

DB2 for OS/390
Version 5



SQL Reference

Note!

Before using this information and the product it supports, be sure to read the general information under “Notices” on page ix.

First Edition (June 1997)

This edition applies to Version 5 of IBM DATABASE 2 Server for OS/390 (DB2 for OS/390), 5655-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.

The technical changes for this edition are summarized under “Summary of Changes to this Book” in the Introduction. Specific changes are indicated by a vertical bar to the left of a change. A vertical bar to the left of a figure caption indicates that the figure has changed. Editorial changes that have no technical significance are not noted.

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 manual since the hardcopy manual was published are indicated by the hash (#) symbol in the left-hand margin.

© Copyright International Business Machines Corporation 1982, 1997. All rights reserved.

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

Contents

Notices	ix
Programming Interface Information	ix
Trademarks	x
Chapter 1. Introduction	1
Who Should Read This Book	3
How to Use This Book	3
SQL Standards	3
How to Read the Syntax Diagrams	4
Conventions for Describing Mixed Data Values	5
How to Use the DB2 Library	5
How to Obtain DB2 Information	7
Summary of Changes to DB2 for OS/390 Version 5	9
Summary of Changes to This Book	16
Chapter 2. DB2 Concepts	19
Structured Query Language	21
Tables	22
Indexes	23
Keys	23
Referential Integrity	24
Check Constraints	26
Storage Structures	26
Storage Groups	27
Databases	27
Catalog	27
Views	27
Application Processes, Concurrency, and Recovery	28
Packages and Application Plans	30
Distributed Data	31
Character Conversion	37
Chapter 3. Language Elements	43
Characters	45
Tokens	45
Identifiers	46
Naming Conventions	48
Aliases and Synonyms	51
Authorization IDs and Authorization-names	52
Data Types	57
Assignment and Comparison	65
Constants	74
Special Registers	78
Column Names	84
Referencing Host Variables	89
Host Structures in PL/I, C, and COBOL	90
Expressions	92
Predicates	106
Search Conditions	118
Options Affecting SQL	119

Chapter 4. Functions	127
Column Functions	130
Scalar Functions	136
Chapter 5. Queries	167
Authorization	169
subselect	170
fullselect	183
select-statement	188
Chapter 6. Statements	193
How SQL Statements Are Invoked	195
ALLOCATE CURSOR	200
ALTER DATABASE	202
ALTER INDEX	205
ALTER STOGROUP	214
ALTER TABLE	217
ALTER TABLESPACE	233
ASSOCIATE LOCATORS	243
BEGIN DECLARE SECTION	246
CALL	248
CLOSE	253
COMMENT ON	255
COMMIT	257
CONNECT	259
CONNECT (Type 1)	262
CONNECT (Type 2)	267
CREATE ALIAS	270
CREATE DATABASE	272
CREATE GLOBAL TEMPORARY TABLE	275
CREATE INDEX	280
CREATE PROCEDURE (SQL procedure)	295
CREATE STOGROUP	303
CREATE SYNONYM	306
CREATE TABLE	308
CREATE TABLESPACE	327
CREATE VIEW	341
DECLARE CURSOR	347
DECLARE STATEMENT	352
DECLARE TABLE	354
DELETE	357
DESCRIBE	362
DESCRIBE CURSOR	368
DESCRIBE INPUT	370
DESCRIBE PROCEDURE	372
DROP	375
END DECLARE SECTION	380
EXECUTE	382
EXECUTE IMMEDIATE	386
EXPLAIN	388
FETCH	397
GRANT	400
GRANT (Collection Privileges)	403
GRANT (Database Privileges)	404

	GRANT (Package Privileges)	406
	GRANT (Plan Privileges)	408
	GRANT (System Privileges)	409
	GRANT (Table or View Privileges)	412
	GRANT (Use Privileges)	415
	INCLUDE	417
	INSERT	419
	LABEL ON	424
	LOCK TABLE	426
	OPEN	428
	PREPARE	433
	RELEASE	437
	RENAME	440
	REVOKE	443
	REVOKE (Collection Privileges)	447
	REVOKE (Database Privileges)	448
	REVOKE (Package Privileges)	450
	REVOKE (Plan Privileges)	452
	REVOKE (System Privileges)	453
	REVOKE (Table or View Privileges)	456
	REVOKE (Use Privileges)	458
	ROLLBACK	460
	SELECT INTO	462
	SET CONNECTION	465
	SET CURRENT DEGREE	468
#	SET CURRENT PACKAGESET	470
	SET CURRENT PRECISION	472
	SET CURRENT RULES	473
	SET CURRENT SQLID	474
	SET host-variable	476
	UPDATE	477
	WHENEVER	483
#	Chapter 7. SQL procedure statements	485
#	Procedure body	486
#	Assignment statement	487
#	CASE statement	489
#	Compound statement	491
#	IF statement	496
#	GET DIAGNOSTICS statement	497
#	GOTO statement	498
#	LEAVE statement	500
#	LOOP statement	501
#	REPEAT statement	502
#	WHILE statement	503
#	SQL procedure statement	504
	Appendix A. Limits in DB2 for OS/390	505
	Appendix B. Characteristics of SQL Statements in DB2 for OS/390	509
	Actions Allowed on SQL Statements	509
#	SQL statements allowed in SQL procedures	511
	Appendix C. SQLCA and SQLDA	513

SQL Communication Area (SQLCA)	513
SQL Descriptor Area (SQLDA)	519
Appendix D. DB2 Catalog Tables	529
Table Spaces and Indexes	529
New and Changed Catalog Tables	534
SYSIBM.IPNAMES Table	537
SYSIBM.LOCATIONS Table	538
SYSIBM.LULIST Table	539
SYSIBM.LUMODES Table	540
SYSIBM.LUNAMES Table	541
SYSIBM.MODESELECT Table	543
SYSIBM.SYSCHECKDEP Table	544
SYSIBM.SYSCHECKS Table	545
SYSIBM.SYSCOLAUTH Table	546
SYSIBM.SYSCOLDIST Table	547
SYSIBM.SYSCOLDISTSTATS Table	548
SYSIBM.SYSCOLSTATS Table	549
SYSIBM.SYSCOLUMNS Table	550
SYSIBM.SYSCOPY Table	554
SYSIBM.SYSDATABASE Table	557
SYSIBM.SYSDBAUTH Table	559
SYSIBM.SYSDBRM Table	562
SYSIBM.SYSDUMMY1 Table	564
SYSIBM.SYSFIELDS Table	565
SYSIBM.SYSFOREIGNKEYS Table	566
SYSIBM.SYSINDEXES Table	567
SYSIBM.SYSINDEXPART Table	570
SYSIBM.SYSINDEXSTATS Table	572
SYSIBM.SYSKEYS Table	573
SYSIBM.SYSPACKAGE Table	574
SYSIBM.SYSPACKAUTH Table	578
SYSIBM.SYSPACKDEP Table	579
SYSIBM.SYSPACKLIST Table	580
SYSIBM.SYSPACKSTMT Table	581
SYSIBM.SYSPKSYSTEM Table	583
SYSIBM.SYSPLAN Table	584
SYSIBM.SYSPLANAUTH Table	587
SYSIBM.SYSPLANDEP Table	588
SYSIBM.SYSPLSYSTEM Table	589
SYSIBM.SYSPROCEDURES Table	590
SYSIBM.SYSRELS Table	593
SYSIBM.SYSRESAUTH Table	594
SYSIBM.SYSSTMT Table	595
SYSIBM.SYSSTOGROUP Table	597
SYSIBM.SYSSTRINGS Table	598
SYSIBM.SYSSYNONYMS Table	599
SYSIBM.SYSTABAUTH Table	600
SYSIBM.SYSTABLEPART Table	603
SYSIBM.SYSTABLES Table	606
SYSIBM.SYSTABLESPACE Table	610
SYSIBM.SYSTABSTATS Table	613
SYSIBM.SYSUSERAUTH Table	614
SYSIBM.SYSVIEWDEP Table	617

