS/390 Language Environment for OS/390 & VM Debugging Guide, SC28-1942*
- *OS/390 Language Environment for OS/390 & VM Programming Guide, SC28-1939*
- *OS/390 Language Environment for OS/390 & VM Programming Reference, SC28-1940*
- *OS/390 MVS Diagnosis: Procedures, LY28-1082*
- *OS/390 MVS Diagnosis: Reference, SY28-1084*
- *OS/390 MVS Diagnosis: Tools and Service Aids, LY28-1085*
- *OS/390 MVS Initialization and Tuning Guide, SC28-1751*
- *OS/390 MVS Initialization and Tuning Reference, SC28-1752*
- *OS/390 MVS Installation Exits, SC28-1753*
- *OS/390 MVS JCL Reference, GC28-1757*
- *OS/390 MVS JCL User's Guide, GC28-1758*
- *OS/390 MVS Planning: Global Resource Serialization, GC28-1759*
- *OS/390 MVS Planning: Operations, GC28-1760*
- *OS/390 MVS Planning: Workload Management, GC28-1761*
- *OS/390 MVS Programming: Assembler Services Guide, GC28-1762*

- *OS/390 MVS Programming: Assembler Services Reference, GC28-1910*
- *OS/390 MVS Programming: Authorized Assembler Services Guide, GC28-1763*
- *OS/390 MVS Programming: Authorized Assembler Services Reference, Volumes 1-4, GC28-1764, GC28-1765, GC28-1766, GC28-1767*
- *OS/390 MVS Programming: Callable Services for High-Level Languages, GC28-1768*
- *OS/390 MVS Programming: Extended Addressability Guide, GC28-1769*
- *OS/390 MVS Programming: Sysplex Services Guide, GC28-1771*
- *OS/390 MVS Programming: Sysplex Services Reference, GC28-1772*
- *OS/390 MVS Programming: Workload Management Services, GC28-1773*
- *OS/390 MVS Routing and Descriptor Codes, GC28-1778*
- *OS/390 MVS Setting Up a Sysplex, GC28-1779*
- *OS/390 MVS System Codes, GC28-1780*
- *OS/390 MVS System Commands, GC28-1781*
- *OS/390 MVS System Messages Volume 1, GC28-1784*
- *OS/390 MVS System Messages Volume 2, GC28-1785*
- *OS/390 MVS System Messages Volume 3, GC28-1786*
- *OS/390 MVS System Messages Volume 4, GC28-1787*
- *OS/390 MVS System Messages Volume 5, GC28-1788*
- *OS/390 MVS Using the Subsystem Interface, SC28-1789*
- *OS/390 SecureWay Security Server Network Authentication and Privacy Service Administration, SC24-5896*
- *OS/390 Security Server External Security Interface (RACROUTE) Macro Reference, GC28-1922*
- *OS/390 Security Server (RACF) Auditor's Guide, SC28-1916*
- *OS/390 Security Server (RACF) Command Language Reference, SC28-1919*
- *OS/390 Security Server (RACF) General User's Guide, SC28-1917*
- *OS/390 Security Server (RACF) Introduction, GC28-1912*
- *OS/390 Security Server (RACF) Macros and Interfaces, SK2T-6700 (OS/390 Collection Kit ), SK2T-2180 (OS/390 Security Server Information Package )*
- *OS/390 Security Server (RACF) Security Administrator's Guide, SC28-1915*

## Bibliography

- *OS/390 Security Server (RACF) System Programmer's Guide, SC28-1913*
- *OS/390 SMP/E Reference, SC28-1806*
- *OS/390 SMP/E User's Guide, SC28-1740*
- *OS/390 V2 R8/R9/R10 Support for Unicode: Using Conversion Services, SC33-7050*
- *OS/390 RMF User's Guide, SC28-1949*
- *OS/390 TSO/E CLISTS, SC28-1973*
- *OS/390 TSO/E Command Reference, SC28-1969*
- *OS/390 TSO/E Customization, SC28-1965*
- *OS/390 TSO/E Messages, GC28-1978*
- *OS/390 TSO/E Programming Guide, SC28-1970*
- *OS/390 TSO/E Programming Services, SC28-1971*
- *OS/390 TSO/E REXX Reference, SC28-1975*
- *OS/390 TSO/E User's Guide, SC28-1968*
- *OS/390 DCE Administration Guide, SC28-1584*
- *OS/390 DCE Introduction, GC28-1581*
- *OS/390 DCE Messages and Codes, SC28-1591*
- *OS/390 UNIX System Services Command Reference, SC28-1892*
- *OS/390 UNIX System Services Messages and Codes, SC28-1908*
- *OS/390 UNIX System Services Planning, SC28-1890*
- *OS/390 UNIX System Services User's Guide, SC28-1891*
- *OS/390 UNIX System Services Programming: Assembler Callable Services Reference, SC28-1899*
- *OS/390 V2R10.0 IBM CS IP Configuration Reference, SC31-8726*
- *Program Directory for OS/390 V2 R8/R9/R10 support for Unicode, GI10-9760*
- *z/OS Support for Unicode: Using Conversion Services, SA22-7649*

### IBM Enterprise PL/I for z/OS and OS/390

- *IBM Enterprise PL/I for z/OS and OS/390 Language Reference, SC26-1460*
- *IBM Enterprise PL/I for z/OS and OS/390 Programming Guide, SC26-1457*

### PL/I for MVS & VM

- *PL/I for MVS & VM Programming Language Reference, SC26-3114*
- *PL/I for MVS & VM Programming Guide, SC26-3113*

### Prolog

- *IBM SAA AD/Cycle Prolog/MVS & VM Programmer's Guide, SH19-6892*

### RAMAC® and Enterprise Storage Server™

- *IBM RAMAC Virtual Array, SG24-4951*
- *RAMAC Virtual Array: Implementing Peer-to-Peer Remote Copy, SG24-5338*
- *Enterprise Storage Server Introduction and Planning, GC26-7294*

### Remote Recovery Data Facility

- *Remote Recovery Data Facility Program Description and Operations, LY37-3710*

### Storage Management

- *DFSMS/MVS Storage Management Library: Implementing System-Managed Storage, SC26-3123*
- *MVS/ESA Storage Management Library: Leading a Storage Administration Group, SC26-3126*
- *MVS/ESA Storage Management Library: Managing Data, SC26-3124*
- *MVS/ESA Storage Management Library: Managing Storage Groups, SC26-3125*
- *MVS Storage Management Library: Storage Management Subsystem Migration Planning Guide, SC26-4659*

### System/370™ and System/390

- *ESA/370 Principles of Operation, SA22-7200*
- *ESA/390 Principles of Operation, SA22-7201*
- *System/390 MVS Sysplex Hardware and Software Migration, GC28-1210*

### System Network Architecture (SNA)

- *SNA Formats, GA27-3136*
- *SNA LU 6.2 Peer Protocols Reference, SC31-6808*
- *SNA Transaction Programmer's Reference Manual for LU Type 6.2, GC30-3084*
- *SNA/Management Services Alert Implementation Guide, GC31-6809*

### TCP/IP

- *IBM TCP/IP for MVS: Customization & Administration Guide, SC31-7134*
- *IBM TCP/IP for MVS: Diagnosis Guide, LY43-0105*
- *IBM TCP/IP for MVS: Messages and Codes, SC31-7132*
- *IBM TCP/IP for MVS: Planning and Migration Guide, SC31-7189*

### VS COBOL II

- *VS COBOL II Application Programming Guide for MVS and CMS, SC26-4045*
- *VS COBOL II Application Programming: Language Reference, GC26-4047*

- *VS COBOL II Installation and Customization for MVS, SC26-4048*

### **VS Fortran**

- *VS Fortran Version 2: Language and Library Reference, SC26-4221*
- *VS Fortran Version 2: Programming Guide for CMS and MVS, SC26-4222*

### **VTAM**

- *Planning for NetView, NCP, and VTAM, SC31-8063*
- *VTAM for MVS/ESA Diagnosis, LY43-0069*
- *VTAM for MVS/ESA Messages and Codes, SC31-6546*
- *VTAM for MVS/ESA Network Implementation Guide, SC31-6548*
- *VTAM for MVS/ESA Operation, SC31-6549*
- *z/OS Communications Server SNA Programming, SC31-8829*
- *z/OS Communicatons Server SNA Programmer's LU 6.2 Reference, SC31-8810*
- *VTAM for MVS/ESA Resource Definition Reference, SC31-6552*

## Bibliography

# Index

## Special characters

\_ (underscore character) as escape character 138, 141  
, (comma) as decimal point 148  
: (colon)  
    preceding a host variable 101  
! (exclamation mark) as not sign 130  
? (question mark) 776  
/ (divide sign) 114  
. (period) as decimal point 148  
'string' clause  
    ALTER FUNCTION statement 397  
    ALTER PROCEDURE statement 430  
    CREATE FUNCTION statement 540  
    CREATE PROCEDURE statement 626, 642  
\* (asterisk)  
    COUNT function 161  
    COUNT\_BIG function 161  
    multiply sign 114  
    use in subselect 350  
- (minus sign) 114  
% (percent sign) as escape character 138, 141  
|| (vertical bars) 112  
+ (plus sign) 114  
+ (plus sign) as escape character 138, 141

## A

ABS function 174  
ACCESSPATH column  
    SYSPACKSTMT catalog table 1082  
    SYSSTMT catalog table 1112  
ACOS function 175  
ACQUIRE  
    column of SYSPLAN catalog table 1087  
ADD  
    clause of ALTER TABLE statement 452  
ADD VOLUMES clause of ALTER STOGROUP statement 444  
ADD\_MONTHS function 176  
AFTER clause of CREATE TRIGGER statement 701  
alias  
    creating 514  
    description 41  
    dropping 765  
    naming convention 34  
    qualifying a column name 95  
    referencing another DB2 16  
    retrieving catalog information about 1145  
    unqualified name 39  
ALIAS clause  
    COMMENT statement 495  
    CREATE ALIAS statement 514  
    DROP statement 765  
    LABEL ON statement 838  
ALL  
    clause of RELEASE statement 860

ALL (*continued*)  
    clause of subselect 349  
    keyword  
        AVG function 162  
        column functions 161  
        COUNT function 163  
        MAX function 166  
        MIN function 167  
        SUM function 170  
    quantified predicate 132  
ALL PRIVILEGES clause  
    GRANT statement 824  
    REVOKE statement 889  
ALL SQL clause of RELEASE statement 860  
ALLOCATE CURSOR statement  
    description 386  
    example 387  
ALLOW PARALLEL clause  
    ALTER FUNCTION statement 403  
    CREATE FUNCTION statement 546  
alphabetic extender 31, 34  
ALTDATETIME function 1156  
ALTER DATABASE statement  
    description 388  
    example 390  
ALTER FUNCTION (external scalar) statement  
    description 391  
    example 407  
ALTER FUNCTION (SQL scalar) statement  
    description 408  
    example 413  
ALTER INDEX statement  
    description 414  
    example 425  
ALTER privilege  
    GRANT statement 824  
    REVOKE statement 889  
ALTER PROCEDURE (external) statement  
    description 427  
    example 437  
ALTER PROCEDURE (SQL) statement  
    description 438  
    example 443  
ALTER STOGROUP statement  
    description 444  
    example 446  
ALTER TABLE statement  
    description 447  
    example 466  
ALTER TABLESPACE statement  
    description 467  
    example 478  
ALTERAUTH column of SYSTABAUTH catalog table 1118  
ALTEREDTS column  
    SYSDATABASE catalog table 1044  
    SYSINDEXES catalog table 1055  
    SYSINDEXPART catalog table 1059

**ALTEREDTS** column (*continued*)
   
 SYSJAROBJECTS catalog table 1067
   
 SYROUTINES catalog table 1103
   
 SYSSEQUENCES catalog table 1109
   
 SYSSTOGROUP catalog table 1114
   
 SYSTABLEPART catalog table 1123
   
 SYSTABLES catalog table 1128
   
 SYSTABLESPACE catalog table 1132
   
**ALTERIN** privilege
   
 GRANT statement 819
   
 REVOKE statement 884
   
**ALTERINAUTH** column of SYSSCHEMAAUTH catalog table 1108
   
**ALTTIME** function 1159
   
**AND**

- truth table 145

**ANY**

- quantified predicate 132
- USING clause of DESCRIBE statement 750
- USING clause of PREPARE statement 848

**APOST** option
   
 precompiler 149
   
**apostrophe**

- string delimiter precompiler option 149

**APOSTSQL** option
   
 precompiler 149
   
**application plan**

- description 14
- invalidated
  - ALTER TABLE statement 465

**privileges**

- GRANT statement 818
- REVOKE statement 883

**application process** 11
   
**application program**

- recovery 11
- SQLCA 979
- SQLDA 987

**ARCHIVE** privilege
   
 GRANT statement 821
   
 REVOKE statement 886
   
**ARCHIVEAUTH** column of SYSUSERAUTH catalog table 1139
   
**arithmetic operators** 114
   
**AS** clause
   
 CREATE VIEW statement 713
   
 naming result columns 350
   
 use in subselect 350
   
**AS DEFINITION ONLY** clause
   
 DECLARE GLOBAL TEMPORARY TABLE statement 729
   
**AS IDENTITY** clause
   
 ALTER TABLE statement 455
   
 CREATE TABLE statement 665
   
 DECLARE GLOBAL TEMPORARY TABLE statement 728
   
**AS LOCATOR** clause
   
 CREATE FUNCTION statement 536, 559, 577
   
 CREATE PROCEDURE statement 623
   
**AS TEMP** clause of CREATE DATABASE statement 520
   
**AS WORKFILE** clause of CREATE DATABASE statement 520
   
**ASC** clause
   
 CREATE INDEX statement 605
   
 of select-statement 371
   
**ASCII**

- definition 21
- effect on DBCS characters 49

**ASIN** function 178
   
**assembler application program**

- host variable
  - EXECUTE IMMEDIATE statement 779
  - referencing 100
- INCLUDE SQLCA 983
- INCLUDE SQLDA 998
- varying-length string variables 52

**assignment**

- compatibility rules 64
- datetime values 70
- distinct type values 71
- IEEE floating-point numbers 67
- numbers 66
- retrieval rules 69
- row ID values 71
- statement
  - example 947
  - SQL procedure 946
- storage rules 69
- strings, basic rules for 68

**ASSOCIATE LOCATORS** statement
   
 description 479
   
 example 481
   
**asterisk (\*)**

- COUNT function 163
- COUNT\_BIG function 164
- multiply sign 114
- use in subselect 350

**ASUTIME** clause
   
 ALTER FUNCTION statement 405
   
 ALTER PROCEDURE statement 435, 441
   
 CREATE FUNCTION statement 548, 568
   
 CREATE PROCEDURE statement 631, 645
   
**ASUTIME** column
   
 SYSPROCEDURES catalog table 1095
   
 SYROUTINES catalog table 1102
   
**ATAN** function 179
   
**ATAN2** function 181
   
**ATANH** function 180
   
**ATTRIBUTES** clause
   
 PREPARE statement 848
   
**AUDIT**

- clause of ALTER TABLE statement 463
- clause of CREATE TABLE statement 673

**auditing**

- ALTER TABLE statement 463
- CREATE TABLE statement 673

**AUDITING** column of SYSTABLES catalog table 1127
   
**AUTHHOWGOT** column
   
 SYSDBAAUTH catalog table 1047
   
 SYSPACKAUTH catalog table 1078
   
 SYSPLANAUTH catalog table 1091

AUTHHOWGOT column (*continued*)  
   SYSRESAUTH catalog table 1098  
   SYSROUTINEAUTH catalog table 1099  
   SYSSCHEMAAUTH catalog table 1108  
   SYSTABAUTH catalog table 1118  
   SYSUSERAUTH catalog table 1137

AUTHID  
   column of MODESELECT catalog table 1022  
   column of SYSCOPY catalog table 1043  
   column of SYSPROCEDURES catalog table 1094  
   column of USERNAMES catalog table 1143

authority  
   retrieving catalog information 1147

authorization  
   clause of CONNECT (type 1) statement 505  
   clause of CONNECT (type 2) statement 511

authorization ID  
   description 42  
   primary  
     description 42  
   resulting from errors 901  
   secondary  
     description 42  
   translating  
     concepts 47

AUX clause of CREATE AUXILIARY TABLE  
   statement 517

AUXILIARY clause of CREATE AUXILIARY TABLE  
   statement 517

auxiliary table  
   CREATE AUXILIARY TABLE statement 516  
   description 4

AUXRELOBID column of SYSAUXRELS catalog  
   table 1023

AUXTBNAME column of SYSAUXRELS catalog  
   table 1023

AUXTBOWNER column of SYSAUXRELS catalog  
   table 1023

AVG function 162

AVGROWLEN column  
   SYSTABLES catalog table 1129  
   SYSTABLES\_HIST catalog table 1130

AVGSIZE column  
   SYSLOBSTATS catalog table 1071  
   SYSPACKAGE catalog table 1073  
   SYSPLAN catalog table 1087

**B**

base table  
   definition 4

basic operations in SQL 64

basic predicate 130

BCREATOR column  
   SYSPLANDEP catalog table 1092  
   SYSVIEWDEP catalog table 1140

BEGIN DECLARE SECTION statement  
   description 482  
   example 482

BETWEEN predicate 134

binary large object (BLOB) 53

BINARY LARGE OBJECT data type 53

binary string  
   description 53

bind behavior for dynamic SQL statements 44

BIND PACKAGE subcommand of DSN  
   options  
     QUALIFIER 39, 40

BIND PLAN subcommand of DSN  
   options  
     QUALIFIER 39, 40

BIND privilege  
   GRANT statement 816, 818  
   REVOKE statement 881, 883

bind process 43

BIND\_OPTS column  
   SYSJAVAOPTS catalog table 1068  
   SYSROUTINES\_OPTS catalog table 1106

BINDADD privilege  
   binding a package 46  
   GRANT statement 821  
   REVOKE statement 886

BINDADDAUTH column of SYSUSERAUTH catalog  
   table 1137

BINDAGENT privilege  
   GRANT statement 821  
   REVOKE statement 886

BINDAGENTAUTH column of SYSUSERAUTH catalog  
   table 1138

BINDAUTH column  
   SYSPACKAUTH catalog table 1078  
   SYSPLANAUTH catalog table 1091

BINDEROR column of SYSPACKSTMT catalog  
   table 1081

binding  
   description 2  
   process 43

BINETIME column  
   SYSPACKAGE catalog table 1073

bit data  
   conversion restrictions 24

BIT data  
   description 49

BLOB (binary large object)  
   data type 660  
   description 53

BNAM column  
   SYSCONSTDEP catalog table 1039  
   SYSPACKDEP catalog table 1079  
   SYSPLANDEP catalog table 1092  
   SYSVIEWDEP catalog table 1140

BOTH  
   USING clause of DESCRIBE statement 750

BOUNDBY column of SYSPLAN catalog table 1088

BOUNDTS column  
   SYSPLAN catalog table 1089

BPOOL column  
  SYSDATABASE catalog table 1044  
  SYSINDEXES catalog table 1055  
  SYSTABLESPACE catalog table 1131  
BQUALIFIER column of SYSPACKDEP catalog table 1079  
BSchema column  
  SYSCONSTDEP catalog table 1039  
  SYSVIEWDEP catalog table 1140  
BSDS (bootstrap data set)  
  privilege  
    granting 821  
    revoking 886  
BSDSAUTH column of SYSUSERAUTH catalog table 1137  
BSEQUENCEID column of SYSSEQUENCEDEP catalog table 1110  
BTYPE column  
  SYSCONSTDEP catalog table 1039  
  SYSPACKDEP catalog table 1079  
  SYSPLANDEP catalog table 1092  
  SYSVIEWDEP catalog table 1140  
buffer pool  
  naming convention 35  
BUFFERPOOL clause  
  ALTER DATABASE statement 388  
  ALTER INDEX statement 416  
  ALTER TABLESPACE statement 469  
  CREATE DATABASE statement 519  
  CREATE INDEX statement 612  
  CREATE TABLESPACE statement 692  
BUFFERPOOL privilege  
  GRANT statement 827  
  REVOKE statement 892  
BUILDDATE column  
  SYSROUTINES\_OPTS catalog table 1106  
  SYSROUTINES\_SRC catalog table 1107  
BUILDNAME column  
  SYSJAVAOPTS catalog table 1068  
  SYSROUTINES\_OPTS catalog table 1106  
BUILDOWNER column  
  SYSJAVAOPTS catalog table 1068  
  SYSROUTINES\_OPTS catalog table 1106  
BUILDSchema column  
  SYSJAVAOPTS catalog table 1068  
  SYSROUTINES\_OPTS catalog table 1106  
BUILDSTATUS column  
  SYSROUTINES\_OPTS catalog table 1106  
  SYSROUTINES\_SRC catalog table 1107  
BUILDTIME column  
  SYSROUTINES\_OPTS catalog table 1106  
  SYSROUTINES\_SRC catalog table 1107  
built-in data type 48  
built-in function 155  
  description 104  
BY clause of REVOKE statement 866

**C**  
C application program  
host variable  
  EXECUTE IMMEDIATE statement 779  
  referencing 100  
INCLUDE SQLCA 982  
INCLUDE SQLDA 998  
  varying-length string 52  
CACHE column of SYSSEQUENCES catalog table 1109  
CACHESIZE  
  column of SYSPLAN catalog table 1088  
CALL statement  
  description 483  
  example 490, 949  
  SQL procedure 948  
CALLED ON NULL INPUT clause  
  ALTER FUNCTION statement 400, 413  
  CREATE FUNCTION statement 543, 563, 592  
capturing changed data  
  ALTER TABLE statement 463  
  CREATE TABLE statement 674  
CARD column  
  SYSTABLEPART catalog table  
    description 1121  
  SYTABSTATS catalog table  
    description 1134  
CARDF column  
  SYSCOLDIST catalog table 1028  
  SYSCOLDIST\_HIST catalog table 1029  
  SYSCOLDISTSTATS catalog table 1030  
  SYSINDEXPART catalog table 1059  
  SYSINDEXPART\_HIST catalog table 1061  
  SYSTABLEPART catalog table 1123  
  SYSTABLEPART\_HIST catalog table 1124  
  SYSTABLES catalog table 1128  
  SYSTABLES\_HIST catalog table 1130  
  SYTABSTATS catalog table 1134  
  SYTABSTATS\_HIST catalog table 1135  
CARDINALITY column  
  SYSROUTINES catalog table 1105  
CASCADE delete rule  
  ALTER TABLE statement 460  
  CREATE TABLE statement 670  
  description 7  
cascade revoke 867  
CASE expression  
  description 124  
  result data type 77  
CASE statement  
  example 951  
  SQL procedure 950  
cast function 105  
CAST specification  
  definition 126  
  NULL 127  
  parameter marker 127  
CAST\_FUNCTION column  
  SYSPARMS catalog table 1084  
  SYSROUTINES catalog table 1101