SYSIBM.SYSVIEWS Table	618
SYSIBM.SYSVOLUMES Table	619
SYSIBM.USERNAMES Table	620
Appendix E. SQL Reserved Words	621
Appendix F. DB2 Objects Required by the DB2 for OS/390 SQL	
Procedure Processor	625
Table Spaces and Indexes	625
The SQL Procedure Source Table (SYSIBM.SYSPSM)	625
The SQL Procedure Options Table (SYSIBM.SYSPSMOPTS)	626
Temporary Table SYSIBM.SYSPSMOUT	627
Glossary	629
Bibliography	643
Index	649

Contents

Notices

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. 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. Subject to IBM's valid intellectual property or other legally protectable rights, any functionally equivalent product, program, or service may be used instead of the IBM product, program, or service. The evaluation and verification of operation in conjunction with other products, except those expressly designated by IBM, are the responsibility of the user.

IBM may have patents or pending patent applications covering subject matter 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.

Licensees of this program who wish to have information about it for the purpose of
enabling (1) the exchange of information between independently created programs
and other programs (including this one) and (2) the mutual use of the information
that has been exchanged, should contact:

IBM Corporation
IBM Corporation
J74/G4
555 Bailey Avenue
P.O. Box 49023
San Jose, CA 95161-9023

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

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 Server for OS/390 (DB2 for OS/390).

General-use programming interfaces allow the customer to write programs that obtain the services of DB2 for OS/390.

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 in order 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 IBM Corporation in the United States or other countries or both:

AD/Cycle	DRDA
AIX	IBM
APL2	IMS
AS/400	IMS/ESA
BookManager	Language Environment
C/370	MVS/ESA
CICS	MVS/XA
CICS/ESA	OS/2
CICS/MVS	OS/390
COBOL/370	OS/400
DATABASE 2	QMF
DataPropagator	RACF
DB2	RRDF
DFSMS	SAA
DFSMS/MVS	SQL/DS
DFSMSdfp	System/370
DFSMSHsm	System/390
Distributed Relational Database Architecture	Systems Application Architecture
DProp	VTAM

Throughout the library, the DB2 for OS/390 licensed program and a particular DB2 for OS/390 subsystem are each referred to as “DB2.” In each case, the context makes the meaning clear.

The term *MVS* is used to represent the MVS/Enterprise Systems Architecture (MVS/ESA). *CICS* is used to represent CICS/MVS and CICS/ESA; *IMS* is used to represent IMS/ESA; *COBOL* is used to represent OS/VS COBOL, VS COBOL II, IBM COBOL for MVS & VM (formerly called COBOL/370); unless noted otherwise, *C* and *C language* are used to represent C/370 and C/C++ for MVS/ESA programming languages.

Other company, product, and service names, which may be denoted by a double asterisk (**), may be trademarks or service marks of others.

Chapter 1. Introduction

Who Should Read This Book	3
How to Use This Book	3
SQL Standards	3
How to Read the Syntax Diagrams	4
Conventions for Describing Mixed Data Values	5
How to Use the DB2 Library	5
How to Obtain DB2 Information	7
DB2 on the Web	8
DB2 Publications	8
How to Order the DB2 Library	9
Summary of Changes to DB2 for OS/390 Version 5	9
Server Solution	9
Performance	11
Increased Capacity	12
Improved Availability	12
Client/Server and Open Systems	13
User Productivity	15
Summary of Changes to This Book	16

Who Should Read This Book

This book serves as a reference for Structured Query Language (SQL) for DB2 for OS/390. It 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 SQL programming concepts.

Unless otherwise stated, references to SQL in this book imply SQL for DB2 for OS/390, and all objects described in this book are objects of DB2 for OS/390. 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.

How to Use This Book

This book has the following sections:

- “Chapter 1. Introduction” on page 1 identifies the purpose, the audience, and the use of the book.
- “Chapter 2. DB2 Concepts” on page 19 describes the basic concepts of relational databases and SQL.
- “Chapter 3. Language Elements” on page 43 describes the basic syntax of SQL and the language elements that are common to many SQL statements.
- “Chapter 4. Functions” on page 127 contains syntax diagrams, semantic descriptions, rules, and use examples of SQL column and scalar functions.
- “Chapter 5. Queries” on page 167 describes the various forms of a query, which is a component of various SQL statements.
- “Chapter 6. Statements” on page 193 contains syntax diagrams, semantic descriptions, rules, and examples of all SQL statements.
- “Chapter 7. SQL procedure statements” on page 485 contains syntax diagrams, semantic descriptions, rules, and examples of SQL procedure statements.
- The appendixes contain information about DB2 limits, SQLCA, SQLDA, catalog tables, and SQL reserved words.

#

When you first use this book, consider reading Chapters 2 and 3 sequentially. The rest of the book is designed for the quick location of answers to specific SQL questions.

SQL Standards

In this book, the use of the term *SQL standard* refers collectively to:

- FIPS (Federal Information Processing Standards) publication 127-2, Database Language SQL, which announces ANSI X3.135-1992 as the standard for SQL
- ANSI (American National Standards Institute) X3.135-1992, Database Language SQL

- ISO (International Standards Organization) 9075-1992, Database Language SQL

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 \blacktriangleright — symbol indicates the beginning of a statement.

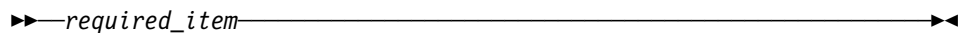
The — \blacktriangleright symbol indicates that the statement syntax is continued on the next line.

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

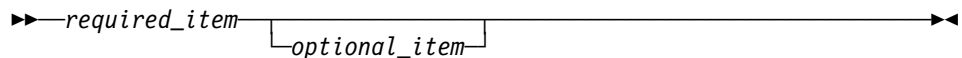
The — \blacktriangleleft symbol indicates the end of a statement.

Diagrams of syntactical units other than complete statements start with the \blacktriangleright — symbol and end with the — \blacktriangleright symbol.

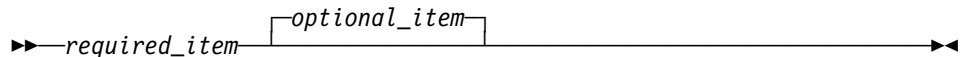
- Required items appear on the horizontal line (the main path).



- Optional items appear below the main path.

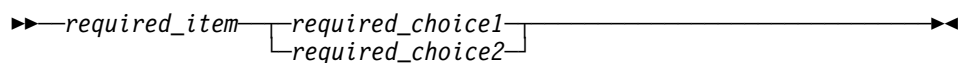


If an optional item appears above the main path, that item has no effect on the execution of the statement and is used only for readability.

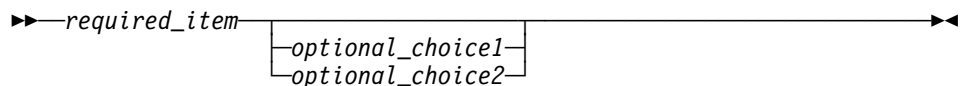


- If you can choose from two or more items, they appear vertically, in a stack.

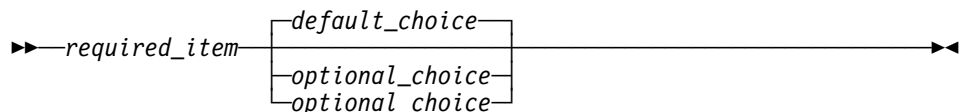
If you *must* choose one of the items, one item of the stack appears on the main path.



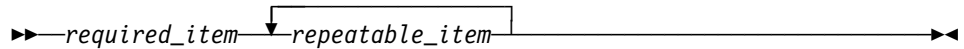
If choosing one of the items is optional, the entire stack appears below the main path.



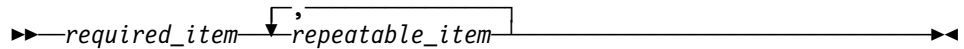
If one of the items is the default, it appears above the main path and the remaining choices are shown below.



- 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.

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
$\$0$	“shift-out” control character (X'0E'), used only for EBCDIC data
$\$I$	“shift-in” control character (X'0F'), used only for EBCDIC data
sbcs-string	SBCS string of zero or more single-byte characters
dbcs-string	DBCS string of zero or more double-byte characters
'	DBCS apostrophe
G	DBCS uppercase G

How to Use the DB2 Library

Titles of books in the library begin with DB2 for OS/390 Version 5. However, references from one book in the library to another are shortened and do not include the product name, version, and release. Instead, they point directly to the section that holds the information. For a complete list of books in the library, and the sections in each book, see the bibliography at the back of this book.

Throughout the library, the DB2 for OS/390 licensed program and a particular DB2 for MVS/ESA subsystem are each referred to as “DB2.” In each case, the context makes the meaning clear.

The most rewarding task associated with a database management system is asking questions of it and getting answers, the task called *end use*. Other tasks are also necessary—defining the parameters of the system, putting the data in place, and so on. The tasks associated with DB2 are grouped into the following major categories (but supplemental information relating to all of the below tasks for new releases of DB2 can be found in *Release Guide*):

Installation: If you are involved with DB2 only to install the system, *Installation Guide* might be all you need.

If you will be using data sharing then you also need *Data Sharing: Planning and Administration*, which describes installation considerations for data sharing.

End use: End users issue SQL statements to retrieve data. They can also insert, update, or delete data, with SQL statements. They might need an introduction to SQL, detailed instructions for using SPUFI, and an alphabetized reference to the types of SQL statements. This information is found in *Application Programming and SQL Guide* and this book.

End users can also issue SQL statements through the Query Management Facility (QMF) or some other program, and the library for that program might provide all the instruction or reference material they need. For a list of some of the titles in the QMF library, see the bibliography at the end of this book.

Application Programming: Some users access DB2 without knowing it, using programs that contain SQL statements. DB2 application programmers write those programs. Because they write SQL statements, they need *Application Programming and SQL Guide*, *SQL Reference*, and *Call Level Interface Guide and Reference* just as end users do.

Application programmers also need instructions on many other topics:

- How to transfer data between DB2 and a host program—written in COBOL, C, or FORTRAN, for example
- How to prepare to compile a program that embeds SQL statements
- How to process data from two systems simultaneously, say DB2 and IMS or DB2 and CICS
- How to write distributed applications across platforms
- How to write applications that use DB2 Call Level Interface to access DB2 servers
- How to write applications that use Open Database Connectivity (ODBC) to access DB2 servers
- How to write applications in the Java programming language to access DB2 servers

The material needed for writing a host program containing SQL is in *Application Programming and SQL Guide* and *Application Programming Guide and Reference for Java*. The material needed for writing applications that use DB2 Call Level Interface or ODBC to access DB2 servers is in *Call Level Interface Guide and Reference*.

For handling errors, see *Messages and Codes*.

Information about writing applications across platforms can be found in *Distributed Relational Database Architecture: Application Programming Guide*.

System and Database Administration: *Administration* covers almost everything else. *Administration Guide* divides those tasks among the following sections:

- Section 2 (Volume 1) of *Administration Guide* discusses the decisions that must be made when designing a database and tells how to bring the design into being by creating DB2 objects, loading data, and adjusting to changes.
- Section 3 (Volume 1) of *Administration Guide* describes ways of controlling access to the DB2 system and to data within DB2, to audit aspects of DB2 usage, and to answer other security and auditing concerns.
- Section 4 (Volume 1) of *Administration Guide* describes the steps in normal day-to-day operation and discusses the steps one should take to prepare for recovery in the event of some failure.
- Section 5 (Volume 2) of *Administration Guide* explains how to monitor the performance of the DB2 system and its parts. It also lists things that can be done to make some parts run faster.

In addition, the appendixes in *Administration Guide* contain valuable information on DB2 sample tables, National Language Support (NLS), writing exit routines, interpreting DB2 trace output, and character conversion for distributed data.

If you are involved with DB2 only to design the database, or plan operational procedures, you need *Administration Guide*. If you also want to carry out your own plans by creating DB2 objects, granting privileges, running utility jobs, and so on, then you also need:

- *SQL Reference*, which describes the SQL statements you use to create, alter, and drop objects and grant and revoke privileges
- *Utility Guide and Reference*, which explains how to run utilities
- *Command Reference*, which explains how to run commands

If you will be using data sharing, then you need *Data Sharing: Planning and Administration*, which describes how to plan for and implement data sharing.

Additional information about system and database administration can be found in *Messages and Codes*, which lists messages and codes issued by DB2, with explanations and suggested responses.

Diagnosis: Diagnosticians detect and describe errors in the DB2 program. They might also recommend or apply a remedy. The documentation for this task is in *Diagnosis Guide and Reference* and *Messages and Codes*.