CAST\_FUNCTION\_ID column of SYSPARMS catalog  
 table 1085  
 casts  
   data types 62  
 catalog name  
   naming convention 35  
 VCAT clause  
   ALTER INDEX statement 419  
   CREATE INDEX statement 606  
   CREATE TABLESPACE statement 684, 687  
 catalog tables  
   description 10, 1005  
   indexes 1006  
   IPNAMES 1016  
   LOCATIONS 1017  
   LULIST 1018  
   LUMODES 1019  
   LUNAMES 1020  
   MODESELECT 1022  
   release dependency indicators 1005  
   retrieving information about  
     parent keys 1148  
     status 1149  
   SQL statements allowed 1011  
   SYSAUXRELS 1023, 1149  
   SYSCHECKDEP 1024  
   SYSCHECKS 1025  
   SYSCHECKS2 1026  
   SYSCOLAUTH 1027  
   SYSCOLDIST  
     contents 1028  
   SYSCOLDISTSTATS  
     contents 1030  
   SYSCOLSTATS  
     contents 1031  
   SYSCOLUMNS  
     contents 1032  
     updated by COMMENT statement 1151  
     updated by CREATE VIEW statement 1147  
   SYSCOLUMNS\_HIST  
     contents 1037  
   SYSCONSTDEP 1039  
   SYSCOPY  
     contents 1040  
   SYSDATABASE  
     contents 1044  
   SYSDATATYPES 1046, 1150  
   SYSDBAUTH 1047  
   SYSDBRM 1049  
   SYSDUMMY1 1051  
   SYSFIELDS 1052  
   SYSFOREIGNKEYS 1053, 1148  
   SYSINDEXES  
     contents 1054  
   SYSINDEXES\_HIST  
     contents 1057  
   SYSINDEXPART  
     contents 1058  
   SYSINDEXPART\_HIST 1061  
   SYSINDEXSTATS  
     contents 1062

catalog tables (*continued*)  
   SYSINDEXSTATS\_HIST 1063  
   SYSJARCLASS\_SOURCE 1064  
   SYSJARCONTENTS 1065  
   SYSJARDATA\_SOURCE 1066  
   SYSJAROBJECTS 1067  
   SYSJAVAOPTS 1068  
   SYSKEYCOLUSE 1069  
   SYSKEYS 1070  
   SYSLOBSTATS 1071  
   SYSLOBSTATS\_HIST 1072  
   SYSPACKAGE 1073  
   SYSPACKAUTH 1078  
   SYSPACKDEP 1079  
   SYSPACKLIST 1080  
   SYSPACKSTMT 1081  
   SYSPARMS 1084  
   SYSPKSYSTEM 1086  
   SYSPLAN 1087  
   SYSPLANAUTH  
     contents 1091  
   SYSPLANDEP  
     contents 1092  
   SYSPLSYSTEM 1093  
   SYSPROCEDURES  
     contents 1094  
   SYSRELS  
     contents 1097  
     describes referential constraints 1148  
   SYSRESAUTH 1098  
   SYSROUTINEAUTH 1099  
   SYSROUTINES 1150  
     contents 1100  
   SYSROUTINES\_OPTS 1106  
   SYSROUTINES\_SRC 1107  
   SYSSCHEMAAUTH 1108  
   SYSSEQUENCES 1109  
   SYSSEQUENCESDEP 1110  
   SYSSTMT 1111  
   SYSSTOGROUP  
     contents 1114  
     sample query 1145  
   SYSSTRINGS  
     contents 1115  
   SYSSYNONYMS 1117  
   SYTABAUTH  
     contents 1118  
     table authorizations 1147  
     updated by CREATE VIEW statement 1147  
     view authorizations 1147  
   SYTABCONST  
     contents 1120  
   SYTABLEPART  
     contents 1121  
   SYTABLEPART\_HIST  
     contents 1124  
   SYTABLES  
     contents 1126  
     rows maintained 1145  
     updated by COMMENT statement 1151  
     updated by CREATE VIEW statement 1147

**catalog tables** (*continued*)
   
 SYSTABLES\_HIST
   
 contents 1130
   
 SYSTABLESPACE
   
 contents 1131
   
 SYSTABSTATS
   
 contents 1134
   
 SYTABSTATS\_HIST
   
 contents 1135
   
 SYSTRIGGERS 1136, 1150
   
 SYSUSERAUTH 1137
   
 SYSVIEWDEP
   
 contents 1140
   
 updated by CREATE VIEW statement 1147
   
 SYSVIEWS 1141
   
 SYSVOLUMES 1142
   
 table space 1006
   
 USERNAMES 1143
   
**catalog, DB2**
  
 constraint information 1149
   
 database design 1145, 1151
   
 description 10
   
 retrieving information from 1145
   
 tables 1005
   
**CCSID**
  
 clause of ALTER DATABASE statement 389
   
 clause of ALTER TABLESPACE statement 477
   
 clause of CREATE DATABASE statement 520
   
 clause of CREATE DISTINCT TYPE statement 524
   
 clause of CREATE FUNCTION statement 536, 558, 576, 589
   
 clause of CREATE GLOBAL TEMPORARY TABLE statement 597
   
 clause of CREATE PROCEDURE statement 623, 641
   
 clause of CREATE TABLE statement 674
   
 clause of CREATE TABLESPACE statement 694
   
 clause of DECLARE GLOBAL TEMPORARY TABLE statement 731
   
 column of SYSPARMS catalog table 1085
   
**CCSID (coded character set identifier)**
  
 definition 20
   
 description 21
   
 system 23
   
**CCSID\_ENCODING function** 183
   
**CEIL function** 184
   
**CEILING function** 184
   
**CHAR**
  
 data type 51
   
 function 185
   
**CHAR LARGE OBJECT data type** 51, 660
   
**CHAR VARYING data type** 51, 659
   
**character** 31
   
**character conversion**

- ASCII 21
- assignment rules 70
- character set 21
- code page 21
- code point 21
- coded character set 21
- comparison rules 73

  
**character conversion** (*continued*)
   
 concatenation rules 366
   
 contracting conversion 25
   
 description 20
   
 EBCDIC 21
   
 encoding scheme 21
   
 expanding conversion 24
   
 substitution byte 22
   
 SYSIBM.SYSSTRINGS catalog table 1115
   
 UCS-2 22
   
 Unicode 22
   
 UNION and UNION ALL rules 366
   
 UTF-16 22
   
 UTF-8 22
   
**CHARACTER data type**
  
 CREATE TABLE statement 659
   
 description 51
   
**character large object (CLOB)** 53
   
**CHARACTER LARGE OBJECT data type** 51, 660
   
**character set** 21
   
**character string**

- assignment 68
- comparison 73
- constants 80
- description 49
- empty 49

  
**CHARACTER VARYING data type** 51, 659
   
**CHARSET column**

- SYSDBRM catalog table 1049
- SYSPACKAGE catalog table 1074

  
**CHECK**

- clause of ALTER TABLE statement 461
- clause of CREATE TABLE statement 670
- column of SYSVIEWS catalog table 1141

  
**check constraint**

- description 9

  
**check pending status**

- retrieving catalog information 1149

  
**CHECKCONDITION column**

- SYSCHECKS catalog table 1025

  
**CHECKFLAG column**

- SYSTABLEPART catalog table 1122
- SYSTABLES catalog table 1127

  
**CHECKNAME column**

- SYSCHECKDEP catalog table 1024
- SYSCHECKS catalog table 1025
- SYSCHECKS2 catalog table 1026

  
**CHECKRID5B column**

- SYSTABLEPART catalog table 1123
- SYSTABLES catalog table 1128

  
**CHECKS column**

- SYSTABLES catalog table 1128

  
**CHILDREN column of SYSTABLES catalog table** 1127
   
**CLASS column**

- SYSJARCONTENTS catalog table 1065
- SYSROUTINES catalog table 1105

  
**CLASS\_SOURCE column**

- SYSJARCLASS\_SOURCE catalog table 1064
- SYSJARCONTENTS catalog table 1065

  
**CLASS\_SOURCE\_ROWID column**

- SYSJARCONTENTS catalog table 1065

CLOB (character large object)  
     description 51, 53, 660  
     function 191  
     host variable 101  
     locator 102  
     restrictions 54

CLOSE  
     clause of ALTER INDEX statement 417  
     clause of ALTER TABLESPACE statement 470  
     clause of CREATE INDEX statement  
         description 612  
     clause of CREATE TABLESPACE statement  
         description 693  
     statement  
         description 491  
         example 491

closed state of cursor 844

CLOSERULE column  
     SYSINDEXES catalog table 1055  
     SYSTABLESPACE catalog table 1131

CLUSTER clause  
     CREATE INDEX statement 611

CLUSTERED column of SYSINDEXES catalog table  
     description 1054

CLUSTERING column  
     SYSINDEXES\_HIST catalog table 1057

CLUSTERING column of SYSINDEXES catalog table  
     description 1054

CLUSTERRATIO column  
     SYSINDEXES catalog table 1055  
     SYSINDEXSTATS catalog table 1062

CLUSTERRATIOF column  
     SYSINDEXES catalog table 1056  
     SYSINDEXES\_HIST catalog table 1057  
     SYSINDEXSTATS catalog table 1062  
     SYSINDEXSTATS\_HIST catalog table 1063

CLUSTERTYPE column of SYSTABLES catalog table 1126

CNAME column  
     SYSPKSYSTEM catalog table 1086  
     SYSPLSYSTEM catalog table 1093

COALESCE function 77, 192, 224

COBOL application program  
     host structure 103  
     host variable  
         description 100  
         EXECUTE IMMEDIATE statement 779

INCLUDE SQLCA 984  
     varying-length string 52

code page 21

code point 21

coded character set 21

COLCARDDATA column of SYSCOLSTATS catalog table 1031

COLCARDF column  
     SYSOLUMNS catalog table 1036  
     SYSOLUMNS\_HIST catalog table 1038

COLCOUNT column  
     SYSINDEXES catalog table 1054  
     SYSRELS catalog table 1097  
     SYSTABCONST catalog table 1120

COLCOUNT column (*continued*)  
     SYSTABLES catalog table 1126  
     SYSTABLES\_HIST catalog table 1130

COLGROUPCOLNO column  
     SYSCOLDIST catalog table 1028  
     SYSCOLDIST\_HIST catalog table 1029  
     SYSCOLDISTSTATS catalog table 1030

collection, package  
     granting privileges 806  
     revoking privileges 870  
     SET CURRENT PACKAGESET statement 912

COLLID clause  
     ALTER FUNCTION statement 404  
     ALTER PROCEDURE statement 434, 440  
     CREATE FUNCTION statement 547, 567  
     CREATE PROCEDURE statement 630, 644

COLLID column  
     SYSCOLAUTH catalog table 1027  
     SYSPACKAGE catalog table 1073  
     SYSPACKAUTH catalog table 1078  
     SYSPACKLIST catalog table 1080  
     SYSPACKSTMT catalog table 1081  
     SYSPKSYSTEM catalog table 1086  
     SYSPROCEDURES catalog table 1095  
     SYSROUTINEAUTH catalog table 1099  
     SYSROUTINES catalog table 1100  
     SYSTABAUTH catalog table 1119

COLNAME column  
     SYSAUXRELS catalog table 1023  
     SYSCHECKDEP catalog table 1024  
     SYSCOLAUTH catalog table 1027  
     SYSFOREIGNKEYS catalog table 1053  
     SYSKEYCOLUSE catalog table 1069  
     SYSKEYS catalog table 1070

COLNO column  
     SYSCOLUMNS catalog table 1032  
     SYSCOLUMNS\_HIST catalog table 1037  
     SYSFIELDS catalog table 1052  
     SYSFOREIGNKEYS catalog table 1053  
     SYSKEYCOLUSE catalog table 1069  
     SYSKEYS catalog table 1070

colon  
     host variable in SQL 101

COLSEQ column  
     SYSFOREIGNKEYS catalog table 1053  
     SYSKEYCOLUSE catalog table 1069  
     SYSKEYS catalog table 1070

COLSTATUS column of SYSCOLUMNS catalog table 1036

COLTYPE column  
     SYSCOLUMNS\_HIST catalog table 1037

COLTYPE column of SYSCOLUMNS catalog table 1032

column  
     controlling changes 9  
     derived  
         CREATE VIEW statement 713  
         functions 155  
         INSERT statement 833  
         null value 351  
         string comparison 73

**column (*continued*)**  
 derived (*continued*)
 

- UPDATE statement 931

**description** 4  
**name**

- ambiguous reference 96
- correlated reference 97
- in a result 351
- undefined reference 96

**retrieving**

- catalog information 1146
- comments 1151

**rules** 365  
**COLUMN clause**

- COMMENT statement 496
- LABEL ON statement 839

**COLVALUE column**

- SYSCOLDIST catalog table
  - description 1028
- SYSCOLDIST\_HIST catalog table 1029
- SYSCOLDISTSTATS catalog table
  - description 1030

**COMMA**

- column of SYSDBRM catalog table 1049
- column of SYSPACKAGE catalog table 1074
- option of precompiler 148

**comment**

- adding 493
- replacing 493
- SQL 152

**COMMENT statement**

- column name qualification 95
- description 493
- example 498
- examples 1151
- storing 1151

**commit**

- description 11

**COMMIT ON RETURN clause**

- ALTER PROCEDURE statement 436, 442
- CREATE PROCEDURE statement 632, 646

**COMMIT statement**

- description 499
- example 500

**COMMIT\_ON\_RETURN column**

- SYSPROCEDURES catalog table 1096
- SYSROUTINES catalog table 1103

**comparison**

- compatibility rules 64
- datetime values 75
- distinct type values 75
- numbers 72
- row ID values 75
- strings 73

**compatibility**

- data types 64
- rules 64

**COMPILE\_OPTS column**

- SYSROUTINES\_OPTS catalog table 1106

**compound statement**

- example 956

**compound statement (*continued*)**

- order of statements in 955
- SQL procedure 952

**COMPRESS**

- clause of ALTER TABLESPACE statement 474
- clause of CREATE TABLESPACE statement 694
- column of SYSTABLEPART catalog table 1122

**CONCAT**

- function 194
- operator 112

**concatenation**

- CONCAT function 194
- operator 112

**concurrency**

- application 11
- LOCK TABLE statement 840

**CONNECT**

- option of precompiler 146
- statement
  - differences, type 1 and type 2 501
  - type 1 504
  - type 2 510

**connected state** 19  
**connection**

- DB2 private protocol 16
- SQL 17

**connection exit routine**

- description 91

**connection state**

- application process 17, 507
- CONNECT (Type 1) statement 507
- SET CONNECTION statement 904
- SQL 17

**constant**

- character string 80
- datetime 81
- decimal 80
- floating-point 80
- graphic string 81
- hexadecimal 80
- integer 79

**CONSTNAME column**

- SYSKEYCOLUSE catalog table 1069
- SYTABCONST catalog table 1120

**constraint**

- description 6
- unique 6

**CONSTRAINT clause**

- ALTER TABLE statement 457, 458, 460
- CREATE TABLE statement 661, 668

**CONSTRAINT**

- clause of CREATE TABLE statement 670

**CONTAINS SQL clause**

- ALTER FUNCTION statement 400, 413
- ALTER PROCEDURE statement 433, 440
- CREATE FUNCTION statement 543, 563, 592
- CREATE PROCEDURE statement 628, 643

**CONTINUE**

- clause of WHENEVER statement 941

**CONTINUE handler**

- SQL procedure 955

CONTOKEN column  
   SYSCOLAUTH catalog table 1027  
   SYSPACKAGE catalog table 1073  
   SYSPACKSTMT catalog table 1081  
   SYSPKSYSTEM catalog table 1086  
   SYSROUTINEAUTH catalog table 1099  
   SYSTABAUTH catalog table 1119

control character 32

conversion of numbers  
   errors 901  
   precision 67  
   scale 67

CONVERT TO clause  
   ALTER INDEX statement 414

CONVLIMIT column of LUMODES catalog table  
   description 1019

COPY  
   clause of ALTER INDEX statement 417  
   clause of CREATE INDEX statement 613  
   column of SYSINDEXES catalog table 1056

COPY privilege  
   GRANT statement 816  
   REVOKE statement 881

COPYAUTH column of SYSPACKAUTH catalog table 1078

COPYLRSN column of SYSINDEXES catalog table 1056

COPYPAGESF column of SYSCOPY catalog table 1043

correlated reference  
   correlation name  
     defining 95  
     FROM clause of subselect 352  
     naming convention 35  
     qualifying a column name 95  
   description 97  
   HAVING clause 359  
   WHERE clause 358

COS function 195

COSH function 196

COUNT function 163

COUNT\_BIG function 164

CPAGESF column of SYSCOPY catalog table 1043

CREATE ALIAS statement  
   description 514  
   example 515

CREATE AUXILIARY TABLE statement  
   description 516  
   example 518

CREATE DATABASE statement  
   description 519  
   example 521

CREATE DISTINCT TYPE statement  
   description 522  
   example 528

CREATE FUNCTION (external scalar) statement  
   description 530  
   example 551

CREATE FUNCTION (external table) statement  
   description 553  
   example 570

CREATE FUNCTION (sourced) statement  
   description 571  
   example 584

CREATE FUNCTION (SQL scalar) statement  
   description 585  
   example 594

CREATE FUNCTION statement 529

CREATE GLOBAL TEMPORARY TABLE statement  
   description 595  
   example 599

CREATE IN privilege  
   binding a package 46  
   GRANT statement 806  
   REVOKE statement 870

CREATE INDEX statement  
   description 601  
   example 615

CREATE PROCEDURE (SQL) statement  
   description 636  
   example 647

CREATE PROCEDURE statement  
   assignment statement 946  
   description 617  
   example 634  
   SQL procedure body 944

CREATE STOGROUP statement  
   description 648  
   example 650

CREATE SYNONYM statement  
   description 651  
   example 651

CREATE TABLE statement  
   description 653  
   example 679

CREATE TABLESPACE statement  
   description 681  
   example 697

CREATE TRIGGER statement  
   description 699  
   example 708

CREATE VIEW statement  
   description 711  
   example 716  
   use 10

CREATEALIAS privilege  
   GRANT statement 821  
   REVOKE statement 886

CREATEALIASAUTH column of SYSUSERAUTH catalog table 1138

CREATEDBA privilege  
   GRANT statement 821  
   REVOKE statement 887

CREATEDBAAUTH column of SYSUSERAUTH catalog table 1137

CREATEDBC privilege  
   GRANT statement 822  
   REVOKE statement 887

CREATEDBCAAUTH column of SYSUSERAUTH catalog table 1137

CREATEDBY column  
   SYSDATABASE catalog table 1044

CREATEDBY column (*continued*)  
     SYSDATATYPES catalog table 1046  
     SYSINDEXES catalog table 1055  
     SYSROUTINES catalog table 1100  
     SYSSEQUENCES catalog table 1109  
     SYSTOGROUP catalog table 1114  
     SYSSYNONYMS catalog table 1117  
     SYSTABLES catalog table 1127  
     SYSTABLESPACE catalog table 1132  
     SYSTRIGGERS catalog table 1136

CREATEDTS column  
     SYSCOLUMNS catalog table 1036  
     SYSDATABASE catalog table 1044  
     SYSDATATYPES catalog table 1046  
     SYSINDEXES catalog table 1055  
     SYSJAROBJECTS catalog table 1067  
     SYSROUTINES catalog table 1103  
     SYSSEQUENCES catalog table 1109  
     SYSTOGROUP catalog table 1114  
     SYSSYNONYMS catalog table 1117  
     SYTABCONST catalog table 1120  
     SYTABLES catalog table 1128  
     SYSTABLESPACE catalog table 1132  
     SYSTRIGGERS catalog table 1136

CREATEIN privilege  
     GRANT statement 819  
     REVOKE statement 884

CREATEINAUTH column of SYSSCHEMAAUTH catalog table 1108

CREATESG privilege  
     GRANT statement 822  
     REVOKE statement 887

CREATESGAUTH column of SYSUSERAUTH catalog table 1137

CREATESTM column  
     SYSROUTINES\_SRC catalog table 1107

CREATETAB privilege  
     GRANT statement 807  
     REVOKE statement 871

CREATETABAUTH column of SYSDBAAUTH catalog table 1047

CREATETMTAB privilege  
     GRANT statement 822  
     REVOKE statement 887

CREATETMTBAUTH column of SYSUSERAUTH catalog table 1139

CREATETS privilege  
     GRANT statement 807  
     REVOKE statement 871

CREATETSAUTH column of SYSDBAAUTH catalog table 1047

CREATOR column  
     SYSCHECKS catalog table 1025  
     SYSCOLAUTH catalog table 1027  
     SYSDATABASE catalog table 1044  
     SYSFOREIGNKEYS catalog table 1053  
     SYSINDEXES catalog table 1054  
     SYSINDEXES\_HIST catalog table 1057  
     SYSPACKAGE catalog table 1073  
     SYSPLAN catalog table 1087  
     SYSRELS catalog table 1097