How to Obtain DB2 Information

DB2 on the Web

Stay current with the latest information about DB2. View the DB2 home page on the World Wide Web. News items keep you informed about the latest enhancements to the product. Product announcements, press releases, fact sheets, and technical articles help you plan your database management strategy. Technical professionals can access DB2 publications on the Web and follow links to other Web sites with more information about DB2 family and OS/390 solutions. Access DB2 on the Web with the following URL:

<http://www.ibm.com/software/db2os390>

DB2 Publications

The DB2 publications are available in both hardcopy and softcopy format. Using online books on CD-ROM, you can read, search across books, print portions of the text, and make notes in these BookManager books. With the appropriate BookManager READ product or IBM Library Readers, you can view these books on the MVS, VM, OS/2, DOS, AIX and Windows platforms.

When you order DB2 Version 5, you are entitled to one copy of the following CD-ROM, which contains the DB2 licensed book for no additional charge:

DB2 Server for OS/390 Version 5 Licensed Online Book, LK2T-9075.

You can order multiple copies for an additional charge by specifying feature code 8207.

When you order DB2 Version 5, you are entitled to one copy of the following CD-ROM, which contains the DB2 and DATABASE 2 Performance Monitor online books for no additional charge:

DB2 Server for OS/390 Version 5 Online Library, SK2T-9092

You can order multiple copies for an additional charge through IBM's publication ordering service.

Periodic updates will be provided on the following collection kit available to licensees of DB2 Version 5:

IBM Online Library Transaction Processing and Data Collection, SK2T-0730

SK2T-9092 will be superseded by SK2T-0730 when updates to the online library are available.

In some countries, including the United States and Canada, you receive one copy of the collection kit at no additional charge when you order DB2 Version 5. You will automatically receive one copy of the collection kit each time it is updated, for no additional charge. To order multiple copies of SK2T-0730 for an additional charge, see "How to Order the DB2 Library" on page 9. In other countries, updates will be available in displayable softcopy format in the IBM Online Book Library Offering (5636-PUB), SK2T-0730 IBM Online Library Transaction Processing and Data Collection at a later date.

See your IBM representative for assistance in ordering the collection.

DB2 Server for OS/390 books are also available for an additional charge on the following collection kits, which contain online books for many IBM products:

IBM Online Library MVS Collection, SK2T-0710, in English

Online Library Omnibus Edition OS/390 Collection, SK2T-6700, in English

IBM Online Library MVS Collection Kit, SK88-8002, in Japanese, for viewing on DOS and Windows platforms

How to Order the DB2 Library

You can order DB2 publications and CD-ROMs through your IBM representative or the IBM branch office serving your locality. If you are located within the United States or Canada, you can place your order by calling one of the toll-free numbers :

- In the U.S., call 1-800-879-2755.
- In Canada, call 1-800-565-1234.

To order additional copies of licensed publications, specify the SOFTWARE option. To order additional publications or CD-ROMs, specify the PUBLICATIONS & SLSS option. Be prepared to give your customer number, the product number, and the feature code(s) or order numbers you want.

Summary of Changes to DB2 for OS/390 Version 5

DB2 for OS/390 Version 5 delivers a database server solution for OS/390. Version 5 supports all functions available in DB2 for MVS/ESA Version 4 plus enhancements in the areas of performance, capacity, and availability, client/server and open systems, and user productivity.

If you are currently using DB2, **you can migrate only from a DB2 for MVS/ESA Version 4 subsystem**. This summary gives you an overview of the differences to be found between these versions.

Server Solution

OS/390 retains the classic strengths of the traditional MVS/ESA operating system, while offering a network-ready, integrated operational environment.

The following features work directly with DB2 for OS/390 applications to help you use the full potential of your DB2 subsystem:

- Net.Data for OS/390
- DB2 Installer
- DB2 Estimator for Windows
- DB2 Visual Explain
- Workstation-based Performance Analysis and Tuning
- DATABASE 2 Performance Monitor

Net.Data for OS/390

Net.Data provides support for Internet access to DB2 data through a Web server. Applications built with Net.Data make data stored in any DB2 server more accessible and useful. Net.Data Web applications provide continuous application availability, scalability, security, and high performance.

This no charge feature can be ordered with DB2 Version 5 or downloaded from Internet. The Net.Data URL is:

#

<http://www.ibm.com/software/data/net.data/downloads.html>

DB2 Installer

DB2 Installer offers the option to install DB2 on an OS/2 workstation. Now, you can use a friendly graphical interface to complete installation tasks easily with DB2 Installer.

This function is delivered on CD-ROM with DB2 Visual Explain.

DB2 Estimator for Windows

DB2 Estimator provides an easy-to-use capacity planning tool. You can estimate the sizes of tables and indexes, and the performance of SQL statements, groups of SQL statements (transactions), utility runs, and groups of transactions (capacity runs). From a simple table sizing to a detailed performance analysis of an entire DB2 application, DB2 Estimator saves time and lowers costs. You can investigate the impact of new or modified applications on your production system, *before* you implement them.

This no charge feature can be ordered with DB2 Version 5 or downloaded from the Internet. From the internet, use the IBM Software URL:

<http://www.ibm.com/software/>

From here, you can access information about DB2 Estimator using the download function.

DB2 Visual Explain

DB2 Visual Explain lets you tune DB2 SQL statements on an OS/2 workstation. You can see DB2 EXPLAIN output in a friendly graphical interface and easily access, modify, and analyze applications with DB2 Visual Explain.

Workstation-based Performance Analysis and Tuning

The new workstation-based Performance Analysis and Tuning function simplifies system administration. You can access statistical data to help you analyze and improve system performance. This function works with the optional DB2 PM feature to provide full analysis and tuning functionality.

DATABASE 2 Performance Monitor (DB2 PM)

DB2 PM lets you monitor, analyze, and optimize the performance of DB2 Version 5 and its applications. An online monitor, for both host and workstation environments, provides an immediate "snap-shot" view of DB2 activities and allows for exception processing while the system is operational. The workstation-based online monitor can connect directly to the Visual Explain function of the DB2 base product.

DB2 PM also offers a history facility, a wide variety of customizable reports for in-depth performance analysis, and an EXPLAIN function to analyze and optimize SQL statements. For more information, see *DB2 PM for OS/390 General Information* .

This feature can be ordered with DB2 Version 5.

Performance

Sysplex Query Parallelism

The increased power of Sysplex query parallelism in DB2 for OS/390 Version 5 allows DB2 to go far beyond DB2 for MVS/ESA Version 4 capabilities; from the ability to split and process a single query within a DB2 subsystem to processing that same query across many different DB2 subsystems in a data sharing group.

The advances this release offers in scalable query processing let you process queries quickly while accommodating the potential growth of data sharing groups and the increasing complexity of queries.

Prepared Statement Caching

DB2 reduces the cost of duplicate prepares for the same dynamic SQL statement by saving them in a cache. Now, different application processes can share prepared statements and they are preserved past the commit point. This performance improvement offers the most benefit for:

- Client/server applications that frequently use dynamic SQL for repeated execution of SQL statements
- Relatively short dynamic SQL statements for which PREPARE cost accounts for most of the CPU expended

Reoptimization

When host variables, parameter markers, or special registers were used in previous releases, DB2 could not always determine the best access path because the values for these variables were unknown. Now, you can tell DB2 to reevaluate the access path at run time, after these values are known. As a result, queries can be processed more efficiently, and response time is improved.