CREATOR column (*continued*)  
     SYSTOGROUP catalog table 1114  
     SYSSYNONYMS catalog table 1117  
     SYTABCONST catalog table 1120  
     SYTABLES catalog table 1126  
     SYTABLES\_HIST catalog table 1130  
     SYTABLESPACE catalog table 1131  
     SYSVIEWS catalog table 1141

CURRENCY function 1161

CURRENT  
     clause of RELEASE statement 859

CURRENT APPLICATION ENCODING SCHEME  
     special register 85

current connection state 18

CURRENT DATE special register 86

CURRENT DEGREE special register  
     assigning a value 907  
     description 86  
     setting 907

CURRENT LC\_CTYPE special register  
     description 87

CURRENT LOCALE LC\_CTYPE special register  
     assigning a value 909  
     description 87

CURRENT MEMBER special register  
     description 87

CURRENT OF clause  
     DELETE statement 744

CURRENT OPTIMIZATION HINT special register  
     assigning a value 911  
     description 87

CURRENT PACKAGESET special register  
     assigning a value 912  
     description 88  
     stored procedures 913

CURRENT PATH special register  
     assigning a value 921  
     description 88

CURRENT PRECISION special register  
     assigning a value 914  
     description 89

CURRENT RULES special register  
     assigning a value 915  
     description 90

current server  
     description 502  
     designating  
         CONNECT (Type 1) statement 504  
         CONNECT (Type 2) statement 510

CURRENT SERVER special register  
     description 91

CURRENT SQLID special register  
     assigning a value 916  
     description 91  
     initial value 44

CURRENT TIME special register  
     description 91

CURRENT TIMESTAMP special register  
     description 92

CURRENT TIMEZONE special register 92

CURRENTSERVER  
  column of SYSPLAN catalog table 1088  
  option of BIND PLAN subcommand 503  
  option of REBIND PLAN subcommand 503

cursor  
  closed state 844  
  closing  
    CLOSE statement 491  
    CONNECT (Type 1) statement 504  
    CONNECT (Type 2) statement 510  
    error in FETCH 800  
    error in UPDATE 933  
  INSENSITIVE 719, 848  
  open state 800  
  opening  
    errors 844  
    OPEN statement 842  
  SCROLL 720, 849  
  SENSITIVE 719, 849  
  STATIC 719  
  using  
    current row 800  
    DECLARE CURSOR statement 718  
    FETCH statement 793  
    positions 800

cursor-name clause  
  DECLARE CURSOR statement 719  
  FETCH statement 798

CYCLE column of SYSEQUENCES catalog  
table 1109

## D

DATA CAPTURE clause  
  ALTER TABLE statement 463  
  CREATE TABLE statement 674

data compression  
  COMPRESS clause  
    ALTER TABLESPACE statement 474  
    CREATE TABLESPACE statement 694

data type  
  built-in 48  
  casting between 62  
  character string 49  
  compatibility matrix 65  
  CREATE TABLE statement 658  
  datetime 56  
  distinct 60  
  graphic string 52  
  list of built-in types 48  
  name, unqualified 40  
  naming convention  
    built-in 35  
    distinct type 36  
  numeric 55  
  promotion 61  
  result column 351  
  results of an operation 77  
  row ID 60  
  string restrictions 54  
  unqualified name 40

database  
  altering  
    ALTER DATABASE statement 388  
  creating 519  
  default database 38  
  description 10  
  designing  
    using catalog 1145  
  dropping 766  
  DSNDB04 (default database) 38  
  implementing a design 1151  
  limits 965  
  naming convention 35  
  privileges  
    granting 807  
    revoking 871

DATABASE  
  clause of ALTER DATABASE statement 388  
  clause of DROP statement 766  
  clause of GRANT statement 808  
  clause of REVOKE statement 872

DATACAPTURE column of SYTABLES catalog  
table 1128

DATATYPEID column  
  DATATYPES catalog table 1046  
  SYSCOLUMNS catalog table 1036  
  SYSPARMS catalog table 1085  
  SYSSEQUENCES catalog table 1109

date  
  arithmetic 119  
  data type 56  
  duration 118  
  strings 58, 59

DATE  
  data type  
    CREATE TABLE statement 660  
    function 197

DATE FORMAT field of panel DSNTIP4 151

date routine  
  CHAR function 185

DATE  
  data type  
    description 56

datetime  
  arithmetic 118  
  constants 81  
  data types  
    description 56  
    string representation 57

EUR (IBM European standard) 58

format  
  setting through the CHAR function 185

ISO (International Standards Organization) 58

JIS (Japanese Industrial Standard) 58

LOCAL 58

string formats 58

USA 58

DAY function 198

day of week calculation 203

DAYNAME function 1163

DAYOFMONTH function 199

DAYOFWEEK function 200  
 DAYOFWEEK\_ISO function 201  
 DAYOFTIME function 202  
 DAYS function 203  
 DB2 private protocol access  
     authorization ID 46  
     description 14, 16  
     mixed environment 969  
 DB2 version identification, current server 506, 513  
 DBADM authority  
     GRANT statement 807  
     REVOKE statement 871  
 DBADMAUTH column of SYSDBAAUTH catalog  
     table 1047  
 DBCLOB  
     function 204  
 DBCLOB (double-byte character large object)  
     data type 52, 660  
     description 53  
     host variable 101  
     locator 102  
     restrictions 54  
 DBCS (double-byte character set)  
     ASCII 49  
     EBCDIC 32, 49  
     SQL ordinary identifier 31, 32  
     Unicode 49  
 DBCS data  
     description 23  
 DBCS\_CCSID column  
     SYSDATABASE catalog table 1044  
     SYSTABLESPACE catalog table 1133  
 DBCTRL authority  
     GRANT statement 807  
     REVOKE statement 871  
 DBCTRLAUTH column of SYSDBAAUTH catalog  
     table 1047  
 DBID  
     column of SYSCHECKS catalog table 1025  
     column of SYSDATABASE catalog table 1044  
     column of SYSINDEXES catalog table 1054  
     column of SYSTABLES catalog table 1126  
     column of SYSTABLESPACE catalog table 1131  
     column of SYSTRIGGERS catalog table 1136  
 DBINFO  
     clause of ALTER FUNCTION statement 404  
     clause of ALTER PROCEDURE statement 433  
     clause of CREATE FUNCTION statement 547, 567  
     clause of CREATE PROCEDURE statement 630  
     column of SYSROUTINES catalog table 1102  
 DBMAINT authority  
     GRANT statement 807  
     REVOKE statement 871  
 DBMAINTAUTH column of SYSDBAAUTH catalog  
     table 1047  
 DBMLIB column of SYSJAVAOPTS catalog table 1068  
 DBNAME column  
     SYSCOPY catalog table 1040  
     SYSINDEXES catalog table 1054  
     SYSLOBSTATS catalog table 1071  
     SYSLOBSTATS\_HIST catalog table 1072  
 DBNAME column (*continued*)  
     SYSTABAUTH catalog table 1118  
     SYSTABLEPART catalog table 1121  
     SYSTABLEPART\_HIST catalog table 1124  
     SYSTABLES catalog table 1126  
     SYSTABLES\_HIST catalog table 1130  
     SYSTABLESPACE catalog table 1131  
     SYTABSTATS catalog table 1134  
     SYTABSTATS\_HIST catalog table 1135  
 DBPROTOCOL column  
     SYSPACKAGE catalog table 1077  
     SYSPLAN catalog table 1089  
 DBRM (database request module)  
     description 14  
 DCLGEN subcommand of DSN  
     description 57  
 DCOLLID column of SYSPACKDEP catalog  
     table 1079  
 DCOLNAME column of SYSSEQUENCEDEP catalog  
     table 1110  
 DCONSTNAME column of SYSCONSTDEP catalog  
     table 1039  
 DCONTOKEN column of SYSPACKDEP catalog  
     table 1079  
 DCREATOR column  
     SYSSEQUENCEDEP catalog table 1110  
     SYSVIEWDEP catalog table 1140  
 deadlock  
     locks and uncommitted changes 11  
 DEC function 205  
 DEC15 precompiler option 114  
 DEC31  
     column of SYSDBRM catalog table 1049  
     column of SYSPACKAGE catalog table 1074  
     precompiler option 114  
 decimal  
     constants 80  
     numbers 55  
 DECIMAL  
     data type  
         CREATE TABLE statement 659  
         description 55  
     function  
         description 205  
 DECIMAL POINT IS field of panel DSNTIPF 148  
 decimal point precompiler option 148  
 DECLARE CURSOR statement  
     description 718  
     example 723  
 DECLARE GLOBAL TEMPORARY TABLE statement  
     description 725  
     example 734  
 DECLARE STATEMENT statement  
     description 735  
     example 735  
 DECLARE TABLE statement  
     description 736  
     example 737  
 DECLARE VARIABLE statement  
     description 739  
     example 741

DEFAULT  
  column of SYSCOLUMNS catalog table 1034  
default database (DSNDB04)  
  implicit specification 38  
DEFAULT REGISTERS clause  
  ALTER PROCEDURE statement 437, 443  
  CREATE PROCEDURE statement 632, 646  
DEFAULT SPECIAL REGISTERS clause  
  ALTER FUNCTION statement 405  
  CREATE FUNCTION statement 549, 569  
DEFAULTVALUE column of SYSCOLUMNS catalog  
  table 1036  
DEFER  
  clause of CREATE INDEX statement 612  
DEFERPREP column  
  SYSPACKAGE catalog table 1075  
  SYSPLAN catalog table 1088  
DEFERPREPARE column of SYSPACKAGE catalog  
  table 1077  
deferred embedded SQL 2  
define behavior for dynamic SQL statements 44  
DEFINE clause  
  CREATE TABLESPACE statement 610, 690  
DEGREE  
  column of SYSPACKAGE catalog table 1075  
  column of SYSPLAN catalog table 1088  
DEGREES function 207  
DELETE  
  clause of TRIGGER statement 701  
  statement  
    description 742  
    example 747  
DELETE privilege  
  GRANT statement 824  
  REVOKE statement 889  
delete rule 7  
delete rules 745  
delete-connected 7  
DELETEAUTH column of SYSTABAUTH catalog  
  table 1119  
DELETERULE column of SYSRELS catalog  
  table 1097  
deleting  
  rows from a table 742  
  SQL objects 763  
delimited identifier in SQL 33  
delimiter  
  SQL 33  
dependency  
  of objects on each other 772  
dependent  
  row 7  
  table 7  
DESC clause  
  CREATE INDEX statement 605  
  select-statement 371  
descendent table 7  
DESCRIBE CURSOR statement  
  description 756  
  example 757

DESCRIBE INPUT statement  
  prepared statement 758  
DESCRIBE PROCEDURE statement  
  description 760  
  example 762  
DESCRIBE statement  
  prepared statement 749  
  table or view 749  
  variables 750  
descriptor name 36  
DETERMINISTIC clause  
  ALTER FUNCTION statement 400, 412  
  ALTER PROCEDURE statement 433, 440  
  CREATE FUNCTION statement 542, 563, 590  
  CREATE PROCEDURE statement 630, 643  
DETERMINISTIC column of SYSROUTINES catalog  
  table 1101  
DEVTYPE column of SYSCOPY catalog table 1040  
DFSMSHsm (Data Facility Hierarchical Storage  
  Manager)  
  dropping an index or table space 772  
digit, description in DB2 31  
DIGITS function 208  
DISALLOW PARALLEL clause  
  ALTER FUNCTION statement 403  
  CREATE FUNCTION statement 546, 567  
DISCONNECT  
  column of SYSPLAN catalog table 1088  
DISPLAY privilege  
  GRANT statement 822  
  REVOKE statement 887  
DISPLAYAUTH column of SYSUSERAUTH catalog  
  table 1137  
DISPLAYDB privilege  
  GRANT statement 807  
  REVOKE statement 871  
DISPLAYDBAUTH column of SYSDBAUTH catalog  
  table 1047  
DISTINCT  
  clause of subselect 349  
  keyword  
    AVG function 162  
    column functions 161  
    COUNT function 163  
    COUNT\_BIG function 164  
    MAX function 166  
    MIN function 167  
    STDDEV function 168  
    STDDEV\_SAMP function 169  
    SUM function 170  
    VAR 171  
    VAR\_SAMP 172  
    VARIANCE function 171  
    VARIANCE\_SAMP function 172  
distinct type  
  assignment of values 71  
  casting 63  
  catalog information 1150  
  comparison of values 75  
  CREATE TABLE statement 661  
  creating 522

**distinct type** (*continued*)
   
 description 60
   
 dropping 766
   
 granting privileges 809
   
 name, unqualified 36, 40
   
 naming convention 36
   
 promotion 61
   
 revoking privileges 874
   
 unqualified name 40
   
**DISTINCT TYPE clause**
  
 COMMENT statement 496
   
 DROP statement 766
   
**distributed data**
  
 CONNECT statement 501
   
 CURRENT SERVER special register 91
   
 description 14
   
 RELEASE (connection) statement 859
   
 SET CONNECTION statement 904
   
**Distributed Relational Database Architecture (DRDA)** 15
   
**DLOCATION column** of SYSPACKDEP catalog table 1079
   
**DNAME column**

- SYSPACKDEP catalog table 1079
- SYSPLANDEP catalog table 1092
- SYSSEQUENCEDEP catalog table 1110
- SYSVIEWDEP catalog table 1140

  
**dormant connection state** 18
   
**DOUBLE data type**

- CREATE TABLE statement 659
- description 55

  
**DOUBLE function** 209
   
**DOUBLE PRECISION data type**

- CREATE TABLE statement 659
- description 55

  
**double precision floating-point number** 55
   
**DOUBLE\_PRECISION function** 209
   
**double-byte character**

- LABEL ON statement 839
- truncated during assignment 69

  
**double-byte character large object (DBCLOB)** 53
   
**DOWNER column** of SYSPACKDEP catalog table 1079
   
**DRDA (Distributed Relational Database Architecture)** 15
   
**DRDA access**

- authorization ID 46
- CONNECT (Type 1) statement 504
- CONNECT (Type 2) statement 510
- description 14, 15
- mixed environment 969
- restricted function 16

  
**DROP FOREIGN KEY clause**

- ALTER TABLE statement 462

  
**DROP PRIMARY KEY clause**

- ALTER TABLE statement 462

  
**DROP privilege**

- GRANT statement 807
- REVOKE statement 872

  
**DROP statement**

- description 763

  
**DROP statement** (*continued*)
   
 example 774
   
**DROP UNIQUE clause**

- ALTER TABLE statement 462

  
**DROPAUTH column** of SYSDBAAUTH catalog table 1048
   
**DROPIN privilege**

- GRANT statement 819
- REVOKE statement 884

  
**DROPINAUTH column** of SYSSCHEMAAUTH catalog table 1108
   
**DSN\_FUNCTION\_TABLE table**

- EXPLAIN statement 782

  
**DSN\_STATEMNT\_TABLE table**

- EXPLAIN statement 782

  
**DSNAME**

- column of SYSCOPY catalog table 1040

  
**DSNUM column**

- SYSCOPY catalog table 1040
- SYSINDEXPART catalog table 1059
- SYSINDEXPART\_HIST catalog table 1061
- SYSTABLEPART catalog table 1123
- SYSTABLEPART\_HIST catalog table 1125

  
**DSSIZE**

- clause of CREATE TABLESPACE statement 691
- column of SYSTABLESPACE catalog table 1133

  
**DSVOLSER column** of SYSCOPY catalog table 1041
   
**DTBCREATOR column** of SYSCONSTDEP catalog table 1039
   
**DTBNAME column** of SYSCONSTDEP catalog table 1039
   
**DTYPE column**

- SYSCONSTDEP catalog table 1039
- SYSPACKDEP catalog table 1079
- SYSVIEWDEP catalog table 1140

  
**duplicate rows, UNION clause** 365
   
**duration**

- date 118
- labeled 117
- time 118
- timestamp 118

  
**DYNAMIC RESULT SET clause**

- ALTER PROCEDURE statement 430, 440
- CREATE PROCEDURE statement 624, 642

  
**dynamic SQL**

- description 2, 380
- DYNAMICRULES bind option 44
- EXECUTE IMMEDIATE statement 779
- EXECUTE statement 776
- execution 382
- INTO clause
  - DESCRIBE statement 749
  - PREPARE statement 847
- invocation of SELECT statement 383
- preparation 382
- SQLDA 986
- statements allowed 969

  
**DYNAMICRULES**

- bind option 39
- column of SYSPACKAGE catalog table 1076
- column of SYSPLAN catalog table 1089

DYNAMICRULES (*continued*)  
dynamic SQL authorization 43  
unqualified names 39  
DYNAMICRULES behavior 44

## E

EBCDIC  
definition 21  
effect on DBCS characters 49  
EBCDIC CODED CHAR SET field of panel  
DSNTIPF 150  
edit routine  
named in CREATE TABLE statement 673  
specified by EDITPROC option 673  
EDITPROC clause  
CREATE TABLE statement 673  
EDPROC column of SYSTABLES catalog table 1126  
empty table, description 4  
ENABLE  
column of SYSPKSYSTEM catalog table 1086  
column of SYSPLSYSTEM catalog table 1093  
encoding scheme 21  
ENCODING\_CCSID column  
SYSPACKAGE catalog table 1077  
SYSPLAN catalog table 1089  
ENCODING\_SCHEME column  
SYSDATABASE catalog table 1044  
SYSDATATYPES catalog table 1046  
SYSPARMS catalog table 1085  
SYSTABLES catalog table 1129  
SYSTABLESPACE catalog table 1133  
ENCRYPTIONPSWDS column of LUNAMES catalog  
table 1021  
END DECLARE SECTION statement  
description 775  
example 775  
EPOCH column of SYTABLEPART catalog  
table 1123  
ERASE clause  
ALTER INDEX statement 420  
ALTER TABLESPACE statement 474  
CREATE INDEX statement 607  
CREATE TABLESPACE statement 687  
ERASERULE column  
SYSINDEXES catalog table 1055  
SYSTABLESPACE catalog table 1131  
error  
arithmetic expression 901  
closes cursor 844  
during FETCH 800  
during update 933  
numeric conversion 901  
signaling 927  
ERRORBYTE column of SYSSTRINGS catalog  
table 1115  
ESCAPE clause  
LIKE predicate 141  
evaluation order 123  
EXCLUSIVE  
option of LOCK TABLE statement 841  
exclusive dependence 867  
executable statement 380, 381  
EXECUTE IMMEDIATE statement  
description 779  
example 780  
EXECUTE privilege  
GRANT statement 812, 816, 818  
REVOKE statement 877, 881, 883  
EXECUTE statement  
description 776  
EXECUTEAUTH column  
SYSPACKAUTH catalog table 1078  
SYSPLANAUTH catalog table 1091  
SYSROUTINEAUTH catalog table 1099  
EXISTS predicate 134  
EXIT handler  
SQL procedure 955  
exit routine  
named in ALTER TABLE statement 462  
named in CREATE TABLE statement 667  
EXITPARM column of SYSFIELDS catalog table 1052  
EXITPARML column of SYSFIELDS catalog  
table 1052  
EXP function 210  
EXPLAIN  
column of SYSPACKAGE catalog table 1074  
statement  
description 781  
example 792  
EXPLAINABLE column  
SYSPACKSTMT catalog table 1083  
SYSSTMT catalog table 1113  
explainable statement  
description 781  
EXPLAIN statement 782  
using bind or rebind 783  
EXPLAN column of SYSPLAN catalog table 1088  
exposed name 98  
EXPREDICATE column of SYSPLAN catalog  
table 1088  
expression  
arithmetic operators 114  
CASE 124  
CAST specification 126  
concatenation operator 112  
datetime operands 117  
decimal operands 114  
floating-point operands 117  
integer operands 114  
precedence of operation 123  
subselect statement 350  
without operators 111  
EXTENTS column  
SYSINDEXPART catalog table 1059  
SYSINDEXPART\_HIST catalog table 1061  
SYTABLEPART catalog table 1123  
SYTABLEPART\_HIST catalog table 1124  
EXTERNAL ACTION clause  
ALTER FUNCTION statement 401, 412  
CREATE FUNCTION statement 543, 564, 591

EXTERNAL clause  
ALTER FUNCTION statement 397  
ALTER PROCEDURE statement 430  
CREATE FUNCTION statement 539, 561  
CREATE PROCEDURE statement 625, 642  
EXTERNAL\_ACTION column of SYSROUTINES catalog table 1101  
EXTERNAL\_NAME column of SYSROUTINES catalog table 1105  
EXTERNAL\_SECURITY column  
  SYSPROCEDURES catalog table 1095  
  SYSROUTINES catalog table 1103  
external-java-routine-name clause  
ALTER FUNCTION statement 397  
ALTER PROCEDURE statement 430  
CREATE FUNCTION statement 540  
CREATE PROCEDURE statement 626

## F

FARINDREF column  
  SYSTABLEPART catalog table 1121  
  SYSTABLEPART\_HIST catalog table 1124  
FAROFFPOSF column  
  SYSINDEXPART catalog table 1059  
  SYSINDEXPART\_HIST catalog table 1061  
FENCED  
  clause of CREATE FUNCTION statement 543, 563  
  clause of CREATE PROCEDURE statement 630, 643  
  column of SYSROUTINES catalog table 1102  
FETCH FIRST clause  
  select-statement 374  
FETCH FIRST n ROWS ONLY clause  
  SELECT INTO statement 902  
FETCH statement  
  description 793  
  example 801  
field description 457  
field procedure  
  comparisons 73  
  named in ALTER TABLE statement 457  
  named in CREATE TABLE statement 667  
FIELDPROC clause  
  ALTER TABLE statement 457  
  CREATE TABLE statement 667  
FILESEQNO column of SYSCOPY catalog table 1040  
FINAL CALL clause  
  ALTER FUNCTION statement 403, 545  
  CREATE FUNCTION statement 565  
FINAL\_CALL column of SYSROUTINES catalog table 1101  
FIRSTKEYCARD column  
  SYSINDEXSTATS catalog table 1062  
FIRSTKEYCARDF column  
  SYSINDEXES catalog table 1055  
  SYSINDEXES\_HIST catalog table 1057  
  SYSINDEXSTATS catalog table 1062  
  SYSINDEXSTATS\_HIST catalog table 1063  
FLDPROC column  
  SYSCOLUMNS catalog table 1035

FLDPROC column (*continued*)  
  SYSFIELDS catalog table 1052  
FLDTYPE column of SYSFIELDS catalog table 1052  
FLOAT  
  data type  
    CREATE TABLE statement 659  
    description 55  
    function 209  
floating-point  
  constants 80  
  double precision number 55  
  single precision number 55  
FLOOR function 212  
FOR  
  clause of CREATE ALIAS statement 515  
  clause of CREATE DISTINCT TYPE statement 524  
  clause of CREATE SYNONYM statement 651  
  clause of CREATE TABLE statement 659  
  clause of EXPLAIN statement 782  
FOR EACH ROW clause of TRIGGER statement 703  
FOR EACH STATEMENT clause of TRIGGER statement 703  
FOR FETCH ONLY clause 371  
FOR READ ONLY clause 371  
FOR RESULT SET clause of ALLOCATE CURSOR statement 386  
FOR UPDATE clause  
  NOFOR precompiler option 153  
  select-statement 372  
foreign key 6  
FOREIGN KEY clause  
  ALTER TABLE statement 458  
  CREATE TABLE statement 669  
FOREIGNKEY column of SYSCOLUMNS catalog table 1035  
Fortran application program  
  host variable 100  
  INCLUDE SQLCA 984  
  varying-length string 52  
FREE LOCATOR statement  
  description 802  
  example 802  
free space  
  index 609  
  table space 471  
FREEPAGE  
  clause of ALTER INDEX statement  
    description 422  
  clause of ALTER TABLESPACE statement  
    description 471  
  clause of CREATE INDEX statement  
    description 609  
  clause of CREATE TABLESPACE statement  
    description 688  
column of SYSINDEXPART catalog table 1058  
column of SYSTABLEPART catalog table 1122  
FREESPACE column  
  SYSLOBSTATS catalog table 1071  
  SYSLOBSTATS\_HIST catalog table 1072  
FREQUENCYF column  
  SYSCOLDIST catalog table 1028

FREQUENCYF column (*continued*)  
 SYSCOLDIST\_HIST catalog table 1029  
 SYSCOLDISTSTATS catalog table 1030

FROM clause  
 DELETE statement 743  
 PREPARE statement 851  
 REVOKE statement 866  
 subselect 352

FULL OUTER JOIN  
 description 356  
 example 362  
 FROM clause of subselect 356

FULLKEYCARD column of SYSINDEXSTATS catalog table 1062

FULLKEYCARDF column  
 SYSINDEXES catalog table 1055  
 SYSINDEXES\_HIST catalog table 1057  
 SYSINDEXSTATS catalog table 1062  
 SYSINDEXSTATS\_HIST catalog table 1063

fullselect  
 CREATE VIEW statement 713  
 description 365  
 example 367  
 INSERT statement 835

function  
 built-in 104, 155  
 cast 105  
 column  
 AVG 162  
 column name 95  
 COUNT 163  
 COUNT\_BIG 164  
 description 161  
 example 161  
 MAX 166  
 MIN 167  
 STDDEV 168  
 STDDEV\_SAMP 169  
 SUM 170  
 VAR 171  
 VAR\_SAMP 172  
 VARIANCE 171  
 VARIANCE\_SAMP 172  
 description 104  
 maximum number in select 966  
 name, unqualified 40  
 nesting 173  
 scalar  
 ABS 174  
 ACOS 175  
 ADD\_MONTHS 176  
 ASIN 178  
 ATAN 179  
 ATAN2 181  
 ATANH 180  
 BLOB 182  
 CCSID\_ENCODING 183  
 CEIL or CEILING 184  
 CHAR 185  
 CLOB 191  
 COALESCE 192

function (*continued*)  
 scalar (*continued*)  
 CONCAT 194  
 COS 195  
 COSH 196  
 DATE 197  
 DAY 198  
 DAYOFMONTH 199  
 DAYOFWEEK 200  
 DAYOFWEEK\_ISO 201  
 DAYOFYEAR 202  
 DAYS 203  
 DBCLOB 204  
 DECIMAL or DEC 205  
 DEGREES 207  
 description 173  
 DIGITS 208  
 DOUBLE or DOUBLE\_PRECISION 209  
 example 173  
 EXP 210  
 FLOAT 209  
 FLOOR 212  
 GENERATE\_UNIQUE 213  
 GRAPHIC 215  
 HEX 218  
 HOUR 219  
 IDENTITY\_VAL\_LOCAL 220  
 IFNULL 224  
 INSERT 225  
 INTEGER or INT 228  
 JULIAN\_DAY 229  
 LAST\_DAY 230  
 LCASE 231  
 LEFT 232  
 LENGTH 234  
 LN 235  
 LOCATE 236  
 LOG 235  
 LOG10 238  
 LOWER 231  
 LTRIM 239  
 MAX 240  
 MICROSECOND 241  
 MIDNIGHT\_SECONDS 242  
 MIN 243  
 MINUTE 244  
 MOD 245  
 MONTH 247  
 MQPUBLISH 248  
 MQPUBLISHXML 250  
 MQREAD 252  
 MQREADCLOB 254  
 MQREAXML 256  
 MQRECEIVE 257  
 MQRECEIVECLOB 259  
 MQRECEIVEXML 261  
 MQSEND 263  
 MQSENDXML 265  
 MQSENDXMLFILE 267  
 MQSENDXMLFILECLOB 269  
 MQSUBSCRIBE 271

**function** (*continued*)  
 scalar (*continued*)  
 MQUNSUBSCRIBE 273  
 MULTIPLY\_ALT 275  
 NEXT\_DAY 276  
 NULLIF 277  
 POSSTR 278  
 POWER 280  
 QUARTER 281  
 RADIANS 282  
 RAISE\_ERROR 283  
 RAND 284  
 REAL 285  
 REPEAT 286  
 REPLACE 288  
 RIGHT 290  
 ROUND 292  
 ROUND\_TIMESTAMP 294  
 ROWID 297  
 RTRIM 298  
 SECOND 299  
 SIGN 300  
 SIN 301  
 SINH 302  
 SMALLINT 303  
 SOAPHTTPC and SOAPHTTPV 304  
 SPACE 305  
 SQRT 306  
 STRIP 307  
 SUBSTR 309  
 TAN 312  
 TANH 313  
 TIME 314  
 TIMESTAMP 315  
 TIMESTAMP\_FORMAT 316  
 TO\_CHAR 327  
 TO\_DATE 316  
 TRANSLATE 317  
 TRUNC\_TIMESTAMP 321  
 TRUNCATE 320  
 UCASE 322  
 UPPER 322  
 VARCHAR 323  
 VARCHAR\_FORMAT 327  
 VARGRAPHIC 328  
 WEEK 331  
 WEEK\_ISO 332  
 YEAR 333  
 table 155  
 description 334  
 MQREADALL 335  
 MQREADALLCLOB 337  
 MQREADALLXML 339  
 MQRECEIVEALL 341  
 MQRECEIVEALLCLOB 343  
 MQRECEIVEALLXML 345  
 types 104  
 unqualified name 40  
 user-defined 105, 155  
**FUNCTION clause**  
 COMMENT statement 496

**FUNCTION clause** (*continued*)  
 DROP statement 766  
**function resolution** 107  
**function table** 781  
**FUNCTION\_TYPE column**  
 SYSROUTINES catalog table 1100  
**function, built-in**  
 invocation 107  
 name, unqualified 40  
 resolution 107  
 scalar  
 VALUE 192  
 unqualified name 40  
**FUNCTIONTS column**  
 SYSPACKAGE catalog table 1077  
 SYSPLAN catalog table 1089

**G**

**GBPCACHE clause**  
 ALTER INDEX statement 422  
 ALTER TABLESPACE statement 475  
 CREATE INDEX statement 609  
 CREATE TABLESPACE statement 689  
**GBPCACHE column**  
 SYSINDEXPART catalog table 1059  
 SYSTABLEPART catalog table 1123  
**GENERATE\_UNIQUE function** 213  
**GENERATED clause**  
 ALTER TABLE statement 454  
 CREATE TABLE statement 664  
 DECLARE GLOBAL TEMPORARY TABLE statement 728  
**GENERIC column of LUNAMES catalog table** 1021  
**GET DIAGNOSTICS statement**  
 example 957  
 SQL procedure 957  
**GMT (Greenwich Mean Time)** 84  
**GO TO clause of WHENEVER statement** 941  
**GOTO statement**  
 example 958  
 SQL procedure 958  
**GRANT statement**  
 collection privileges 806  
 database privileges 807  
 description 803  
 function privileges 811  
 package privileges 816  
 plan privileges 818  
 procedure privileges 811  
 schema privileges 819  
 system privileges 821  
 table privileges 824  
 USAGE privilege 809  
 use privileges 827  
 view privileges 824  
**GRANTEDTS column**  
 SYSCOLAUTH catalog table 1027  
 SYSDBAUTH catalog table 1048  
 SYSPLANAUTH catalog table 1091  
 SYSRESAUTH catalog table 1098

GRANTEDTS column (*continued*)  
   SYSROUTINEAUTH catalog table 1099  
   SYSSCHEMAAUTH catalog table 1108  
   SYSTABAUTH catalog table 1119  
   SYSUSERAUTH catalog table 1139

GRANTEE column  
   SYSCOLAUTH catalog table 1027  
   SYSDBAUTH catalog table 1047  
   SYSPACKAUTH catalog table 1078  
   SYSPLANAUTH catalog table 1091  
   SYSRESAUTH catalog table 1098  
   SYSROUTINEAUTH catalog table 1099  
   SYSSCHEMAAUTH catalog table 1108  
   SYSTABAUTH catalog table 1118  
   SYSUSERAUTH catalog table 1137

GRANTEETYPE column  
   SYSCOLAUTH catalog table 1027  
   SYSPACKAUTH catalog table 1078  
   SYSROUTINEAUTH catalog table 1099  
   SYSTABAUTH catalog table 1118

GRANTOR column  
   SYSCOLAUTH catalog table 1027  
   SYSDBAUTH catalog table 1047  
   SYSPACKAUTH catalog table 1078  
   SYSPLANAUTH catalog table 1091  
   SYSRESAUTH catalog table 1098  
   SYSROUTINEAUTH catalog table 1099  
   SYSSCHEMAAUTH catalog table 1108  
   SYSTABAUTH catalog table 1118  
   SYSUSERAUTH catalog table 1137

GRANULARITY column of SYSTRIGGERS catalog  
   table 1136

GRAPHIC  
   data type  
     CREATE TABLE statement 660  
     description 52  
     function 215  
     option of precompiler 50, 150

graphic string  
   constants 81  
   description 52

Greenwich Mean Time (GMT) 84

GROUP BY clause  
   cannot join view 715  
   subselect  
     description 359  
     results 350

GROUP\_MEMBER column  
   SYSCOPY catalog table 1042  
   SYSDATABASE catalog table 1044  
   SYSPACKAGE catalog table 1075  
   SYSPLAN catalog table 1088

grouping column 359

**H**

handler  
   SQL procedure 955

handling errors  
   SQL procedure 955

HAVING clause of subselect  
   description 359  
   results 350

held connection state 18

HEX function 218

hexadecimal constant 80

HIGH2KEY column  
   SYSCOLSTATS catalog table  
     description 1031  
   SYSCOLUMNS catalog table  
     description 1033  
   SYSCOLUMNS\_HIST catalog table 1038

HIGHDSNUM column of SYSCOPY catalog table 1042

HIGHKEY column of SYSCOLSTATS catalog  
   table 1031

HOLD LOCATOR statement  
   description 829  
   example 829

host identifier 34

host structure  
   description 103

host variable  
   colon 101  
   description 100  
   EXECUTE IMMEDIATE statement 779  
   EXPLAIN statement 782  
   FETCH statement 798  
   input 100  
   naming convention 36  
   output 100  
   PREPARE statement 851  
   SELECT INTO statement 900  
   substitution for parameter markers 776

HOSTLANG column  
   SYSDBRM catalog table 1049  
   SYSPACKAGE catalog table 1074

HOUR function 219

HPJCOMPILE\_OPTS column  
   SYSJAVAOPTS catalog table 1068

**I**

I/O processing  
   CURRENT DEGREE special register 86

IBM SQL xv

IBMREQD column  
   IPNAMES catalog table 1016  
   LOCATIONS catalog table 1017  
   LULIST catalog table 1018  
   LUMODES catalog table 1019  
   LUNAMES catalog table 1021  
   MODESELECT catalog table 1022  
   release dependency indicators 1005  
   SYSAUXRELS catalog table 1023  
   SYSCHECKDEP catalog table 1024  
   SYSCHECKS catalog table 1025  
   SYSCHECKS2 catalog table 1026  
   SYSCOLAUTH catalog table 1027  
   SYSCOLDIST catalog table 1028  
   SYSCOLDIST\_HIST catalog table 1029  
   SYSCOLDISTSTATS catalog table 1030

IBMREQD column (*continued*)

- SYSCOLSTATS catalog table 1031
- SYSCOLUMNS catalog table 1033
- SYSCOLUMNS\_HIST catalog table 1038
- SYSCONSTDEP catalog table 1039
- SYSCOPY catalog table 1040
- SYSDATABASE catalog table 1044
- SYSDATATYPES catalog table 1046
- SYSDBAUTH catalog table 1048
- SYSDBRM catalog table 1049
- SYSDUMMY1 catalog table 1051
- SYSFIELDS catalog table 1052
- SYSFOREIGNKEYS catalog table 1053
- SYSENDEXES catalog table 1055
- SYSENDEXES\_HIST catalog table 1057
- SYSENDEXPART catalog table 1058
- SYSENDEXPART\_HIST catalog table 1061
- SYSENDEXSTATS catalog table 1062
- SYSENDEXSTATS\_HIST catalog table 1063
- SYSJARCONTENTS catalog table 1065
- SYSJAROBJECTS catalog table 1067
- SYSJAVAOPTS catalog table 1068
- SYSEKEYCOLUSE catalog table 1069
- SYSEKEYS catalog table 1070
- SYSLOBSTATS catalog table 1071
- SYSLOBSTATS\_HIST catalog table 1072
- SYSPACKAGE catalog table 1075
- SYSPACKAUTH catalog table 1078
- SYSPACKDEP catalog table 1079
- SYSPACKLIST catalog table 1080
- SYSPACKSTMT catalog table 1081
- SYSPARMS catalog table 1085
- SYSPKSYSTEM catalog table 1086
- SYSPLAN catalog table 1087
- SYSPLANAUTH catalog table 1091
- SYSPLANDEP catalog table 1092
- SYSPLSYSTEM catalog table 1093
- SYSPROCEDURES catalog table 1095
- SYSPRELS catalog table 1097
- SYSPRESAUTH catalog table 1098
- SYSPROUTINEAUTH catalog table 1099
- SYSPROUTINES catalog table 1103
- SYSPROUTINES\_OPTS catalog table 1106
- SYSPROUTINES\_SRC catalog table 1107
- SYSSCHEMAAUTH catalog table 1108
- SYSSSEQUENCEDEP catalog table 1110
- SYSSSEQUENCES catalog table 1109
- SYSSSTM catalog table 1111
- SYSTOGROUP catalog table 1114
- SYSSSTRINGS catalog table 1115
- SYSSSYNONYMS catalog table 1117
- SYTABAUTH catalog table 1119
- SYTABCONST catalog table 1120
- SYTABLEPART catalog table 1122
- SYTABLEPART\_HIST catalog table 1125
- SYTABLES catalog table 1126
- SYTABLES\_HIST catalog table 1130
- SYTABLESPACE catalog table 1132
- SYTABSTATS catalog table 1134
- SYTABSTATS\_HIST catalog table 1135
- SYSTRIGGERS catalog table 1136

IBMREQD column (*continued*)

- SYSUSERAUTH catalog table 1138
- SYSVIEWDEP catalog table 1140
- SYSVIEWS catalog table 1141
- SYSVOLUMES catalog table 1142
- USERNAMES catalog table 1143
- ICBACKUP column of SYSCOPY catalog table 1041
- ICDATE column of SYSCOPY catalog table 1040
- ICTIME column of SYSCOPY catalog table 1041
- ICTYPE column of SYSCOPY catalog table 1040
- ICUNIT column of SYSCOPY catalog table 1041
- identifier in SQL
  - delimited 33
  - long 33
  - ordinary 32
- identity column
  - ALTER TABLE statement 455
  - CREATE TABLE statement 665
- IDENTITY\_VAL\_LOCAL function 220
- IF statement
  - example 960
  - SQL procedure 960
- IFNULL function 224
- IMAGCOPY privilege
  - GRANT statement 808
  - REVOKE statement 872
- IMAGCOPYAUTH column of SYSDBAUTH catalog table 1048
- IMMEDWRITE column
  - SYSPACKAGE catalog table 1077
  - SYSPPLAN catalog table 1089
- IMPLICIT column of SYSTABLESPACE catalog table 1131
- IN
  - clause of CREATE AUXILIARY TABLE statement 517
  - clause of CREATE PROCEDURE statement 622, 640
  - clause of CREATE TABLE statement 672
  - clause of CREATE TABLESPACE statement 684 predicate 77, 136
- IN EXCLUSIVE MODE clause of LOCK TABLE statement 841
- IN SHARE MODE clause of LOCK TABLE statement 840
- INCCSID column of SYSSTRINGS catalog table 1115
- INCLUDE statement
  - assembler declarations 983
  - description 830
  - example 831
  - SQLCA
    - C 983
    - COBOL 984
    - Fortran 984
  - SQLDA
    - assembler 998
    - C 998
    - C++ 998
    - COBOL 1002
    - PL/I 985, 1002

**INCLUDING COLUMN DEFAULTS** clause  
 DECLARE GLOBAL TEMPORARY TABLE statement 730  
**INCLUDING IDENTITY COLUMN ATTRIBUTES** clause  
 DECLARE GLOBAL TEMPORARY TABLE statement 729, 730  
**INCREMENT** column of SYSEQUENCES catalog table 1109  
**index**  
 altering  
     ALTER INDEX statement 414  
     catalog information about 1147, 1148  
     catalog table 1006  
     creating with CREATE INDEX statement 601  
     description 5  
     dropping 767  
     name, unqualified 39  
     naming convention 36  
     partitioning 611  
     primary 5  
     space  
         description 10  
     types  
         changing 414  
         primary 1148  
         unique 5  
     unqualified name 39  
**INDEX** clause  
 ALTER INDEX statement 414  
 CREATE INDEX statement 604  
 DROP statement 767  
**INDEX privilege**  
 GRANT statement 824  
 REVOKE statement 889  
**INDEX**  
 clause of COMMENT statement 497  
**INDEXAUTH** column of SYSTABAUTH catalog table 1119  
**INDEXBP**  
 clause of ALTER DATABASE statement 388  
 clause of CREATE DATABASE statement 520  
 column of SYSDATABASE catalog table 1045  
**INDEXSPACE** column of SYSINDEXES catalog table 1054  
**INDEXTYPE** column of SYSINDEXES catalog table 1055  
**indicator array** 103  
**indicator variable**  
 description 100  
 string expression 779  
**infix operators** 114  
**INHERIT SPECIAL REGISTERS** clause  
 ALTER FUNCTION statement 405  
 ALTER PROCEDURE statement 437, 442  
 CREATE FUNCTION statement 549, 569  
 CREATE PROCEDURE statement 632, 646  
**INNER JOIN**  
 description 356  
 example 361  
 FROM clause of subselect 356  
**INOUT** clause  
 CREATE PROCEDURE statement 622, 640  
**input host variable** 100  
**INSENSITIVE** clause  
 FETCH statement 794  
**INSENTITIVE** clause  
 DECLARE CURSOR statement 719  
**INSERT** clause of CREATE TRIGGER statement 701  
**INSERT** function 225  
**INSERT** privilege  
 GRANT statement 825  
 REVOKE statement 889  
**insert rule** 7, 835  
**INSERT** statement  
 description 832  
 example 837  
**INSERTAUTH** column of SYSTABAUTH catalog table 1119  
**inserting**  
 declaration in a program 830  
 rows in a table 832  
**INSTS\_PER\_INVOC** column of SYSROUTINES catalog table 1105  
**INT** function 228  
**INTEGER**  
 data type  
 CREATE TABLE statement 658  
 large 55  
 small 55  
 function 228  
**integer constants** 79  
**integrated catalog facility**  
 CREATE INDEX statement 608  
 identifier 35  
**interactive SQL** 3, 384  
**INTIAL\_INSTS** column of SYSROUTINES catalog table 1105  
**INTIAL\_IOS** column of SYSROUTINES catalog table 1105  
**INTO** clause  
 DESCRIBE CURSOR statement 756  
 DESCRIBE INPUT statement 758  
 DESCRIBE PROCEDURE statement 761  
 DESCRIBE statement 750  
 FETCH statement 798  
 INSERT statement 833  
 PREPARE statement 847  
 SELECT INTO statement 900  
 VALUES INTO statement 939  
**INTO DESCRIPTOR** clause  
 FETCH statement 798  
**invoke behavior for dynamic SQL statements** 44  
**IOS\_PER\_INVOC** column of SYSROUTINES catalog table 1104  
**IPADDR** column of IPNAMES catalog table 1016  
**IPREFIX** column  
 SYSINDEXPART catalog table 1059  
 SYSTABLEPART catalog table 1123  
**IS** clause  
 COMMENT statement 498  
 LABEL ON statement 839