Faster Transactions and Batch

- Caching of package authorization improves performance at run time for remote packages and applications that use pattern-matching characters in a package list.
- You can define a table space to use **selective partition locking**, which can reduce locking costs for applications that do partition-at-a-time processing. It also can reduce locking costs for certain data sharing applications that rely on an affinity between members and data partitions.
- A new standalone utility lets you preformat active logs.
- With LOAD and REORG, you can preformat data sets up to the high allocated RBA, which can make processing for sequential inserts more predictable.

Faster Utilities

- LOAD and REORG jobs run faster and more efficiently with enhanced index key sorting that reduces CPU and elapsed time, and an inline copy feature that lets you make an image copy without a separate copy step.
- New REORG options let you select rows to discard during a REORG and, optionally, write the discarded records to a file.
- When you run the REBUILD, RECOVER, REORG, or LOAD utility on DB2-managed indexes or table spaces, a new option lets you logically reset and reuse the DB2-managed objects.

#

- RECOVER INDEX and LOAD run faster on large numbers of rows per page.
- Sampling support for RUNSTATS reduces the processing required to collect nonindexed column statistics.
- BSAM striping improves the I/O capability of DB2 utilities.

Other Performance Enhancements

- There are several significant performance enhancements to data sharing, including selective partition locking, the MAXROWS option, and several optimizations to reduce data sharing overhead.
- DB2 installations that run in the OS/390 Version 2 Release 6 environment can now have as many as (approximately) 25 000 open DB2 data sets at one time. The maximum number of open data sets in earlier releases of OS/390 is 10000.
- You can easily alter the length of variable-length character columns using the new ALTER COLUMN clause of the ALTER TABLE statement.
- SQL CASE expressions let you eliminate queries with multiple UNIONS and improve performance by using only one table scan.
- You can collect a new statistic on concatenated index keys to improve the performance of queries with correlated columns. The statistic lets DB2 estimate the number of rows that qualify for the query more accurately, and select access paths more efficiently.
- DB2 scans partitions more efficiently and allows scans during parallel processing.
- Query enhancements include the ability to:
 - Use indexes for joins on string columns that have different lengths
 - Use an index to access predicates with noncorrelated IN subqueries
- Noncolumn expressions in simple predicates are evaluated at stage 1 and can be indexable.

Increased Capacity

DB2 for OS/390 Version 5 introduces the concept of a *large* partitioned table space. Defining your table space as large allows a substantial capacity increase: to approximately one terabyte of data and up to 254 partitions. In addition to accommodating growth potential, large partitioned table spaces make database design more flexible, and can improve availability.

Improved Availability

Online REORG

DB2 for OS/390 Version 5 adds a major improvement to availability with *Online REORG*. Now, you can avoid the severe availability problems that occurred while offline reorganization of table spaces restricted access to read only during the unload phase and no access during reload phase of the REORG utility. Online REORG gives you full read and write access to your data through most phases of the process with only very brief periods of read only or no access.

Data Sharing Enhancements

- Version 5 provides continuous availability with group buffer pool duplexing. Prior releases of DB2 rely on DASD and the merged recovery logs to recover group buffer pool (GBP) data that is lost if a coupling facility fails. With group buffer pool duplexing, DB2 writes changed pages to both a *primary GBP* and a *secondary GBP*. Overlapped writes to the GBPs provide good performance and eliminate the writes to DASD.
- Group buffer pool rebuild makes coupling facility maintenance easier and improves access to the group buffer pool during connectivity losses.
- Automatic group buffer pool recovery accelerates GBP recovery time, eliminates operator intervention, and makes data available faster when GBPs are lost because of coupling facility failures.
- Improved restart performance for members of a data sharing group reduces the impact of retained locks by making data available faster when a group member fails.
- Changes to traces and DISPLAY GROUPBUFFERPOOL output improve monitoring.

Tracker site for disaster recovery

You can set up a tracker site that shadows the activity of a primary site, and eliminate the need to constantly ship image copies.

Client/Server and Open Systems

Native TCP/IP Network Support

DB2's support of TCP/IP networks allows DRDA clients to connect directly to DDF and eliminate the gateway machine. In addition, customers can now use asynchronous transfer mode (ATM) as the underlying communication protocol for both SNA and TCP/IP connections to DB2.

Stored Procedures

- Return multiple SQL result sets to local and remote clients in a single network operation.
- Receive calls from applications that use standard interfaces, such as Open Database Connectivity** (ODBC) and X/Open** Call Level Interface, to access data in DB2 for OS/390.
- Run in an enhanced environment. DB2 supports multiple stored procedures address spaces managed by the MVS Workload Manager (WLM). The WLM environment offers efficient program management and allows WLM-managed stored procedures to run as subprograms and use RACF security.
- Use individual MVS dispatching priorities to improve stored procedure scheduling.
- Access data sources outside DB2 with two-phase commit coordination.
- Use an automatic COMMIT feature on return to the caller that reduces network traffic and the length of time locks are held.
- Have the ability to invoke utilities, which means you can now invoke utilities from an application that uses the SQL CALL statement.

- #
- #
- Support IMS Open Database Access (ODBA). Now a DB2 stored procedure can directly connect to IMS DBCTL and access IMS data.

Dynamic Query and Network Performance

Improvements for DRDA Applications

- Reduced processing costs for block fetch operations
- DRDA support for OPTIMIZE FOR n ROWS on SELECT
- Faster dynamic SQL queries and reduced processing costs for VTAM network operations
- Reduced message traffic for dynamic SQL SELECT statements

Improved Application Portability

- DB2 for OS/390 Version 5 introduces the DB2 Call Level Interface (CLI) to MVS/ESA. Unlike applications that use embedded SQL to access DB2 data, applications that choose CLI are not tied to a precompiler, packages, or a plan.

Workstation and desktop applications use standard interfaces, such as Open Database Connectivity (ODBC), to access relational data. Standard interfaces need one version of an application to access many data sources. Now, you can port UNIX workstation and PC desktop applications to DB2 for OS/390 and exploit the CLI (ODBC) capabilities without modification. In addition, applications can issue ODBC or CLI calls from within a stored procedure.

- #
- #
- #
- #
- #
- #
- You can now access DB2 for OS/390 databases in your Java applications. You can use DB2 Connect Java Database Connectivity (JDBC) for your dynamic SQL applications, or SQLJ for your static SQL applications.
- DB2 adds DRDA support for the DESCRIBE INPUT statement to improve performance for many ODBC applications.
- Now, you can write multithreaded DB2 CLI applications, and restrictions on connection switching no longer exist.
- DB2 now provides ASCII table support for clients and servers across platforms. This support reduces the cost of translation between EBCDIC and ASCII encoding schemes. ASCII table support also offers an alternative to writing field procedures that provide the ASCII sort sequence, which improves performance.

Improved Security

- DB2 for OS/390 supports Distributed Computing Environment (DCE) for authenticating remote DRDA clients. DCE offers the following benefits:
 - Network security: By providing an encrypted DCE ticket for authentication, remote clients do not need to send an MVS password in readable text.
 - Simplified security administration: End users do not need to maintain a valid password on MVS to access DB2; instead, they maintain their DCE password only.
- New descriptive error codes help you determine the cause of network security errors.
- You can change end user MVS passwords from DRDA clients.

User Productivity

Improved SQL Compatibility

DB2 conforms to the ANSI/ISO SQL entry level standard of 1992. Application programmers can take advantage of a more complete set of standard SQL to use across the DB2 family to write portable applications. New SQL function includes:

- More check options for view definitions.
- Foreign keys that reference UNIQUE keys as well as PRIMARY keys.
- An extension to GRANT that lets the REFERENCES privilege apply to a list of columns.
- A new delete rule, NO ACTION, that you can use to define referential constraints for self-referencing tables.
- SQL CASE expressions provide the capability to create conditional logic wherever an expression is allowed.
- SQL temporary tables allow application programs to easily create and use temporary tables that store results of SQL transactions without logging or recovery.

New Access Choice

A new attachment facility, the Recoverable Resource Manager Services attachment facility, improves access in a client/server environment. It coordinates two-phase commit processing between DB2 and other participating resource managers in any MVS application environment. Other key features include the ability for multiple users to run in a single address space, thread reuse, and moving threads between MVS tasks.

Image Copy Enhancements

The COPY, LOAD, and REORG utilities provide:

- Features of the COPY utility that help you quickly determine what type of image copy to take, when to take it, and let DB2 automatically take it for you.
- Inline copy in LOAD and REORG that lets you create an image copy while improving data availability.

Improved Integration of C++ and IBM COBOL for MVS & VM Support

It is easier for application programmers to use object-oriented programming techniques in their DB2 applications. DB2 for OS/390 Version 5 adds COBOL and C++ languages as options on installation panels, DB2I panels, the DSNH command, and DCLGEN.

Other Usability Enhancements

- To prevent long running units of work and to help avoid unnecessary work during the recovery phase of restart, DB2 issues new warning messages at an interval of your choice.
- A new special register for decimal precision provides better granularity, so that applications that need different values for decimal precision can run in the same DB2 subsystem.

#

- Trace records for IFCID 0022 now include most information in the PLAN_TABLE.
- An increase from 127 to 255 rows on a page improves table space processing and eliminates the need for compression.
- Install SYSOPR can recover objects using the START DATABASE command.
- A filtering capability for DISPLAY BUFFERPOOL limits statistics information to a specified set of page sets.
- You can enter comments within the SYSIN input stream for DB2 utilities.

Summary of Changes to This Book

Listed below are the major changes to this edition of the book:

Refer to Page

Stored procedures:

The following statements are added to support stored procedures enhancements:

ALLOCATE CURSOR	200
ASSOCIATE LOCATORS	243
CREATE PROCEDURE (SQL procedure)	295
DESCRIBE CURSOR	368
DESCRIBE PROCEDURE	372

SQL procedures:

Chapter 7. SQL procedure statements is added to describe the statements that can be used in an SQL procedure. 485

Increased capacity:

New syntax in the CREATE TABLESPACE statement allows you to define large partitioned table spaces, which can have a maximum of 254 partitions and hold up to one terabyte of data. 233

ASCII table support:

To support storing ASCII data in tables, the following SQL statements have a CCSID clause:

CREATE DATABASE	272
CREATE GLOBAL TEMPORARY TABLE	275
CREATE TABLE	308
CREATE TABLESPACE	327

SQL standards conformance:

The check option of the CREATE VIEW statement is extended. 341

The ALTER TABLE and CREATE TABLE statements are extended so that foreign keys can reference unique keys as well as primary keys. 217 and 308

	Refer to Page
The ALTER TABLE and CREATE TABLE statements are extended to support the new delete rule of NO ACTION.	217 and 308
A list of columns can be specified on the REFERENCE privilege for the GRANT statement.	412
In a searched update or delete, the SELECT privilege is required in addition to the UPDATE or DELETE privilege for standards conformance.	357 and 477
Additional SQL enhancements:	
SQL CASE expressions provide the ability to create conditional logic wherever an expression is allowed.	103
The new NULLIF scalar function allows you to provide a null value when a nonnull might otherwise be returned. This function is passed two expressions and returns null if the arguments are equal.	153
The new STRIP function removes blanks or another specified character from the end, the beginning, or both ends of a string expression.	155
The new CREATE GLOBAL TEMPORARY TABLE statement allows you to create and use temporary tables that store the results of SQL transactions without logging and recovery.	275
The new MAXROWS clause of the ALTER and CREATE TABLESPACE statements allows you to specify a maximum number of rows to be inserted, loaded, or reorganized on each data page.	233 and 327
The new LOCKPART clause of the ALTER and CREATE TABLESPACE statements allows you to indicate whether selective partition locking is to be used when locking a partitioned table space.	233 and 327
The new PIECESIZE clause of the ALTER and CREATE INDEX statements allows you to indicate how large DB2 should make the data sets that make up a nonpartitioning index.	205 and 280
The new RENAME statement allows you to rename existing tables.	440
Catalog tables:	
In addition to changes in existing catalog tables, there are eight new catalog tables. Some of these new catalog tables are to support the redesign of the communications database (CDB), which is now implemented within the DB2 catalog.	

SYSIBM.IPNAMES
SYSIBM.LOCATIONS
SYSIBM.LULIST
SYSIBM.LUNAMES
SYSIBM.LUMODES
SYSIBM.MODESELECT
SYSIBM.SYSDUMMY1
SYSIBM.USERNAMES

534

Chapter 2. DB2 Concepts

Structured Query Language	21
Static SQL	21
Dynamic SQL	21
Deferred Embedded SQL	21
Interactive SQL	22
DB2 Call Level Interface (CLI)	22
Tables	22
Indexes	23
Keys	23
Unique Keys	23
Primary Keys	23
Parent Keys	24
Foreign Keys	24
Referential Integrity	24
Check Constraints	26
Storage Structures	26
Storage Groups	27
Databases	27
Catalog	27
Views	27
Application Processes, Concurrency, and Recovery	28
Locking, Commit, and Rollback	28
Unit of Work	29
Unit of Recovery	30
Rolling Back Work	30
Packages and Application Plans	30
Distributed Data	31
DB2 Private Protocol Access	32
DRDA Access	32
Remote Unit of Work	33
Connection Management for DB2 Private Protocol and DRDA Access	33
Character Conversion	37
Character Sets and Code Pages	38
System CCSIDs	39
Restrictions on BIT Data	40
Expanding Conversions	40
Contracting Conversions	41

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 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.

A source program containing static SQL statements must be processed by an SQL precompiler before it is compiled. The precompiler 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 Section 5 of *Application Programming and SQL Guide*.

Dynamic SQL

Programs containing embedded *dynamic* SQL statements must be precompiled like those containing 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 Section 6 of *Application Programming and SQL Guide*.

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 Section 2 of *Application Programming and SQL Guide*.