ISOBJID column of SYSINDEXES catalog table 1054  
**ISOLATION**  
 column of SYSPACKAGE catalog table 1074  
 column of SYSPACKSTMT catalog table 1081  
 column of SYSPLAN catalog table 1087  
 column of SYSTMT catalog table 1111  
**isolation level**  
 control by SQL statement  
   DELETE statement 745  
   INSERT statement 835  
   SELECT INTO statement 901  
   select-statement 373  
**IXCREATOR** column  
   SYSINDEXPART catalog table 1058  
   SYSINDEXPART\_HIST catalog table 1061  
   SYSKEYS catalog table 1070  
   SYSTABLEPART catalog table 1121  
**IXNAME** column  
   SYSINDEXPART catalog table 1058  
   SYSINDEXPART\_HIST catalog table 1061  
   SYSKEYS catalog table 1070  
   SYTABCONST catalog table 1120  
   SYSTABLEPART catalog table 1121  
**IXNAME** column of SYSRELS catalog table 1097  
**IXOWNER** column  
   SYSRELS catalog table 1097  
   SYTABCONST catalog table 1120

## J

**JAR** privileges  
 granting 809  
 revoking 874  
**JAR\_DATA** column  
   SYSJARDATA catalog table 1066  
   SYSJAROBJECTS catalog table 1067  
**JAR\_DATA\_ROWID** column  
   SYSJAROBJECTS catalog table 1067  
**JAR\_ID** column  
   SYSJARCONTENTS catalog table 1065  
   SYSJAROBJECTS catalog table 1067  
   SYSJAVAOPTS catalog table 1068  
   SYSROUTINES catalog table 1105  
**JARSCHEMA** column  
   SYSJARCONTENTS catalog table 1065  
   SYSJAROBJECTS catalog table 1067  
   SYSJAVAOPTS catalog table 1068  
   SYSROUTINES catalog table 1105  
**JAVA\_SIGNATURE** column  
   SYSROUTINES catalog table 1105  
**JDBC** 3  
**JOBNAME** column of SYSCOPY catalog table 1043  
**join operation**  
 example 361  
 FROM clause of subselect 358  
**FULL OUTER JOIN**  
 FROM clause of subselect 356  
**INNER JOIN**  
 FROM clause of subselect 356  
**joining tables** 356

join operation (*continued*)  
**LEFT OUTER JOIN**  
 FROM clause of subselect 356  
**RIGHT OUTER JOIN**  
 FROM clause of subselect 356  
 summary of results 358  
**JULIAN\_DAY** function 229

## K

Katakana character 32  
**KATAKANA** value for EBCDIC CODED CHAR SET 32  
**KEEPDYNAMIC** column  
   SYSPACKAGE catalog table 1077  
   SYSPLAN catalog table 1089  
**key**  
 composite  
   description 5  
 description 5  
 foreign  
   catalog information 1148  
   description 6  
 length  
   maximum 966  
   partitioning index 423, 611, 931  
 parent 6  
   catalog information 1148  
 primary  
   defining on a single column 661  
   description 5  
   unique 5  
**KEYCOLUMNS** column of SYSTABLES catalog  
 table 1127  
**KEYCOUNT** column of SYSINDEXSTATS catalog  
 table 1062  
**KEYCOUNTF** column  
   SYSINDEXSTATS catalog table 1062  
   SYSINDEXSTATS\_HIST catalog table 1063  
**KEYOBJID** column of SYSTABLES catalog table 1127  
**KEYSEQ** column of SYSCOLUMNS catalog  
 table 1035  
**keywords, reserved** 1153

## L

**LABEL**  
 column of SYSTABLES catalog table 1127  
**LABEL ON** statement  
 description 838  
 example 839  
**LABEL**  
 column of SYSCOLUMNS catalog table 1035  
**labeled duration** 117  
**LABELS**  
 USING clause of DESCRIBE statement 750  
 USING clause of PREPARE statement 848  
**LANGUAGE**  
 clause of ALTER FUNCTION statement 398  
 clause of ALTER PROCEDURE statement 431  
 clause of CREATE FUNCTION statement 541, 562

**LANGUAGE** (*continued*)  
 clause of CREATE PROCEDURE statement 627, 643  
 column of SYSPROCEDURES catalog table 1095  
**LANGUAGE** column  
 SYSROUTINES catalog table 1100  
**LANGUAGE SQL** clause  
 ALTER FUNCTION statement 411  
 CREATE FUNCTION statement 590  
**LARGE** clause  
 CREATE TABLESPACE statement 683  
**large object (LOB)**  
 description 53  
**LAST\_DAY** function 230  
**LCASE** function 231  
**LEAFDIST** column  
 SYSINDEXPART\_HIST catalog table 1061  
**LEAFDIST** column of SYSINDEXPART catalog table  
 description 1058  
**LEAFFAR** column  
 SYSINDEXPART catalog table 1060  
 SYSINDEXPART\_HIST catalog table 1061  
**LEAFNEAR** column  
 SYSINDEXPART catalog table 1059  
 SYSINDEXPART\_HIST catalog table 1061  
**LEAVE** statement  
 example 961  
 SQL procedure 961  
**LEFT** function 232  
**LEFT OUTER JOIN**  
 example 362  
 FROM clause of subselect 356  
**length** attribute of column 51  
**LENGTH** column  
 SYSCOLUMNS catalog table 1033  
 SYSCOLUMNS\_HIST catalog table 1037  
 SYSDATATYPES catalog table 1046  
 SYSFIELDS catalog table 1052  
 SYSPARMS catalog table 1085  
**LENGTH** function 234  
**LENGTH2** column  
 SYSCOLUMNS catalog table 1036  
 SYSCOLUMNS\_HIST catalog table 1038  
**letter**, description in DB2 31  
**LIKE** clause  
 CREATE GLOBAL TEMPORARY TABLE  
 statement 597  
 CREATE TABLE statement 671  
 DECLARE GLOBAL TEMPORARY TABLE  
 statement 728  
**LIKE** predicate 137  
**LIMITKEY** column  
 SYSINDEXPART catalog table 1058  
 SYSTABLEPART catalog table 1122  
**limits**, DB2 965  
**LINK\_OPTS** column  
 SYSROUTINES\_OPTS catalog table 1106  
**LINKAGE** column of SYSPROCEDURES catalog  
 table 1094  
**LINKNAME** column  
 IPNAMES catalog table 1016

**LINKNAME** column (*continued*)  
 LOCATIONS catalog table 1017  
 LULIST catalog table 1018  
 USERNAMES catalog table 1143  
**literal** 79  
**LN** function 235  
**LOAD** privilege  
 GRANT statement 808  
 REVOKE statement 872  
**LOADAUTH** column of SYSDBAUTH catalog  
 table 1048  
**LOADMOD** column  
 SYSPROCEDURES catalog table 1094  
**LOB** (large object)  
 clause of CREATE TABLESPACE statement 684  
 description 53  
 host variable 101  
 locator 54, 102  
 restrictions 54  
 retrieving catalog information 1149  
**LOBCOLUMNS** column of SYSROUTINES catalog  
 table 1103  
**local DB2** 14  
**locale**  
 CURRENT LOCALE LC\_CTYPE special register 87  
**LOCATE** function 236  
**location**  
 LOB 54  
**LOCATION**  
 column of LOCATIONS catalog table 1017  
 column of SYSPACKAGE catalog table 1073  
 column of SYSPACKAUTH catalog table 1078  
 column of SYSPACKLIST catalog table 1080  
 column of SYSPACKSTMT catalog table 1081  
 column of SYSPKSYSTEM catalog table 1086  
 column of SYSTABLES catalog table 1128  
**location identifier** 34  
**locator**  
 LOB 54, 102  
 result set 102  
**LOCATOR** column of SYSPARMS catalog table 1085  
**locator variable**  
 freeing 802  
 holding beyond a unit of work 829  
**lock**  
 ALTER TABLESPACE statement 469  
 CREATE TABLESPACE statement 692  
 description 11  
 during update 933  
 LOCK TABLE statement 840  
 object  
 table space (table) 840  
**LOCK TABLE** statement  
 description 840  
 example 841  
**LOCKMAX** clause  
 ALTER TABLESPACE statement  
 description 470  
 CREATE TABLESPACE statement  
 description 693

**LOCKMAX** column  
 SYSTABLESPACE catalog table 1132  
**LOCKPART**  
 clause of ALTER TABLESPACE statement 475  
 clause of CREATE TABLESPACE statement 695  
 column of SYSTABLESPACE catalog table 1133  
**LOCKRULE** column of SYSTABLESPACE catalog table 1131  
**LOCKSIZE** clause  
 ALTER TABLESPACE statement  
     description 469  
 CREATE TABLESPACE statement  
     description 692  
**LOG**  
 clause of ALTER TABLESPACE statement 476  
 clause of CREATE TABLESPACE statement 690  
 column of SYSTABLESPACE catalog table 1133  
     function 235  
**LOG10** function 238  
**logical operator** 145  
**long column string** 52  
**long strings**  
     restrictions on use 54  
     types 51  
**LONG VARCHAR** data type  
 CREATE TABLE statement 656  
     description 51  
**LONG VARGRAPHIC** data type  
 CREATE TABLE statement 656  
     description 52  
**LOOP** statement  
     example 962  
     SQL procedure 962  
**LOW2KEY** column  
 SYSCOLSTATS catalog table 1031  
 SYSCOLUMNS catalog table  
     description 1033  
 SYSCOLUMNS\_HIST catalog table 1038  
**LOWDSNUM** column of SYSCOPY catalog table 1042  
**LOWER** function 231  
 lowercase character folded to uppercase 32  
**LOWKEY** column of SYSCOLSTATS catalog table 1031  
**LTRIM** function 239  
**LUNAME**  
 column of LULIST catalog table 1018  
 column of LUMODES catalog table 1019  
 column of LUNAMES catalog table 1020  
 column of MODESELECT catalog table 1022  
 column of SYSPROCEDURES table 1094

**MAXROWS** (*continued*)  
 column of SYSTABLESPACE catalog table 1133  
**MAXVALUE** column of SYSSEQUENCES catalog table 1109  
**MBCS** data  
     description 23, 49  
**MEMBER CLUSTER** clause  
 CREATE TABLESPACE statement 691  
**message**  
 precompiler processing of DECLARE TABLE statement 737  
**METATYPE** column of SYSDATATYPES catalog table 1046  
**MICROSECOND** function 241  
**MIDNIGHT\_SECONDS** function 242  
**MIN**  
     column function 167  
     scalar function 243  
**MINUTE** function 244  
**MINVALUE** column of SYSSEQUENCES catalog table 1109  
**MIXED** column  
 SYSDBRM catalog table 1049  
 SYSPACKAGE catalog table 1074  
**mixed data**  
     convention xvi  
     description 49  
     in string assignments 69  
     LIKE predicate 140  
**MIXED DATA**  
     field of panel DSNTIPF 50, 150  
**MIXED\_CCSID** column  
 SYSDATABASE catalog table 1045  
 SYSTABLESPACE catalog table 1133  
**MOD** function 245  
**MODE** SQL clause of TRIGGER statement 704  
**MODENAME** column  
 LUMODES catalog table 1019  
 MODESELECT catalog table 1022  
**MODESELECT** column of LUNAMES catalog table 1021  
**MODIFIES SQL DATA** clause  
 ALTER FUNCTION statement 400  
 ALTER PROCEDURE statement 433, 440  
 CREATE FUNCTION statement 543  
 CREATE PROCEDURE statement 628, 643  
**MON1AUTH** column of SYSUSERAUTH catalog table 1138  
**MON2AUTH** column of SYSUSERAUTH catalog table 1138  
**MONITOR1** privilege  
 GRANT statement 822  
 REVOKE statement 887  
**MONITOR2** privilege  
 GRANT statement 822  
 REVOKE statement 887  
**MONTH** function 247  
**MONTHNAME** function 1164  
**MQPUBLISH** function 248  
**MQPUBLISHXML** function 250  
**MQREAD** function 252

## M

**MAX**  
 column function 166  
 scalar function 240  
**MAXASSIGNEDVAL** column of SYSSEQUENCES catalog table 1109  
**MAXROWS**  
 clause of ALTER TABLESPACE statement 476  
 clause of CREATE TABLESPACE statement 695

MQREADALL function 335  
MQREADALLCLOB function 337  
MQREADALLXML function 339  
MQREADCLOB function 254  
MQREADXML function 256  
MQRECEIVE function 257  
MQRECEIVEALL function 341  
MQRECEIVEALLCLOB function 343  
MQRECEIVEALLXML function 345  
MQRECEIVECLOB function 259  
MQRECEIVEXML function 261  
MQSEND function 263  
MQSENDXML function 265  
MQSENDXMLFILE function 267  
MQSENDXMLFILECLOB function 269  
MQSeries functions 105, 155  
MQSUBSCRIBE function 271  
MQUNSUBSCRIBE function 273  
MULTIPLY\_ALT function 275

## N

NACTIVE column  
  SYSTABLESPACE catalog table  
    description 1131  
  SYTABSTATS catalog table 1134  
NACTIVEF column  
  SYSTABLESPACE catalog table  
    description 1133  
NAME  
  column of SYSCOLDIST catalog table 1028  
  column of SYSCOLDISTSTATS catalog table 1030  
  column of SYSCOLSTATS catalog table 1031  
  column of SYSCOLUMNS catalog table 1032  
NAME clause  
  ALTER FUNCTION statement 397  
  ALTER PROCEDURE statement 430  
  CREATE FUNCTION statement 540  
  CREATE PROCEDURE statement 626, 642  
NAME column  
  SYSCOLDIST\_HIST catalog table 1029  
  SYSCOLUMNS\_HIST catalog table 1037  
  SYSDATABASE catalog table 1044  
  SYSDATATYPES catalog table 1046  
  SYSDBAUTH catalog table 1047  
  SYSDBRM catalog table 1049  
  SYSFIELDS catalog table 1052  
  SYSINDEXES catalog table 1054  
  SYSINDEXES\_HIST catalog table 1057  
  SYSINDEXSTATS catalog table 1062  
  SYSINDEXSTATS\_HIST catalog table 1063  
  SYSLOBSTATS catalog table 1071  
  SYSLOBSTATS\_HIST catalog table 1072  
  SYSPACKAGE catalog table 1073  
  SYSPACKAUTH catalog table 1078  
  SYSPACKLIST catalog table 1080  
  SYSPACKSTMT catalog table 1081  
  SYSPARMS catalog table 1084  
  SYSPKSYSTEM catalog table 1086  
  SYSPPLAN catalog table 1087  
  SYSPLANAUTH catalog table 1091

NAME column (*continued*)  
  SYSPLSYSTEM catalog table 1093  
  SYSRESAUTH catalog table 1098  
  SYSROUTINES catalog table 1100  
  SYSSEQUENCES catalog table 1109  
  SYSSTMT catalog table 1111  
  SYSSTOGROUP catalog table 1114  
  SYSSYNONYMS catalog table 1117  
  SYSTABLES catalog table 1126  
  SYSTABLES\_HIST catalog table 1130  
  SYSTABLESPACE catalog table 1131  
  SYTABSTATS catalog table 1134  
  SYTABSTATS\_HIST catalog table 1135  
  SYSTRIGGERS catalog table 1136  
  SYSVIEWS catalog table 1141  
names, prepared SQL statements 735  
NAMES  
  USING clause of DESCRIBE statement 750  
  USING clause of PREPARE statement 848  
naming convention  
  SQL 34  
NEARINDREF column  
  SYSTABLEPART catalog table 1121  
  SYSTABLEPART\_HIST catalog table 1124  
NEAROFFPOSF column  
  SYSINDEXPART catalog table 1059  
  SYSINDEXPART\_HIST catalog table 1061  
nested table expressions 353  
NEW AS clause of CREATE TRIGGER statement 702  
NEW TABLE AS clause of CREATE TRIGGER  
  statement 702  
NEW\_TABLE AS clause of CREATE TRIGGER  
  statement 700  
NEWAUTHID column of USERNAMES catalog  
  table 1143  
NEXT\_DAY function 276  
NLEAF column  
  SYSINDEXES catalog table  
    description 1054  
  SYSINDEXES\_HIST catalog table 1057  
  SYSINDEXSTATS catalog table 1062  
  SYSINDEXSTATS\_HIST catalog table 1063  
NLEVELS column  
  SYSINDEXES catalog table  
    description 1054  
  SYSINDEXES\_HIST catalog table 1057  
  SYSINDEXSTATS catalog table 1062  
  SYSINDEXSTATS\_HIST catalog table 1063  
NO ACTION delete rule  
  CREATE TABLE statement 670  
  description 7  
NO CASCADE BEFORE clause of CREATE TRIGGER  
  statement 701  
NO COLLID clause  
  ALTER FUNCTION statement 404  
  CREATE FUNCTION statement 547, 567  
NO DBINFO clause  
  ALTER FUNCTION statement 404  
  ALTER PROCEDURE statement 433  
  CREATE FUNCTION statement 547, 567  
  CREATE PROCEDURE statement 630, 644

NO EXTERNAL ACTION clause  
   ALTER FUNCTION statement 401, 412  
   CREATE FUNCTION statement 543, 564, 591  
 NO FINAL CALL clause  
   ALTER FUNCTION statement 403, 545  
   CREATE FUNCTION statement 565  
 NO SCRATCHPAD clause  
   ALTER FUNCTION statement 402  
   CREATE FUNCTION statement 544, 565  
 NO SQL clause  
   ALTER FUNCTION statement 400  
   ALTER PROCEDURE statement 433  
   CREATE FUNCTION statement 543, 563  
   CREATE PROCEDURE statement 628  
 NO WLM ENVIRONMENT clause  
   ALTER PROCEDURE statement 434, 441  
   CREATE PROCEDURE statement 629, 644  
 NOCOLLID clause  
   ALTER PROCEDURE statement 434, 440  
   CREATE PROCEDURE statement 630, 644  
 NOFOR option  
   precompiler 153  
 NOGRAPHIC option of precompiler 150  
 nonexecutable statement 380, 381  
 NOT DETERMINISTIC clause  
   ALTER FUNCTION statement 400, 412  
   ALTER PROCEDURE statement 433, 440  
   CREATE FUNCTION statement 542, 563, 590  
   CREATE PROCEDURE statement 630, 643  
 NOT FOUND clause of WHENEVER statement 941  
 NOT NULL CALL clause  
   ALTER FUNCTION statement 394  
   CREATE FUNCTION statement 533, 556  
 NOT NULL clause  
   ALTER TABLE statement 452  
   CREATE GLOBAL TEMPORARY TABLE  
     statement 597  
   CREATE TABLE statement  
     description 661  
   DECLARE GLOBAL TEMPORARY TABLE  
     statement 727  
 NOT VARIANT clause  
   ALTER FUNCTION statement 410  
   ALTER PROCEDURE statement 428, 439  
   CREATE FUNCTION statement 533, 556, 587  
   CREATE PROCEDURE statement 620, 639  
 notices, legal 1173  
 NPAGES column  
   SYSTABLES catalog table  
     description 1126  
   SYTABSTATS catalog table 1134  
   SYTABSTATS\_HIST catalog table 1135  
 NPAGESF column  
   SYSCOPY catalog table 1043  
   SYSTABLES catalog table 1129  
   SYSTABLES\_HIST catalog table 1130  
 NTABLES column of SYSTABLESPACE catalog  
   table 1131  
 NULL  
   CAST specification 127  
   predicate 144

NULL CALL clause  
   ALTER FUNCTION statement 394  
   CREATE FUNCTION statement 533, 556  
 null value  
   assigned to host variable 900  
   assignment 65  
   description 48  
   duplicate rows 349  
   grouping columns 359  
   result columns 351  
   specified by indicator variable 100  
 NULL\_CALL column of SYSROUTINES catalog  
   table 1101  
 NULLIF function 277  
 NULLS column  
   SYSCOLUMNS catalog table 1033  
   SYSCOLUMNS\_HIST catalog table 1038  
 numbers  
   data types  
     string representation 56  
 numbers in SQL 55  
 NUMCOLUMNS column  
   SYSCOLDIST catalog table 1028  
   SYSCOLDIST\_HIST catalog table 1029  
   SYSCOLDISTSTATS catalog table 1030  
 numeric  
   assignments 66  
   comparisons 72  
   conversion errors 901  
   data type 55  
 NUMERIC data type  
   CREATE TABLE statement 659  
   description 55  
 NUMPARTS  
   clause of CREATE TABLESPACE statement 692

## O

OBID  
   clause of CREATE TABLE statement 674  
   column of SYSCHECKS catalog table 1025  
   column of SYSINDEXES catalog table 1054  
   column of SYSTABLES catalog table 1126  
   column of SYSTABLESPACE catalog table 1131  
   column of SYSTRIGGERS catalog table 1136  
 object name, unqualified 39  
 object table 95  
 OBTYPE column of SYSRESAUTH catalog table 1098  
 ODBC (Open Database Connectivity) 3  
 OLD AS clause of TRIGGER statement 702  
 OLD TABLE AS clause of CREATE TRIGGER  
   statement 702  
 OLD\_TABLE AS clause of CREATE TRIGGER  
   statement 700  
 ON clause  
   CREATE INDEX statement 605  
   CREATE TRIGGER statement 702  
   joining tables 356  
 ON COMMIT ROWS clause  
   DECLARE GLOBAL TEMPORARY TABLE  
     statement 731