DB2 Call Level Interface (CLI)

DB2 Call Level Interface (DB2 CLI) is an alternative to using embedded static or dynamic SQL. DB2 CLI 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. Some of the features DB2 CLI provides that are not available in embedded SQL include:

- DB2 CLI 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 CLI can return result sets to those programs.

The *Call Level Interface Guide and Reference* describes the APIs supported with this interface.

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.
temporary table	A table described by the SQL statement CREATE GLOBAL TEMPORARY TABLE and used to hold data temporarily, such as the intermediate results of SQL transactions. Temporary tables persist as long as the application supports them. Table space and database operations, locking, logging, and recovery do not apply.
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 *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. 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 Section 2 (Volume 1) of *Administration Guide*.

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 “Referential Integrity” on page 24. 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. A unique key can be defined using the UNIQUE clause of the CREATE TABLE statement. A table can have an arbitrary number of unique keys.

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.

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.

Referential Integrity

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 nonnull values of a foreign key are valid only if they also appear as values of a parent key. The table containing the parent key is called the *parent table* of the referential constraint, and the table containing the foreign key is a *dependent* of that table.

Referential constraints are optional and can be defined using SQL CREATE TABLE and ALTER TABLE statements. Refer to Section 2 (Volume 1) of *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 <i>self-referencing constraint</i> .

The rules of referential integrity are:

insert rule	A nonnull insert value of the foreign key must match some value of the parent key of the parent table.
update rule	A nonnull 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 dependents of P in D.
- SET NULL, each nullable column of the foreign key of each dependent 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 descendents 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 Section 2 (Volume 1) of *Administration 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.

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. 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 Section 2 (Volume 1) of *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 containing 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 subselect 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 341.

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 DSN8510.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 DSN8510.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 419 and “UPDATE” on page 477 for restrictions that apply to views with derived columns. For information on referential constraints, see “Referential Integrity” on page 24.

Read-only views cannot be used for insert, update, and delete operations. For a discussion of read-only views, see “CREATE VIEW” on page 341.

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 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 Section 5 (Volume 2) of *Administration Guide* for more information about DB2 locks.

¹ See the description of a table space under “Storage Structures” on page 26. Concurrent execution of SQL statements and utilities is discussed in Section 5 (Volume 2) of *Administration Guide*.

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 or CICS environment. Refer to Section 4 of *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 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 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 and the changes can 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 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, the commit operation can be used to end the unit of work, thereby making the changes available to other application processes.

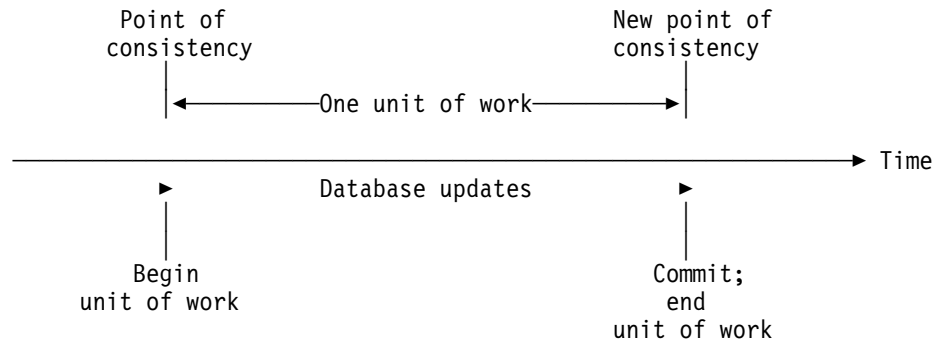


Figure 1. Unit of Work with a Commit Operation

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

If the 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:

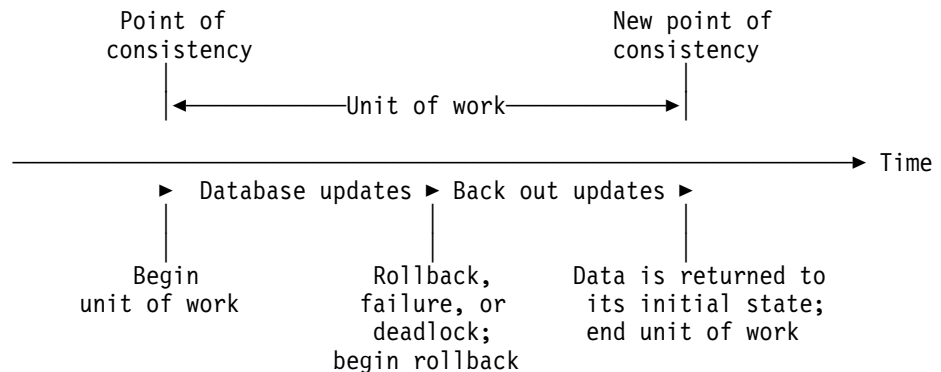


Figure 2. Rolling Back Changes from a Unit of Work

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 *Command Reference*. See Section 5 of *Application Programming and SQL Guide* for a description of program preparation and identifying packages at run time. Refer to “SET CURRENT PACKAGESET” on page 470 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 *application servers*. Any application server other than the local DB2 is considered a *remote server*, and access to its data is a distributed operation. DB2 provides two methods of accessing data at remote application servers:

- “DRDA Access” on page 32
- “DB2 Private Protocol Access” on page 32

For application 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 “Remote Unit of Work” on page 33. 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 21.
Servers	Can use any server that uses the DRDA protocols	Can use DB2 servers only
SQL statements	Can use any SQL statement supported by the system which executes the statement	Limited to SQL INSERT, UPDATE, and DELETE statements, and to statements supporting SELECT
Connection management	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 application server does not support two-phase commit.

DB2 Concepts

- 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 application requester to a DB2 Version 2 Release 2 application server is not supported).

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

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. 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 260 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 application server. A three-part name has the form:

location-name.aaaaa.sssss

where *aaaaa.sssss* uniquely identifies the object at the server named *location-name*. For example, the name USIBMSTODB21.DSN8510.EMP refers to a table named DSN8510.EMP at the server whose location name is USIBMSTODB21. Location naming conventions are described in “Location Identifiers” on page 48. Preparing DB2 for incoming SQL requests is discussed in Section 3 of *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 51. For more on three-part names, and on SQL naming conventions in general, see “Naming Conventions” on page 48.

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.
- 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 is based on a set of protocols known as *Distributed Relational Database Architecture* (DRDA), as documented in *Distributed Relational Database Architecture Reference*. DRDA communication conventions are invisible to DB2 applications, and allow a DB2 to bind and rebind packages at other servers and to

execute the statements in those packages. See Section 5 of *Application Programming and SQL Guide* for the steps involved in binding packages and plans. If the application 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 Section 3 of *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” on page 509.
- 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

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 application 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 259 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 262.