**ON DELETE clause**  
 ALTER TABLE statement 460  
 CREATE TABLE statement 670  
**ON ROLLBACK RETAIN CURSORS clause**  
 SAVEPOINT statement 897  
**ON ROLLBACK RETAIN LOCKS clause**  
 SAVEPOINT statement 898  
**ON TABLE clause**  
 GRANT statement 825  
 REVOKE statement 890  
**OPEN**  
 statement  
 description 842  
 example 845  
 open cursor 800  
 Open Database Connectivity (ODBC) 3  
**operands**  
 datetime 117  
 decimal 114  
 floating-point 117  
 integer 114  
**operation**  
 SQL  
 assignment 64  
 comparison 72  
 description 64  
**OPERATIVE column**  
 SYSPACKAGE catalog table 1073  
 SYSPLAN catalog table 1087  
**operator**  
 arithmetic 114  
**OPTHINT column**  
 SYSPACKAGE catalog table 1077  
 SYSPLAN catalog table 1089  
**OPTIMIZE FOR n ROWS clause** 372  
**OR truth table** 145  
**ORDER BY clause**  
 select-statement 370  
**ORDER column of SYSSEQUENCES catalog**  
 table 1109  
 order of evaluation, operators 123  
 order of statements in a compound statement 955  
**ORDERING column of SYSKEYS catalog table** 1070  
**ORDINAL column of SYSPARMS catalog table** 1085  
 ordinary identifier in SQL 32  
**ORGRATIO column**  
 SYSLOBSTATS catalog table 1071  
 SYSLOBSTATS\_HIST catalog table 1072  
**ORIGIN column of SYSROUTINES catalog table** 1100  
**OTYPE column of SYSCOPY catalog table** 1042  
 OUT clause of CREATE PROCEDURE statement 622, 640  
**OUTCCSID column of SYSSTRINGS catalog**  
 table 1115  
**outer join**  
 FULL OUTER JOIN  
 example 362  
 FROM clause of subselect 356  
 LEFT OUTER JOIN  
 example 362  
 FROM clause of subselect 356

outer join (*continued*)  
 RIGHT OUTER JOIN  
 example 363  
 FROM clause of subselect 356  
 output host variable 100  
**OVERRIDING USER VALUE**  
 clause of INSERT statement 834  
**OWNER**  
 column of SYSDATATYPES catalog table 1046  
 column of SYSINDEXSTATS catalog table 1062  
 column of SYSINDEXSTATS\_HIST catalog  
 table 1063  
 column of SYSJAROBJECTS catalog table 1067  
 column of SYSPACKAGE catalog table 1073  
 column of SYSPARMS catalog table 1084  
 column of SYSROUTINES catalog table 1100  
 column of SYSSEQUENCES catalog table 1109  
 column of SYSTABSTATS catalog table 1134  
 column of SYSTABSTATS\_HIST catalog table 1135  
 column of SYSTRIGGERS catalog table 1136

## P

**PACKADM authority**  
 GRANT statement 806  
 REVOKE statement 870  
**package**  
 binding  
 remote 46  
 description 14  
 dropping 768  
 invalidated  
 ALTER TABLE statement 465  
**privileges**  
 granting 816  
 remote bind 46  
 revoking 881  
**PACKAGE**  
 clause of DROP statement 768  
 clause of GRANT statement 816  
 clause of REVOKE statement 881  
**page set**  
 description 10  
**PAGESAVE column**  
 SYSTABLEPART catalog table 1122  
 SYSTABLEPART\_HIST catalog table 1124  
**PARALLEL column of SYSROUTINES catalog**  
 table 1101  
**parallel processing**  
 SET CURRENT DEGREE statement 907  
**parameter**  
 passing to stored procedure 486, 948  
**PARAMETER CCSID clause**  
 CREATE FUNCTION statement 539, 561, 579, 590  
 CREATE PROCEDURE statement 624, 642  
**parameter marker**  
 CAST specification 127  
 description 852  
 EXECUTE statement 776  
 EXPLAIN statement 782  
 host variables in dynamic SQL 101

|                                                     |          |
|-----------------------------------------------------|----------|
| parameter marker ( <i>continued</i> )               |          |
| obtaining information with DESCRIBE INPUT           | 758      |
| OPEN statement                                      | 843      |
| PREPARE statement                                   | 852      |
| rules                                               | 852      |
| PARAMETER STYLE clause                              |          |
| ALTER FUNCTION statement                            | 399      |
| ALTER PROCEDURE statement                           | 432      |
| CREATE FUNCTION statement                           | 542, 562 |
| CREATE PROCEDURE statement                          | 628      |
| PARAMETER VARCHAR clause                            |          |
| CREATE FUNCTION statement                           | 539, 561 |
| CREATE PROCEDURE statement                          | 624      |
| PARAMETER_STYLE column of SYSROUTINES catalog table | 1102     |
| parent key                                          | 6        |
| parent row                                          | 7        |
| parent table                                        | 7        |
| PARENTS column of SYSTABLES catalog table           | 1127     |
| PARM_COUNT column of SYSROUTINES catalog table      | 1100     |
| PARM_SIGNATURE column of SYSROUTINES catalog table  | 1105     |
| PARM1 column of SYSROUTINES catalog table           | 1103     |
| PARM10 column of SYSROUTINES catalog table          | 1104     |
| PARM11 column of SYSROUTINES catalog table          | 1104     |
| PARM12 column of SYSROUTINES catalog table          | 1104     |
| PARM13 column of SYSROUTINES catalog table          | 1104     |
| PARM14 column of SYSROUTINES catalog table          | 1104     |
| PARM15 column of SYSROUTINES catalog table          | 1104     |
| PARM16 column of SYSROUTINES catalog table          | 1104     |
| PARM17 column of SYSROUTINES catalog table          | 1104     |
| PARM18 column of SYSROUTINES catalog table          | 1104     |
| PARM19 column of SYSROUTINES catalog table          | 1104     |
| PARM2 column of SYSROUTINES catalog table           | 1103     |
| PARM20 column of SYSROUTINES catalog table          | 1104     |
| PARM21 column of SYSROUTINES catalog table          | 1104     |
| PARM22 column of SYSROUTINES catalog table          | 1104     |
| PARM23 column of SYSROUTINES catalog table          | 1104     |
| PARM24 column of SYSROUTINES catalog table          | 1104     |
| PARM25 column of SYSROUTINES catalog table          | 1104     |
| PARM26 column of SYSROUTINES catalog table          | 1104     |
| PARM27 column of SYSROUTINES catalog table          | 1104     |
| PARM28 column of SYSROUTINES catalog table          | 1104     |
| PARM29 column of SYSROUTINES catalog table          | 1104     |
| PARM3 column of SYSROUTINES catalog table           | 1103     |
| PARM30 column of SYSROUTINES catalog table          | 1104     |
| PARMLIST column                                     |          |
| SYSFIELDS catalog table                             | 1052     |
| SYSPROCEDURES catalog table                         | 1095     |
| PARMNAME column of SYSPARMS catalog table           | 1084     |
| PART                                                |          |
| clause of ALTER INDEX statement                     | 423      |
| clause of ALTER TABLESPACE statement                | 471      |
| clause of CREATE AUXILIARY TABLE statement          | 518      |
| clause of CREATE INDEX statement                    | 611      |
| clause of CREATE TABLESPACE statement               | 692      |
| clause of LOCK TABLE statement                      | 840      |
| PARTITION column                                    |          |
| SYSAUXRELS catalog table                            | 1023     |
| SYSCOLDISTSTATS catalog table                       | 1030     |
| SYSCOLSTATS catalog table                           | 1031     |
| SYSINDEXPART catalog table                          | 1058     |
| SYSINDEXPART_HIST catalog table                     | 1061     |
| SYSINDEXSTATS catalog table                         | 1062     |
| SYSINDEXSTATS_HIST catalog table                    | 1063     |
| SYSTABLEPART catalog table                          | 1121     |
| SYSTABLEPART_HIST catalog table                     | 1124     |
| SYSTABLESPACE catalog table                         | 1131     |
| SYTABSTATS catalog table                            | 1134     |
| SYTABSTATS_HIST catalog table                       | 1135     |
| PASSWORD column                                     |          |
| USERNAMES catalog table                             | 1143     |
| PATH clause                                         |          |
| SET PATH statement                                  | 922      |
| PATHSCHEMAS column                                  |          |
| SYSCHECKS2 catalog table                            | 1026     |
| SYSPACKAGE catalog table                            | 1077     |
| SYSPLAN catalog table                               | 1089     |
| SYSVIEWS catalog table                              | 1141     |
| PCTFREE                                             |          |
| clause of ALTER INDEX statement                     | 422      |
| clause of ALTER TABLESPACE statement                | 471      |
| clause of CREATE INDEX statement                    | 609      |
| clause of CREATE TABLESPACE statement               | 688      |
| column of SYSINDEXPART catalog table                | 1058     |
| column of SYSTABLEPART catalog table                | 1122     |
| PCTTIMESTAMP column of SYSPACKAGE catalog table     | 1075     |
| PCTPAGES column                                     |          |
| SYSTABLES catalog table                             | 1126     |
| SYSTABLES_HIST catalog table                        | 1130     |
| SYTABSTATS catalog table                            | 1134     |

PCTROWCOMP column  
   SYSTABLES catalog table  
     description 1128  
   SYSTABLES\_HIST catalog table 1130  
   SYTABSTATS catalog table 1134  
 PDSNAME column  
   SYSDBRM catalog table 1049  
   SYSPACKAGE catalog table 1075  
 PERACTIVE column  
   SYTABLEPART catalog table  
     description 1121  
   SYTABLEPART\_HIST catalog table 1124  
 PERCDROP column  
   SYTABLEPART catalog table  
     description 1121  
   SYTABLEPART\_HIST catalog table 1124  
 PERIOD option of precompiler 148  
 PGM\_TYPE column of SYSPROCEDURES catalog  
   table 1095  
 PGSIZE column  
   SYSINDEXES catalog table 1055  
   SYSTABLESPACE catalog table 1131  
 PIECESIZE clause  
   ALTER INDEX statement 417  
   CREATE INDEX statement 613  
   effect on indexes 417  
 PIECESIZE column of SYSINDEXES catalog  
   table 1056  
 PIT\_RBA column of SYSCOPY catalog table 1042  
 PKSIZE column of SYSPACKAGE catalog table 1073  
 PL/I application program  
   host structure 103  
   host variable  
     description 100  
   INCLUDE SQLCA 985  
   INCLUDE SQLDA 1002  
   varying-length string 52  
 plan element 14, 912  
 plan table 781  
 PLAN\_TABLE table  
   EXPLAIN statement 782  
 PLAN  
   clause of EXPLAIN statement 781  
   clause of GRANT statement 818  
   clause of REVOKE statement 883  
 PLANNAMEN column  
   SYSIBM.MODESELECT catalog table 1022  
   SYSPACKLIST catalog table 1080  
 PLCREATOR column  
   SYSDBRM catalog table 1049  
   SYSSTMT catalog table 1111  
 PLENTRIES column of SYSPLAN catalog table 1088  
 PLNAME column  
   SYSDBRM catalog table 1049  
   SYSSTMT catalog table 1111  
 PLSIZE column of SYSPLAN catalog table 1087  
 POBJECT\_LIB column  
   SYSJAVAOPTS catalog table 1068  
 point of consistency  
   description 11  
 PORT column of LOCATIONS catalog table 1017  
 POSSTR function 278  
 POWER function 280  
 PQTY column  
   SYSINDEXPART catalog table 1058  
   SYSINDEXPART\_HIST catalog table 1061  
   SYSTABLEPART catalog table 1121  
   SYSTABLEPART\_HIST catalog table 1124  
 precedence of operators 123  
 precision of numbers  
   description 55  
   determined by SQLLEN variable 995  
   in assignments 66  
   in comparisons 72  
   results of arithmetic operations 114  
   values for data types 55  
 PRECOMPDAT column of SYSDBRM catalog  
   table 1049  
 PRECOMPILE\_OPTS column of  
   SYSROUTINES\_OPTS catalog table 1106  
 precompiler  
   checks SQL statements 736  
   DECLARE TABLE statement 736  
   DECLARE VARIABLE statement 739  
   escape character 33  
   options  
     COBOL decimal point 148  
     CONNECT 146  
     date 151  
     NOFOR 153  
     STDSQL 151  
     string delimiter 149  
     time 151  
   SET CURRENT APPLICATION ENCODING  
     SCHEME statement 906  
     using INCLUDE statements 830  
 PRECOMPTIME column of SYSDBRM catalog  
   table 1049  
 PRECOMPTS column of SYSDBRM catalog  
   table 1050  
 predicate  
   basic 130  
   BETWEEN 134  
   description 130  
   EXISTS 134  
   IN 136  
   LIKE 137  
   NULL 144  
   quantified 132  
 prefix operator 114  
 PRELINK\_OPTS column  
   SYSROUTINES\_OPTS catalog table 1106  
 PREPARE statement  
   description 846  
   example 857  
 prepared SQL statement  
   dynamically prepared by PREPARE 846  
   executing 776  
   identifying by DECLARE 735  
   obtaining information  
     with DESCRIBE 749  
     with DESCRIBE INPUT 758

prepared SQL statement (*continued*)  
     SQLDA provides information 987  
     statements allowed 969  
**PRIMARY KEY** clause  
     ALTER TABLE statement  
         description 458  
     CREATE TABLE statement 661, 668  
**PRIQTY** clause  
     ALTER INDEX statement 419  
     ALTER TABLESPACE statement 473  
     CREATE INDEX statement 607  
     CREATE TABLESPACE statement 685  
**privilege**  
     granting 803  
     revoking 865  
     types 803  
**PRIVILEGE** column of SYSCOLAUTH catalog  
     table 1027  
**procedure, stored**  
     naming convention 37  
**PROCEDURE**  
     clause of COMMENT statement 497  
     clause of DROP statement 768  
     column of SYSPROCEDURES catalog table 1094  
**process**  
     description 11  
**PROGRAM TYPE** clause  
     ALTER FUNCTION statement 405  
     ALTER PROCEDURE statement 435  
     CREATE FUNCTION statement 548, 568  
     CREATE PROCEDURE statement 441, 631, 645  
**PROGRAM\_TYPE** column of SYSROUTINES catalog  
     table 1103  
**promotion of data types** 61  
**PSEUDO\_DEL\_ENTRIES** column  
     SYSINDEXPART catalog table 1059  
     SYSINDEXPART\_HIST catalog table 1061  
**PSID** column of SYSTABLESPACE catalog table 1131  
**PUBLIC AT ALL LOCATIONS** clause  
     GRANT statement 804  
     REVOKE statement 866  
**PUBLIC** clause  
     GRANT statement 804  
     REVOKE statement 866

## Q

qualification of column names 95

**QUALIFIER**  
     column of SYSPACKAGE catalog table 1073  
     column of SYPLAN catalog table 1088  
     column of SYSRESAUTH catalog table 1098  
         unqualified object names 39, 40  
**quantified predicate** 132  
**QUARTER** function 281  
**query** 348  
**QUERYNO** clause  
     DELETE statement 745  
     INSERT statement 835  
     SELECT INTO statement 902  
     select-statement 374

**QUERYNO** clause (*continued*)  
     UPDATE statement 933  
**QUERYNO** column  
     SYSPACKSTMT catalog table 1083  
     SYSSTMT catalog table 1113  
**question mark (?)** 776  
**quotation mark** 33, 149  
**QUOTE**  
     column of SYSDBRM catalog table 1049  
     column of SYSPACKAGE catalog table 1074  
     option of precompiler 149  
**QUOTESQL** option of precompiler 149

## R

**RACF** (Resource Access Control Facility)  
     security for remote execution 47  
**RADIANS** function 282  
**RAISE\_ERROR** function 283  
**RAND** function 284  
**RBA** column of SYSCHECKS catalog table 1025  
**RBA1** column of SYSTABLES catalog table 1128  
**RBA2** column of SYSTABLES catalog table 1128  
**READ SQL** clause  
     ALTER PROCEDURE statement 433  
**READ SQL DATA** clause  
     ALTER FUNCTION statement 400  
**read-only**  
     FOR FETCH ONLY clause 371  
     FOR READ ONLY clause 371  
     result table 721  
     view 715  
**READS SQL DATA** clause  
     ALTER FUNCTION statement 413  
     ALTER PROCEDURE statement 440  
     CREATE FUNCTION statement 543, 563, 592  
     CREATE PROCEDURE statement 628, 643  
**REAL** data type  
     CREATE TABLE statement 659  
     description 55  
**REAL** function 285  
**RECLENGTH** column of SYSTABLES catalog  
     table 1127  
**RECOVER** privilege  
     GRANT statement 822  
     REVOKE statement 887  
**RECOVERAUTH** column of SYSUSERAUTH catalog  
     table 1138  
**RECOVERDB** privilege  
     GRANT statement 808  
     REVOKE statement 872  
**RECOVERDBAUTH** column of SYSDBAUTH catalog  
     table 1048  
**recovery**  
     COMMIT statement 499  
     description  
         restoring data consistency 11  
**REFCOLS** column of SYTABAUTH catalog  
     table 1119  
**REFERENCES** clause  
     ALTER TABLE statement 459

REFERENCES clause (*continued*)  
     CREATE TABLE statement 669  
 REFERENCES privilege  
     GRANT statement 825  
     REVOKE statement 889  
 REFERENCESAUTH column of SYSTABAUTH catalog  
     table 1119  
 REFERENCING clause of TRIGGER statement 702  
 referential constraint  
     ALTER TABLE statement 458  
     CREATE TABLE statement 668  
     description 7  
 referential cycle 7  
 referential integrity  
     description 7  
 REFTBCREATOR column of SYSRELS catalog  
     table 1097  
 REFTBNAME column of SYSRELS catalog table 1097  
 RELBOUND column  
     SYSPACKAGE catalog table 1077  
     SYSPLAN catalog table 1090  
 RELCREATED column  
     SYSTABLES catalog table 1129  
     SYSVIEWS catalog table 1141  
 RELEASE  
     column of SYSPACKAGE catalog table 1074  
     column of SYSPLAN catalog table 1087  
 RELEASE (connection) statement  
     description 859  
     example 860  
 release dependency indicators 1005  
 release level identification, current server 506, 513  
 release pending connection state 18  
 RELEASE SAVEPOINT statement  
     description 862  
     example 862  
 RELNAME column  
     SYSFOREIGNKEYS catalog table 1053  
     SYSRELS catalog table 1097  
 RELOBID1 column of SYSRELS catalog table 1097  
 RELOBID2 column of SYSRELS catalog table 1097  
 REMARKS column  
     SYSCOLUMNS catalog table 1034  
     SYSDATATYPES catalog table 1046  
     SYSINDEXES catalog table 1056  
     SYSROUTINES catalog table 1105  
     SYSSEQUENCES catalog table 1109  
     SYSTABLES catalog table 1126, 1145  
     SYSTRIGGERS catalog table 1136  
 REMOTE column of SYSPACKAGE catalog  
     table 1075  
 Remote Recovery Data Facility (RRDF) 674  
 REMOVE VOLUMES clause of ALTER STOGROUP  
     statement 445  
 RENAME statement  
     description 863  
     example 864  
 REOPTVAR column  
     SYSPACKAGE catalog table 1076  
     SYSPLAN catalog table 1089  
 REORG privilege  
     GRANT statement 808  
     REVOKE statement 872  
 REORGAUTH column of SYSDBAAUTH catalog  
     table 1048  
 REPAIR privilege  
     GRANT statement 808  
     REVOKE statement 872  
 REPAIRAUTH column of SYSDBAAUTH catalog  
     table 1048  
 REPEAT function 286  
 REPEAT statement  
     example 963  
     SQL procedure 963  
 REPLACE function 288  
 reserved keywords 1153  
 RESET  
     clause of CONNECT (Type 1) statement 505  
     clause of CONNECT (Type 2) statement 511  
 RESTRICT  
     delete rule  
         ALTER TABLE statement 460  
         CREATE TABLE statement 670  
         description 7  
 RESTRICT clause of REVOKE statement 867, 875  
 result column  
     data type 351  
     names 351  
 result set locator  
     description 102  
 result table  
     description 4  
 RESULT\_COLS column of SYSROUTINES catalog  
     table 1105  
 RESULT\_SETS column  
     SYSPROCEDURES catalog table 1095  
     SYSROUTINES catalog table 1103  
 RETURN clause of CREATE FUNCTION (SQL scalar)  
     statement 592  
 RETURN\_TYPE column of SYSROUTINES catalog  
     table 1100  
 RETURNS clause  
     CREATE FUNCTION statement 589  
 RETURNS clause of CREATE FUNCTION  
     statement 538, 578  
 RETURNS NULL ON NULL INPUT clause  
     ALTER FUNCTION statement 400  
     CREATE FUNCTION statement 543, 563  
 RETURNS TABLE clause of CREATE FUNCTION  
     statement 560  
 REVOKE statement  
     cascading effect 867  
     collection privileges 870  
     database privileges 871  
     description 865  
     distinct type privileges 874  
     function privileges 876  
     JAR privileges 874  
     package privileges 881  
     plan privileges 883  
     procedure privileges 876

**REVOKE** statement (*continued*)
   
 schema privileges 884
   
 system privileges 886
   
 table privileges 889
   
 use privileges 892
   
 view privileges 889
   
**RIGHT** function 290
   
**RIGHT OUTER JOIN**

- example 363
- FROM** clause of subselect 356

**rollback**

- description 11

**ROLLBACK** statement
   
 description 894
   
 example 895
   
**ROUND** function 292
   
**ROUND\_TIMESTAMP** function 294
   
**ROUTINEID** column
   
 SYSPARMS catalog table 1084
   
 SYSROUTINES catalog table 1100
   
**ROUTINENAME** column
   
 SYSROUTINES\_OPTS catalog table 1106
   
 SYSROUTINES\_SRC catalog table 1107
   
**ROUTINETYPE** column
   
 SYSPARMS catalog table 1084
   
 SYSROUTINEAUTH catalog table 1099
   
 SYSROUTINES catalog table 1100
   
**row**

- deleting 742
- description 4
- inserting 832
- selecting single row 900
- updating 928

**row ID**

- assignment of values 71
- comparison of values 75
- data type 60, 661

**ROWID**

- data type
  - CREATE TABLE statement 661
  - description 60
  - function 297

**ROWTYPE** column of SYSPARMS catalog table 1084
   
**RRDF** (Remote Recovery Data Facility)
 

- altering a table for 463
- creating a table for 674

**RTRIM** function 298
   
 run behavior for dynamic SQL statements 44
   
**RUN OPTIONS** clause
 

- ALTER FUNCTION statement 406
- ALTER PROCEDURE statement 436, 442
- CREATE FUNCTION statement 549, 569
- CREATE PROCEDURE statement 631, 646

**RUNOPTS** column
 

- SYSPROCEDURES catalog table 1095
- SYSROUTINES catalog table 1105

  
**S**

- sample table
  - description 4

 sample user-defined functions 1155
   
**savepoint**

- releasing 862
- setting 897

**savepoint identifier**

- naming convention 38

**savepoint name**

- naming convention 38

**SAVEPOINT** statement
 

- description 897
- example 898

**SBCS** data
 

- description 23, 49

**SBCS\_CCSID** column
 

- SYSDATABASE catalog table 1044
- SYSTABLESPACE catalog table 1133

**SCALE** column
 

- SYSCOLUMNS catalog table 1033
- SYSDATATYPES catalog table 1046
- SYSFIELDS catalog table 1052
- SYSPARMS catalog table 1085

**scale of numbers**

- assignments 67
- comparisons 72
- description 55
- results of arithmetic operations 115

**schema**

- description 3
- privileges 819, 884

**SCHEMA** column
 

- SYSDATATYPES catalog table 1046
- SYSPARMS catalog table 1084
- SYSROUTINEAUTH catalog table 1099
- SYSROUTINES catalog table 1100
- SYSROUTINES\_OPTS catalog table 1106
- SYSROUTINES\_SRC catalog table 1107
- SYSSEQUENCES catalog table 1109
- SYSTRIGGERS catalog table 1136

**schema name**

- naming convention 37

**schema processor** 5, 604, 661, 662
   
**SCHEMANAME** column
 

- SYSSCHEMAAUTH catalog table 1108

**SCRATCHPAD** clause
 

- ALTER FUNCTION statement 402
- CREATE FUNCTION statement 544, 565

**SCRATCHPAD** column of SYSROUTINES catalog
 

- table 1101

**SCRATCHPAD\_LENGTH** column of SYSROUTINES catalog table 1101
   
**SCREATOR** column of SYSTABAUTH catalog
 

- table 1118

**SCROLL** clause
 

- DECLARE CURSOR statement 720

**search condition**

- DELETE statement 744
- description 145
- HAVING clause 359
- order of evaluation 145
- UPDATE statement 932
- WHERE clause 358

## S

sample table
   
 description 4

SECOND function 299  
 SECQTY clause  
   ALTER INDEX statement 420  
   ALTER TABLESPACE statement 473  
   CREATE INDEX statement 607  
   CREATE TABLESPACE statement 686  
 SECQTYI column  
   SYSINDEXPART catalog table 1059  
   SYSINDEXPART\_HIST catalog table 1061  
   SYSTABLEPART catalog table 1123  
   SYSTABLEPART\_HIST catalog table 1124  
 SECTNO column  
   SYSPACKSTMT catalog table 1081  
   SYSSTM catalog table 1111  
 SECTNOI column  
   SYSPACKSTMT catalog table 1083  
   SYSSTM catalog table 1113  
 SECURITY clause  
   ALTER FUNCTION statement 406  
   ALTER PROCEDURE statement 435, 442  
   CREATE FUNCTION statement 548, 569  
   CREATE PROCEDURE statement 632, 645  
 SECURITY\_IN column of LUNAMES catalog  
   table 1020  
 SECURITY\_OUT column  
   IPNAMES catalog table 1016  
   LUNAMES catalog table 1020  
 SEGSIZE  
   clause of CREATE TABLESPACE statement 694  
   column of SYSTABLESPACE catalog table 1132  
 SELECT  
   clause as syntax component 349  
 SELECT INTO statement  
   description 900  
   example 902  
 SELECT privilege  
   GRANT statement 825  
   REVOKE statement 890  
 SELECT statement  
   description 369  
   dynamic invocation 383  
   example 375  
     SYSIBM.SYSCOLUMNS 1146  
     SYSIBM.SYSINDEXES 1147  
     SYSIBM.SYSTABAUTH 1147  
     SYSIBM.SYSTABLES 1145, 1151  
   fullselect 365  
   list  
     application 350  
     description 349  
     maximum number of elements 966  
     notation 350  
     static invocation 383  
     subselect 349  
 SELECTAUTH column of SYSTABAUTH catalog  
   table 1119  
 selecting  
   single row 900  
 self-referencing constraint 7  
 self-referencing row 7  
 self-referencing table 7  
 SENSITIVE clause  
   FETCH statement 794  
 SENTITIVE clause  
   DECLARE CURSOR statement 719  
 SEQNO column  
   SYSPACKLIST catalog table 1080  
   SYSPACKSTMT catalog table 1081  
   SYSROUTINES\_SRC catalog table 1107  
   SYSSTM catalog table 1111  
   SYSTRIGGERS catalog table 1136  
   SYSVIEWS catalog table 1141  
 SEQTYPE column of SYSSEQUENCES catalog  
   table 1109  
 SEQUENCEID column of SYSSEQUENCES catalog  
   table 1109  
 server  
   accessible 15  
   current 16  
   establishing with CONNECT 503  
   remote 14  
 SET clause of UPDATE statement 931  
 SET CONNECTION statement  
   description 904  
   example 905  
 SET CURRENT APPLICATION ENCODING SCHEME  
   statement  
     description 906  
     example 906  
 SET CURRENT DEGREE statement  
   description 907  
   example 907  
 SET CURRENT LOCALE LC\_CTYPE statement  
   description 909  
   example 910  
 SET CURRENT OPTIMIZATION HINT statement  
   description 911  
   example 911  
 SET CURRENT PACKAGESET statement  
   description 912  
   example 913  
 SET CURRENT PRECISION statement  
   description 914  
   example 914  
 SET CURRENT RULES statement  
   description 915  
   example 915  
 SET CURRENT SQLID statement  
   description 916  
   example 917  
 SET host-variable assignment statement  
   description 918  
   example 919  
 SET NULL delete rule  
   ALTER TABLE statement 460  
   CREATE TABLE statement 670  
   description 7  
 SET PATH statement  
   description 921  
   example 923  
 SET QUERYNO clause of EXPLAIN statement 782

SET transition-variable assignment statement  
     description 924  
     example 926  
 SGCREATOR column of SYSVOLUMES catalog  
     table 1142  
 SGNAME column of SYSVOLUMES catalog  
     table 1142  
 SHARE  
     option of LOCK TABLE statement 840  
 shift-in character  
     convention xvi  
     LABEL ON statement 839  
     not truncated by assignments 69  
 shift-out character  
     convention xvi  
     LABEL ON statement 839  
 short identifier in SQL 33  
 short string column 51, 52  
 SHRLEVEL  
     column of SYSCOPY catalog table 1041  
 SIGN function 300  
 sign-on exit routine  
     CURRENT SQLID special register 44, 91  
 SIGNAL SQLSTATE statement  
     description 927  
     example 927  
 SIN function 301  
 single precision floating-point number 55  
 single-fetch clause  
     FETCH statement 798  
 SINH function 302  
 SMALLINT function 303  
 SOAPHTTPC and SOAPHTTPV functions 304  
 SOME quantified predicate 132  
 SOURCE clause of CREATE FUNCTION  
     statement 579  
 SOURCEDSN column  
     SYSROUTINES\_OPTS catalog table 1106  
 SOURCESCHEMA column  
     SYSDATATYPES catalog table 1046  
     SYSROUTINES catalog table 1100  
 SOURCESPECIFIC column of SYSROUTINES catalog  
     table 1100  
 SOURCETYPE column of SYSDATATYPES catalog  
     table 1046  
 SOURCETYPEID column  
     DATATYPES catalog table 1046  
     SYSCOLUMNS catalog table 1036  
     SYSPARMS catalog table 1085  
     SYSSEQUENCES catalog table 1109  
 space character 31  
 SPACE column  
     SYSINDEXES catalog table 1055  
     SYSINDEXPART catalog table 1059  
     SYSSTOGROUP catalog table 1114  
     SYSTABLEPART catalog table 1122  
     SYSTABLESPACE catalog table 1132  
 SPACE function 305  
 SPACEF column  
     SYSINDEXES catalog table 1056  
     SYSINDEXES\_HIST catalog table 1057  
 SPACEF column (*continued*)  
     SYSINDEXPART catalog table 1059  
     SYSINDEXPART\_HIST catalog table 1061  
     SYSTABLEPART catalog table 1123  
     SYSTABLEPART\_HIST catalog table 1124  
     SYSTABLES catalog table 1129  
     SYSTABLES\_HIST catalog table 1130  
 SPCDATE column of SYSSTOGROUP catalog  
     table 1114  
 special character 31  
 special register  
     behavior in user-defined functions and stored  
         procedures 93  
 CURRENT APPLICATION ENCODING  
     SCHEME 85  
 CURRENT DATE 86  
 CURRENT DEGREE 86  
 CURRENT LOCALE LC\_CTYPE 87  
 CURRENT MEMBER 87  
 CURRENT OPTIMIZATION HINT 87  
 CURRENT PACKAGESET 88  
 CURRENT PATH 88  
 CURRENT PRECISION 89  
 CURRENT RULES 90  
 CURRENT SERVER 91  
 CURRENT SQLID 91  
 CURRENT TIME 91  
 CURRENT TIMESTAMP 92  
 CURRENT TIMEZONE 92  
 CURRENT\_DATE 86  
 CURRENT\_TIME 91  
 CURRENT\_TIMESTAMP 92  
 description 82  
 USER 92  
 values in trigger 707  
 SPECIAL REGISTER column  
     SYSROUTINES catalog table 1105  
 SPECIFIC clause  
     CREATE FUNCTION statement 538, 560, 578, 589  
 SPECIFIC FUNCTION clause of ALTER FUNCTION  
     statement 397, 411  
 specific name  
     naming convention 37  
     unqualified name 40  
 SPECIFICNAME column  
     SYSPARMS catalog table 1084  
     SYSROUTINEAUTH catalog table 1099  
     SYSROUTINES catalog table 1100  
 SQL (Structured Query Language)  
     assignment operation 64  
     character 31  
     comparison operation 64  
     constants 79  
     data types  
         binary strings 53  
         casting 62  
         character strings 49  
         datetime 56  
         description 48  
         graphic strings 52  
         LOBs (large objects) 53

SQL (Structured Query Language) (*continued*)  
 data types (*continued*)  
   numbers 55  
   promotion 61  
   results of an operation 77  
   row ID 60  
   deferred embedded 2  
   delimited identifier 33  
   description 2  
   dynamic  
     description 2  
     statements allowed 969  
   identifier 32  
   interactive 3  
   JDBC 3  
   keywords, reserved 1153  
   limits 965  
   naming conventions 34  
   null value 48  
   ODBC (Open Database Connectivity) 3  
   Open Database Connectivity (ODBC) 3  
   ordinary identifier 31  
   rules 90  
   SQLJ 3  
   standard xvi, 151  
   static  
     description 2  
   token 31  
   value 48  
   variable names 34  
 SQL path 40, 107  
 SQL procedure  
   statements allowed 974  
 SQL procedure statement  
   assignment statement 946  
   CALL statement 948  
   CASE statement 950  
   compound statement 952  
   CONTINUE handler 955  
   EXIT handler 955  
   GET DIAGNOSTICS statement 957  
   GOTO statement 958  
   handler 955  
   handling errors 955  
   IF statement 960  
   LEAVE statement 961  
   LOOP statement 962  
   order of statements 955  
   REPEAT statement 963  
   WHILE statement 964  
 SQL scalar statements  
   ALTER FUNCTION 408  
 SQL statements  
   ALLOCATE CURSOR 386  
   ALTER DATABASE 388  
   ALTER FUNCTION 391  
   ALTER INDEX 414  
   ALTER PROCEDURE (external) 427  
   ALTER PROCEDURE (SQL) 438  
   ALTER STOGROUP 444  
   ALTER TABLE 447

SQL statements (*continued*)  
   ALTER TABLESPACE 467  
   ASSOCIATE LOCATORS 479  
   BEGIN DECLARE SECTION 482  
   CALL 483  
   catalog table restrictions 1011  
   CLOSE 491  
   COMMENT 493  
   COMMIT 499  
   CONNECT (Type 1) 504  
   CONNECT (Type 2) 510  
   CONNECT differences 501  
   CONTINUE 941  
   CREATE ALIAS 514  
   CREATE AUXILIARY TABLE 516  
   CREATE DATABASE 519  
   CREATE DISTINCT TYPE 522  
   CREATE FUNCTION 529  
   CREATE FUNCTION (external scalar) 530  
   CREATE FUNCTION (external table) 553  
   CREATE FUNCTION (sourced) 571  
   CREATE FUNCTION (SQL scalar) 585  
   CREATE GLOBAL TEMPORARY TABLE 595  
   CREATE INDEX 601  
   CREATE PROCEDURE 617, 636  
   CREATE STOGROUP 648  
   CREATE SYNONYM 651  
   CREATE TABLE 653  
   CREATE TABLESPACE 681  
   CREATE TRIGGER 699  
   CREATE VIEW 711  
   DECLARE CURSOR  
     description 718  
     example 723  
   DECLARE GLOBAL TEMPORARY TABLE 725  
   DECLARE STATEMENT 735  
   DECLARE TABLE 736  
   DECLARE VARIABLE 739  
   DELETE  
     description 742  
     example 747  
   DESCRIBE 749  
   DESCRIBE CURSOR 756  
   DESCRIBE INPUT 758  
   DESCRIBE PROCEDURE 760  
   DROP 763  
   END DECLARE SECTION 775  
   EXECUTE 776  
   EXECUTE IMMEDIATE 779  
   EXPLAIN  
     description 781  
     example 792  
   FETCH  
     description 793  
     example 801  
   FOR 782  
   FREE LOCATOR 802  
   GRANT 803  
   HOLD LOCATOR 829  
   INCLUDE  
     description 830

**SQL statements (*continued*)**  
**INCLUDE (*continued*)**  
 example 831  
**SQLCA** 984  
**SQLDA** 998  
**INSERT**  
 description 832  
 example 837  
 invocation 380  
**LABEL ON** 838  
**LOCATE LC\_CTYPE** 909  
**LOCK TABLE** 840  
**OPEN**  
 description 842  
 example 845  
**PREPARE** 846  
**RELEASE (connection)** 859  
**RELEASE SAVEPOINT** 862  
 remote execution  
 description 46  
 dynamic execution 47  
 static execution 47  
**RENAME** 863  
**REVOKE** 865  
**ROLLBACK** 894  
**SAVEPOINT** 897  
**SELECT** 899  
**SELECT INTO** 900  
**SET CONNECTION** 904  
**SET CURRENT APPLICATION ENCODING SCHEME** 906  
**SET CURRENT DEGREE** 907  
**SET CURRENT OPTIMIZATION HINT** 911  
**SET CURRENT PRECISION** 914  
**SET CURRENT RULES** 915  
**SET CURRENT SQLID** 916  
**SET host-variable assignment** 918  
**SET PATH** 921  
**SET transition-variable assignment** 924  
**SIGNAL SQLSTATE** 927  
**UPDATE**  
 description 928  
 example 935  
**VALUES** 938  
**VALUES INTO** 939  
**WHENEVER** 941  
**SQL variable name**  
 naming convention 37  
**SQL\_DATA\_ACCESS column of SYSROUTINES catalog table** 1102  
**SQLCA (SQL communication area)**  
 contents 979  
 entry changed by UPDATE 933  
 INCLUDE statement 830  
**SQLCABC field of SQLCA** 979  
**SQLCAID field of SQLCA** 979  
**SQLCODE**  
 -752 508  
 -900 508  
 -918 508  
 +100 384, 835, 842, 900, 941  
**SQLCODE (*continued*)**  
 description 384  
 field of SQLCA 979  
**SQLD field of SQLDA** 752, 989  
**SQLDA (SQL descriptor area)**  
 clause of INCLUDE statement 830  
 contents 986, 988  
**SQLDABC field of SQLDA** 752, 988  
**SQLDAID field of SQLDA** 752, 988  
**SQLDATA field of SQLDA** 753, 992  
**SQLDATAL field of SQLDA** 994  
**SQLDATALEN field of SQLDA** 994  
**SQLDATATYPE field of SQLDA** 754  
**SQLDATATYPE-NAME field of SQLDA** 995  
**SQLERRD(3) field of SQLCA** 746  
**SQLERRD(n) field of SQLCA** 979  
**SQLERRMC field of SQLCA** 979  
**SQLERRML field of SQLCA** 979  
**SQLERROR**  
 clause of WHENEVER statement 941  
 column of SYSPACKAGE catalog table 1075  
**SQLERRP field of SQLCA** 979  
**SQLIND field of SQLDA** 753, 992  
**SQLJ** 3  
**SQLLEN field of SQLDA** 753, 992  
**SQLLONGL field of SQLDA** 993  
**SQLLONGLEN field of SQLDA** 754, 993  
**SQLN field of SQLDA**  
 description 750, 989  
**SQLNAME field of SQLDA** 753, 993  
**SQLRULES**  
 column of SYSPLAN catalog table 1088  
**SQLSTATE**  
 '02000' 835, 842, 900, 941  
 description 385  
 field of SQLCA 979  
 signaling 927  
**SQLTNAME field of SQLDA** 995  
**SQLTYPE field of SQLDA**  
 description 992  
 values 753, 995  
**SQLVAR**  
 base 753  
 extended 753  
**SQLVAR field of SQLDA** 753  
**SQLWARN6 field of SQLCA** 120  
**SQLWARNING clause**  
 WHENEVER statement 941  
**SQLWARNn field of SQLCA** 979  
**SQRT function** 306  
**SQTY column**  
 SYSINDEXPART catalog table 1058  
 SYSTABLEPART catalog table 1121  
**standard, SQL (ANSI/ISO)**  
 description xvi  
 SET CONNECTION statement 904  
 SQL-style comments 152  
 STDSQL precompiler option 151  
**START column of SYSSEQUENCES catalog table** 1109  
**START\_RBA column of SYSCOPY catalog table** 1040

**STARTDB** privilege  
 GRANT statement 808  
 REVOKE statement 872  
**STARTDBAUTH** column of **SYSDBAUTH** catalog  
 table 1048  
**state**  
 application process 504  
 SQL connection 18  
**statement**  
 descriptions 2  
 operational form 2  
 preparation 2  
 source form 2  
**STATEMENT** clause of **DECLARE STATEMENT**  
 statement 735  
**statement table**  
 EXPLAIN statement 781  
**STATIC** clause  
 DECLARE CURSOR statement 719  
**STATIC DISPATCH** clause  
 ALTER FUNCTION statement 406, 413  
 CREATE FUNCTION statement 549, 569, 592  
**static SQL**  
 description 2, 380  
 invocation of SELECT statement 383  
**STATS** privilege  
 GRANT statement 808  
 REVOKE statement 872  
**STATSAUTH** column of **SYSDBAUTH** catalog  
 table 1048  
**STATSTIME** column  
 SYSCOLDIST catalog table 1028  
 SYSCOLDIST\_HIST catalog table 1029  
 SYSCOLDISTSTATS catalog table 1030  
 SYSCOLSTATS catalog table 1031  
 SYSCOLUMNS catalog table 1035  
 SYSCOLUMNS\_HIST catalog table 1038  
 SYSINDEXES catalog table 1055  
 SYSINDEXES\_HIST catalog table 1057  
 SYSINDEXPART catalog table 1059  
 SYSINDEXPART\_HIST catalog table 1061  
 SYSINDEXSTATS catalog table 1062  
 SYSINDEXSTATS\_HIST catalog table 1063  
 SYSLOBSTATS catalog table 1071  
 SYSLOBSTATS\_HIST catalog table 1072  
 SYSSTOGROUP catalog table 1114  
 SYSTABLEPART catalog table 1122  
 SYSTABLEPART\_HIST catalog table 1124  
 SYSTABLES catalog table 1128  
 SYSTABLES\_HIST catalog table 1130  
 SYSTABLESPACE catalog table 1132  
 SYSTABSTATS catalog table 1134  
 SYSTABSTATS\_HIST catalog table 1135  
**STATUS** column  
 SYSPACKSTMT catalog table 1082  
 SYSSTMT catalog table 1112  
 SYSTABLES catalog table 1127  
 SYSTABLESPACE catalog table 1131  
**STAY RESIDENT** clause  
 ALTER FUNCTION statement 405  
 ALTER PROCEDURE statement 435, 441

**STAY RESIDENT** clause (*continued*)  
 CREATE FUNCTION statement 548, 568  
 CREATE PROCEDURE statement 631, 645  
**STAYRESIDENT** column  
 SYSPROCEDURES catalog table 1095  
 SYSROUTINES catalog table 1102  
 STD SQL LANGUAGE field of panel DSNTIP4 151  
 STDDEV function 168  
 STDDEV\_SAMP function 169  
 STDSQL option  
 precompiler 151  
**STGROUPO** column of **SYSDATABASE** catalog  
 table 1044  
**STM** column of **SYSPACKSTM** catalog table 1081  
**STMNO** column  
 SYSPACKSTM catalog table 1081  
 SYSSTM catalog table 1111  
**STMNOI** column  
 SYSPACKSTM catalog table 1082  
 SYSSTM catalog table 1112  
**STNAME** column of **SYSTABAUTH** catalog table 1118  
**STOGROUP**  
 clause of ALTER DATABASE statement 389  
 clause of ALTER INDEX statement 419, 422  
 clause of ALTER STOGROUP statement 444  
 clause of ALTER TABLESPACE statement 472  
 clause of CREATE DATABASE statement 520  
 clause of CREATE INDEX statement 606, 608  
 clause of CREATE STOGROUP statement 648  
 clause of CREATE TABLESPACE statement 685, 688  
 clause of DROP statement 769  
**STOGROUP** privilege  
 GRANT statement 827  
 REVOKE statement 892  
**STOPALL** privilege  
 GRANT statement 822  
 REVOKE statement 887  
**STOPALLAUTH** column of **SYSUSERAUTH** catalog  
 table 1138  
**STOPAUTH** column of **SYSDBAUTH** catalog  
 table 1048  
**STOPDB** privilege  
 GRANT statement 808  
 REVOKE statement 872  
**storage group, DB2**  
 altering 444  
 creating 648  
 description 10  
 dropping 769  
 retrieving catalog information 1145  
**storage structure** 10  
**stored procedure**  
 altering  
 with ALTER PROCEDURE (external)  
 statement 427  
 with ALTER PROCEDURE (SQL) statement 438  
**CALL** statement 483  
 creating  
 with CREATE PROCEDURE (external)  
 statement 617

stored procedure (*continued*)
   
     creating (*continued*)
   
         with CREATE PROCEDURE (SQL)
   
             statement 636
   
 CURRENT PACKAGESET special register 913
   
 dropping 768
   
 invoking 483
   
 name, unqualified 40
   
 naming convention 37
   
 privileges
   
     granting 811
   
     revoking 876
   
 statements allowed 972
   
 unqualified name 40
   
**STORES** clause of CREATE AUXILIARY TABLE
   
     statement 517
   
**STORNAME** column
   
     SYSINDEXPART catalog table 1058
   
     SYSTABLEPART catalog table 1121
   
**STORTYPE** column
   
     SYSINDEXPART catalog table 1058
   
     SYSTABLEPART catalog table 1121
   
**STOSPACE** privilege
   
     GRANT statement 822
   
     REVOKE statement 887
   
**STOSPACEAUTH** column of SYSUSERAUTH catalog
   
     table 1138
   
**string**
  
     binary 53
   
     character 49
   
     comparison 73
   
     constant 80
   
     conversion 21
   
     datetime values 57
   
     delimiter
   
         COBOL 149
   
         controlling representation 149
   
         SQL 149
   
     description 20
   
     fixed-length
   
         description 51, 52
   
     graphic 52
   
     long column
   
         description 51, 52, 54
   
         limitations 350
   
         use restrictions 54
   
     numbers 56
   
     short 51, 52
   
     varying-length
   
         description 51, 52
   
**string delimiter** precompiler option 149
   
**STRIP** function 298, 307
   
**strong typing** 61
   
**STYPE** column of SYSCOPY catalog table 1042
   
**SUBBYTE** column of SYSSTRINGS catalog table 1115
   
**subquery**
  
     description 97
   
     HAVING clause 359
   
     WHERE clause 358
   
**subselect**
  
     CREATE VIEW statement 349
   
     subselect (*continued*)
   
         description 349
   
         example 360
   
         INSERT statement 349
   
**substitution byte** 22
   
**substitution character** 70
   
**SUBSTR** function 309
   
**SUBTYPE** column
   
     SYSDATATYPES catalog table 1046
   
     SYSPARMS catalog table 1085
   
**SUM** function 170
   
**synonym**
  
     defining 651
   
     description 41
   
     dropping 769
   
     naming convention 38
   
     qualifying a column name 95
   
**SYNONYM** clause
   
     CREATE SYNONYM statement 651
   
     DROP statement 769
   
**syntax diagrams, how to read** xvii
   
**SYSADM** authority
   
     GRANT statement 822
   
     REVOKE statement 887
   
**SYSADMAUTH** column of SYSUSERAUTH catalog
   
     table 1138
   
**SYSCTRL** authority
   
     GRANT statement 822
   
     REVOKE statement 887
   
**SYSCTRLAUTH** column of SYSUSERAUTH catalog
   
     table 1138
   
**SYSENTRIES** column
   
     SYSPACKAGE catalog table 1073
   
     SYSPLAN catalog table 1088
   
**SYSMODENAME** column of LUNAMES catalog
   
     table 1020
   
**SYSOPR** authority
   
     GRANT statement 822
   
     REVOKE statement 887
   
**SYSOPRAUTH** column of SYSUSERAUTH catalog
   
     table 1138
   
**system**
  
     limits 965
   
**SYSTEM** column
   
     SYSPKSYSTEM catalog table 1086
   
     SYSPLSYSTEM catalog table 1093
   
**SYSTEM PATH** clause
   
     SET PATH statement 921

## T

**table**
  
     altering
   
         ALTER TABLE statement 447
   
     auxiliary table 4
   
     base table 4
   
     controlling changes 9
   
     creating
   
         CREATE AUXILIARY TABLE statement 516
   
         CREATE GLOBAL TEMPORARY TABLE
   
             statement 595

table (*continued*)  
 creating (*continued*)  
 CREATE TABLE statement 653  
 DECLARE GLOBAL TEMPORARY TABLE statement 725  
 description 4  
 designator 96  
 dropping  
   DROP statement 769  
 empty table 4  
 joining 356  
 obtaining information with DESCRIBE 749  
 privileges 824  
   revoking 889  
 renaming with RENAME statement 863  
 result table 4, 844  
 retrieving  
   catalog information 1145  
   comments 1151  
 sample table 4  
 temporary copy 844  
 temporary table 4

**TABLE**  
 column of SYSPARMS catalog table 1085

table check constraint  
 catalog information 1149  
 defining  
   ALTER TABLE statement 461  
   CREATE TABLE statement 670  
 deleting rows 746  
 inserting rows 836  
 SYSCHECKDEP catalog table 1024  
 updating rows 934

table function 155

**TABLE LIKE** clause  
 CREATE FUNCTION statement 537, 559, 577  
 CREATE PROCEDURE statement 624, 641

table name  
 naming convention 38  
 qualifying a column name 95  
 unqualified 39

table space  
 altering with ALTER TABLESPACE statement 467  
 catalog table 1006  
 creating  
   CREATE TABLESPACE statement 681  
   implicitly 672  
 description 10  
 dropping 769  
 naming convention 38

**TABLE\_COLNO** column of SYSPARMS catalog table 1085

**TABLE\_LOCATION** function 1165

**TABLE\_NAME** function 1167

**TABLE\_SCHEMA** function 1169

**TABLE**  
 clause of COMMENT statement 497  
 clause of DECLARE TABLE statement 736  
 clause of DROP statement 769  
 clause of LABEL ON statement 838

**TABLESPACE**  
 clause of ALTER TABLESPACE statement 467  
 clause of DROP statement 769

**TABLESPACE** privilege  
 GRANT statement 827  
 REVOKE statement 892

**TABLESTATUS** column of SYSTABLES catalog table 1129

TAN function 312

TANH function 313

**TBCREATOR** column  
 SYSCOLUMNS catalog table 1032  
 SYSCOLUMNS\_HIST catalog table 1037  
 SYSFIELDS catalog table 1052  
 SYSINDEXES catalog table 1054  
 SYSINDEXES\_HIST catalog table 1057  
 SYSKEYCOLUSE catalog table 1069  
 SYSSYNONYMS catalog table 1117  
 SYTABCONST catalog table 1120  
 SYSTABLES catalog table 1128

**TBNAME** column  
 SYSAUXRELS catalog table 1023  
 SYSCHECKDEP catalog table 1024  
 SYSCHECKS catalog table 1025  
 SYSCHECKS2 catalog table 1026  
 SYSCOLDIST catalog table 1028  
 SYSCOLDIST\_HIST catalog table 1029  
 SYSCOLDISTSTATS catalog table 1030  
 SYSCOLSTATS catalog table 1031  
 SYSCOLUMNS catalog table 1032  
 SYSCOLUMNS\_HIST catalog table 1037  
 SYSFIELDS catalog table 1052  
 SYSFOREIGNKEYS catalog table 1053  
 SYSINDEXES catalog table 1054  
 SYSINDEXES\_HIST catalog table 1057  
 SYSKEYCOLUSE catalog table 1069  
 SYSRELS catalog table 1097  
 SYSSYNONYMS catalog table 1117  
 SYTABCONST catalog table 1120  
 SYSTABLES catalog table 1128  
 SYSTRIGGERS catalog table 1136

**TBOWNER** column  
 SYSAUXRELS catalog table 1023  
 SYSCHECKDEP catalog table 1024  
 SYSCHECKS catalog table 1025  
 SYSCHECKS2 catalog table 1026  
 SYSCOLDIST catalog table 1028  
 SYSCOLDIST\_HIST catalog table 1029  
 SYSCOLDISTSTATS catalog table 1030  
 SYSCOLSTATS catalog table 1031  
 SYSTRIGGERS catalog table 1136

**TCREATOR** column of SYTABAUTH catalog table 1118

**TEMP** database  
 creating 520

**temporary**  
 copy of table 844

**temporary table**  
 creating 595, 725  
 description 4

**TEXT** column  
 SYSTMT catalog table 1111  
 SYSTRIGGERS catalog table 1136  
 SYSVIEWS catalog table 1141  
**three-part name**  
 description 16  
**time**  
 arithmetic 121  
 data type 57  
 duration 118  
 strings 58, 59  
**TIME**  
 data type  
 CREATE TABLE statement 661  
 description 57  
 function 314  
 TIME FORMAT field of panel DSNTIP4 151  
**timestamp**  
 arithmetic 122  
 data type 57  
 duration 118  
 strings 59  
**TIMESTAMP**  
 column of SYSCOPY catalog table 1041  
 column of SYSDATABASE catalog table 1044  
 column of SYSDBRM catalog table 1049  
 column of SYSPACKAGE catalog table 1073  
 column of SYSPACKAUTH catalog table 1078  
 column of SYSPACKLIST catalog table 1080  
 column of SYSRELS catalog table 1097  
 data type  
 CREATE TABLE statement 661  
 description 57  
 function 315  
**TIMESTAMP\_FORMAT** function 316  
**TIMESTAMP**  
 column of SYSCHECKS catalog table 1025  
**TNAME** column of SYSCOLAUTH catalog table 1027  
**TO**  
 clause of CONNECT (Type 1) statement 504  
 clause of CONNECT (Type 2) statement 510  
 clause of GRANT statement 804  
**TO SAVEPOINT** clause  
 ROLLBACK statement 894  
**TO\_CHAR** function 327  
**TO\_DATE** function 316  
 token in SQL 31  
**TPN** column of LOCATIONS catalog table 1017  
**TRACE** privilege  
 GRANT statement 823  
 REVOKE statement 888  
**TRACEAUTH** column of SYSUSERAUTH catalog  
 table 1138  
**TRACKMOD**  
 clause of ALTER TABLESPACE statement 476  
 clause of CREATE TABLESPACE statement 690  
 column of SYSTABLEPART catalog table 1123  
**TRANSLATE** built-in function 317  
**TRANSPROC** column of SYSSTRINGS catalog  
 table 1115  
**TRANSTAB** column of SYSSTRINGS catalog  
 table 1115  
**TRANSTYPE** column of SYSSTRINGS catalog  
 table 1115  
**TRIGEVENT** column of SYSTRIGGERS catalog  
 table 1136  
**trigger**  
 catalog information 1150  
 creating 699  
 description 9  
 dropping 770  
 naming convention 38  
**TRIGGER** clause  
 COMMENT statement 497  
 DROP statement 770  
**TRIGGER** privilege  
 GRANT statement 825  
 REVOKE statement 890  
**TRIGERAUTH** column of SYSTABAUTH catalog  
 table 1119  
**TRIGTIME** column of SYSTRIGGERS catalog  
 table 1136  
**TRUNC\_TIMESTAMP** function 321  
**TRUNCATE** function 320  
**truncation**  
 numbers 66  
**truth table** 145  
**truth valued logic** 145  
**TSNAME** column  
 SYSCOPY catalog table 1040  
 SYSTABLEPART catalog table 1121  
 SYSTABLEPART\_HIST catalog table 1124  
 SYSTABLES catalog table 1126  
 SYSTABLES\_HIST catalog table 1130  
 SYSTABSTATS catalog table 1134  
 SYSTABSTATS\_HIST catalog table 1135  
**TTNAME** column of SYSTABAUTH catalog table 1118  
**TYPE 2** clause of CREATE INDEX statement 603  
**TYPE** column  
 SYSCOLDIST catalog table 1028  
 SYSCOLDIST\_HIST catalog table 1029  
 SYSCOLDISTSTATS catalog table 1030  
 SYSDATABASE catalog table 1044  
 SYSPACKAGE catalog table 1077  
 SYTABCONST catalog table 1120  
 SYSTABLES catalog table 1126  
 SYTABLESPACE catalog table 1132  
 SYSVIEWS catalog table 1141  
 USERNAMES catalog table 1143  
 typed parameter marker 852  
**TYPENAME** column  
 SYSCOLUMNS catalog table 1036  
 SYSPARMS catalog table 1085  
**TYPESCHEMA** column  
 SYSCOLUMNS catalog table 1036  
 SYSPARMS catalog table 1085

## U

UCASE function 322  
 UCS-2 22

**UDF**  
 catalog information 1150  
 unary operation 114  
 unconnected state 19  
**Unicode**  
 definition 22  
 effect on DBCS characters 49  
**UNION clause**  
 duplicate rows 365  
 fullselect 365  
 result data type 77  
**UNIQUE clause**  
 ALTER TABLE statement 458  
 CREATE INDEX statement 603  
 CREATE TABLE statement 661, 668  
 SAVEPOINT statement 897  
**unique constraint** 6  
**unique index**  
 description 5  
**unique key** 5  
**UNIQUERULE column of SYSINDEXES catalog**  
 table 1054  
**unit of recovery**  
 COMMIT statement 499  
 description 11  
 ROLLBACK statement 894  
**unit of work**  
 closes cursors 844  
 description 12  
 dynamic caching 856  
 ending 12, 499, 894  
 initiating 12  
 persistence of prepared statements 856  
 referring to prepared statements 846  
**universal time, coordinated (UTC)** 84  
**unqualified object names** 39  
**untyped parameter marker** 852  
**UPDATE**  
 clause of TRIGGER statement 701  
 statement  
 description 928  
 example 935  
**UPDATE privilege**  
 GRANT statement 825  
 REVOKE statement 890  
**update rule** 7, 933  
**UPDATEAUTH column of SYSTABAUTH catalog**  
 table 1119  
**UPDATECOLS column of SYSTABAUTH catalog**  
 table 1118  
**UPDATES column of SYSCOLUMNS catalog**  
 table 1033  
**updating**  
 rows in a table 928  
**UPPER function** 322  
**USAGE privilege**  
 GRANT statement 809  
 REVOKE statement 874  
**USEAUTH column of SYSRESAUTH catalog**  
 table 1098  
**USER clause of SET PATH statement** 922

**USER special register** 92  
**user-defined function** 155  
 altering with ALTER FUNCTION statement 391, 408  
 creating with CREATE FUNCTION statement 529, 530, 553, 571, 585  
 dropping 766  
 privileges 811  
 revoking 876  
 statements allowed 972  
**user-defined function (UDF)**  
 description 105  
 external functions 105  
 inheriting special registers 93  
 invocation 107  
 MQSeries functions 105, 155  
 name, unqualified 40  
 naming convention 36  
 resolution 107  
 sample  
     ALTDATETIME 1156  
     ALTTIME 1159  
     CURRENCY 1161  
     DAYNAME 1163  
     MONTHNAME 1164  
     TABLE\_LOCATION 1165  
     TABLE\_NAME 1167  
     TABLE\_SCHEMA 1169  
     WEATHER 1171  
 sourced functions 105  
 table functions 105  
 unqualified name 40  
**USERNAMES column**  
 IPNAMES catalog table 1016  
 LUNAMES catalog table 1021  
**USING clause**  
 ALTER INDEX statement 419, 421  
 ALTER TABLESPACE statement 471  
 CREATE INDEX statement 606, 608  
 CREATE TABLESPACE statement 684, 687  
 DESCRIBE statement 750  
 EXECUTE statement 776  
 OPEN statement 842  
 PREPARE statement 847  
**USING DESCRIPTOR clause**  
 EXECUTE statement 776  
 OPEN statement 843  
**USING TYPE DEFAULTS clause**  
 DECLARE GLOBAL TEMPORARY TABLE  
 statement 730  
**UTC (universal time, coordinated)** 84  
**UTF-16** 22  
**UTF-8** 22

## V

**VALID column**  
 SYSPACKAGE catalog table 1073  
 SYSPLAN catalog table 1087  
**VALIDATE**  
 column of SYSPACKAGE catalog table 1074  
 column of SYSPLAN catalog table 1087

validation procedure 462  
 validation routine  
     VALIDPROC clause 462, 673  
 VALIDPROC clause  
     ALTER TABLE statement 462  
     CREATE TABLE statement 673  
 VALPROC column of SYSTABLES catalog table 1126  
 value  
     composite 5  
     SQL 48  
 VALUE function 77, 192, 224  
 VALUES clause  
     ALTER INDEX statement 423  
     CREATE INDEX statement 611  
     INSERT statement 834  
     VALUES INTO statement 939  
     VALUES statement 938  
 VALUES INTO statement  
     description 939  
     example 940  
 VALUES statement  
     description 938  
     example 938  
 VAR function 171  
 VAR\_SAMP function 172  
 VARCHAR  
     data type  
         CREATE TABLE statement 659  
         description 51  
         function 323  
 VARCHAR\_FORMAT function 327  
 VARGRAPHIC  
     data type  
         CREATE TABLE statement 660  
         description 52  
         function 328  
 variable  
     description 99  
     host  
         referencing 100  
         SQL syntax 100  
     referencing 99  
     SQL syntax 99  
 VARIABLE clause  
     DECLARE VARIABLE statement 739  
 VARIANCE function 171  
 VARIANCE\_SAMP function 172  
 VARIANT clause  
     ALTER FUNCTION statement 410  
     ALTER PROCEDURE statement 428, 439  
     CREATE FUNCTION statement 533, 556, 587  
     CREATE PROCEDURE statement 620, 639  
 VCAT  
     clause of CREATE STOGROUP statement 649  
 USING clause  
     ALTER INDEX statement 419  
     ALTER TABLESPACE statement 472  
     CREATE INDEX statement 421, 606, 608  
     CREATE TABLESPACE statement 684, 687  
 VCATNAME column  
     SYSINDEXPART catalog table 1058  
 VCATNAME column (*continued*)  
     SYSSTOGROUP catalog table 1114  
     SYSTABLEPART catalog table 1121  
 VERSION  
     clause of DROP statement 768  
     column of SYSDBRM catalog table 1050  
     column of SYSPACKAGE catalog table 1075  
     column of SYSPACKSTMT catalog table 1081  
 version identifier, current server 506, 513  
 version-id naming convention 39  
 view  
     creating  
         CREATE VIEW statement 711  
         description 10  
     dropping  
         description 770  
         name, unqualified 39  
         naming convention 39  
         privileges 889  
         granting 824  
         unqualified name 39  
     using  
         adding comments 1151  
         obtaining information with DESCRIBE 749  
         read-only 715  
         retrieving catalog information 1147  
         retrieving comments 1151  
 VIEW clause  
     CREATE VIEW statement 711  
     DROP statement 770  
 VOLID column of SYSVOLUMES catalog table 1142  
 VOLUMES clause of CREATE STOGROUP  
     statement 648  
 VSAM (virtual storage access method)  
     catalog 608

## W

WEATHER function 1171  
 WEEK function 331  
 WEEK\_ISO function 332  
 WHEN clause of TRIGGER statement 704  
 WHENEVER statement  
     description 941  
     example 942  
 WHERE clause  
     DELETE statement 744  
     description 358  
     search condition 358  
     UPDATE statement 932  
 WHILE statement  
     example 964  
     SQL procedure 964  
 WITH CHECK OPTION clause of CREATE VIEW  
     statement 714  
 WITH clause  
     DELETE statement 745  
     INSERT statement 835  
     SELECT INTO statement 901  
     select-statement 373

WITH GRANT OPTION clause of GRANT  
statement 804  
WITH HOLD clause of DECLARE CURSOR  
statement 720  
WITH HOLD clause of PREPARE statement 849  
WITH PROCEDURE clause of ASSOCIATE LOCATORS  
statement 479  
WITH RETURN clause of DECLARE CURSOR  
statement 721  
WITH RETURN clause of PREPARE statement 850  
WITH RRIRSICSIUR clause 901  
WLM ENVIRONMENT clause  
    ALTER FUNCTION statement 404  
    ALTER PROCEDURE statement 434, 440  
    CREATE FUNCTION statement 547, 567  
    CREATE PROCEDURE statement 629, 644  
WLM\_ENV column clause of SYSPROCEDURES  
catalog table 1095  
WLM\_ENV\_FOR\_NESTED column of SYSROUTINES  
catalog table 1103  
WLM\_ENVIRONMENT column of SYSROUTINES  
catalog table 1102  
work file database  
    creating 520  
WORKAREA column of SYSFIELDS catalog  
table 1052

## Y

YEAR function 333



---

## Readers' Comments — We'd Like to Hear from You

DB2 Universal Database for OS/390 and z/OS  
SQL Reference  
Version 7

Publication No. SC26-9944-05

**Overall, how satisfied are you with the information in this book?**

|                      | Very Satisfied           | Satisfied                | Neutral                  | Dissatisfied             | Very Dissatisfied        |
|----------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|
| Overall satisfaction | <input type="checkbox"/> |

**How satisfied are you that the information in this book is:**

|                          | Very Satisfied           | Satisfied                | Neutral                  | Dissatisfied             | Very Dissatisfied        |
|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|
| Accurate                 | <input type="checkbox"/> |
| Complete                 | <input type="checkbox"/> |
| Easy to find             | <input type="checkbox"/> |
| Easy to understand       | <input type="checkbox"/> |
| Well organized           | <input type="checkbox"/> |
| Applicable to your tasks | <input type="checkbox"/> |

**Please tell us how we can improve this book:**

Thank you for your responses. May we contact you?  Yes  No

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you.

---

Name

---

Address

---

Company or Organization

---

---

Phone No.

---

**Readers' Comments — We'd Like to Hear from You**  
SC26-9944-05



Cut or Fold  
Along Line

Fold and Tape

Please do not staple

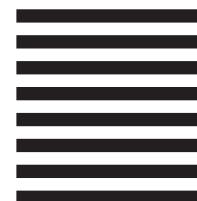
Fold and Tape



---

NO POSTAGE  
NECESSARY  
IF MAILED IN THE  
UNITED STATES

---



## BUSINESS REPLY MAIL

FIRST-CLASS MAIL PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

International Business Machines Corporation  
Reader Comments  
DTX/E269  
555 Bailey Avenue  
San Jose, CA 95141-9989  
U. S. A.



Fold and Tape

Please do not staple

Fold and Tape

SC26-9944-05

Cut or Fold  
Along Line





Program Number: 5675-DB2

Printed in USA

SC26-9944-05