Connection Management for DB2 Private Protocol and DRDA Access

An *SQL connection* is an association between an application process and a local or remote application 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 application 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 application 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:

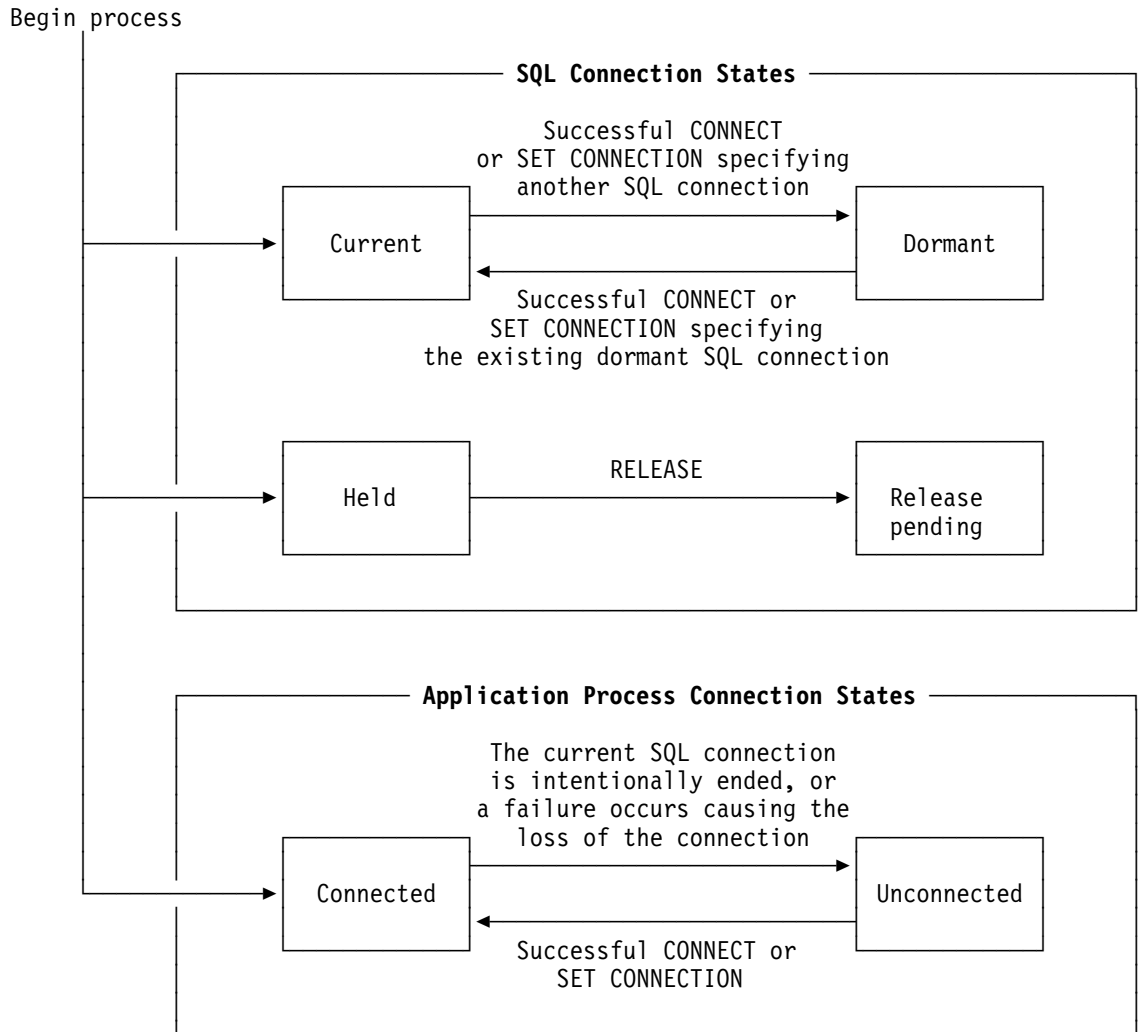


Figure 3. SQL Connection and Application Process Connection State Transitions

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 state.

An SQL connection is placed in the release pending state 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 state can still be used until the next commit operation. Likewise, DB2 private connections in the release pending state 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 application 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 application server of the current SQL connection. The application process can execute SQL statements that refer to objects managed by that application server. If the application 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 application 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 application requester are CONNECT, SET CONNECTION, RELEASE, COMMIT, ROLLBACK, and local SET statements. COMMIT and ROLLBACK are also processed by an application server. If the application process is in the unconnected state, the application 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 application 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 state

- The connection is not in the release pending state 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 both 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 implicit or explicit CONNECT statement has not already been successfully or unsuccessfully executed by the application process.

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

DB2 Private Connections

When the application 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 state.

An application process cannot have an SQL connection and a DB2 private connection to the same DB2 subsystem at the same time. 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 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 will be closed when the connection is ended during the next commit operation.

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 application requester to the current server.
- The values of result columns sent from the current server to the application 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 519).

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 when discussing character conversion.

character set	A defined set of characters. For example, the following character set appears in several code pages: <ul style="list-style-type: none"> • 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'. Within a code page, each code point has only one specific meaning.
code point	A unique bit pattern that represents a character.
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.

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.

encoding scheme

A set of rules used to represent character data. For example:

- Single-byte EBCDIC
- Single-byte ASCII²
- Double-byte EBCDIC
- Mixed single-byte and double-byte ASCII

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.

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 185

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 The term ASCII is used throughout this book to refer to IBM-PC Data or ISO 8 data.

code page: pp1 (ASCII)

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

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					!	:	Â	}	
F				À	ç	;	Á	{	

character set ss1
(in code page pp2)

Even with the same encoding scheme, there are many different coded character sets, 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 double-byte 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*.

System CCSIDs

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

- An SBCS string (*single-byte character set*). 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.
- A mixed string, in which both single-byte and double-byte characters can occur. In an EBCDIC mixed string, certain *shift characters* serve as left- and

right-delimiters for sequences of double-byte characters. MIXED is a subtype of the character data type.

At a given DB2, all columns containing SBCS strings are assumed to have a common CCSID known as the *corresponding ASCII or EBCDIC system CCSID* for SBCS data. Likewise, all columns containing graphic strings have a common CCSID, known as the corresponding ASCII or EBCDIC system CCSID for graphic data, and all columns containing mixed strings have a common CCSID known as the corresponding ASCII or EBCDIC system CCSID for mixed data. For example, DB2 can use a system CCSID when character data is fetched from a table at another DBMS. The system CCSID is used to convert the incoming data to the appropriate CCSID. If the character string has a subtype of BIT, its bytes do not represent characters and are not converted.

The values specified in fields ASCII CODED CHAR SET and EBCDIC CODED CHAR SET on installation panel DSNTIPF when DB2 was installed determine the ASCII and EBCDIC system CCSIDs. Those fields should contain valid SBCS CCSIDs if field MIXED DATA on that same installation panel is NO, or valid MIXED CCSIDs if field MIXED DATA is YES.

Field DEF ENCODING SCHEME on the same installation panel determines whether the default encoding scheme for the DB2 system is ASCII or EBCDIC. 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.

For more information about character string subtypes and SBCS and DBCS DB2 sites, see “Data Types” on page 57. For information on the subsystem parameters that determine the default encoding scheme and the system CCSIDs, see *Installation Guide*.

Restrictions on BIT Data

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.

Expanding Conversions

An *expanding conversion* occurs when the length of the converted string is greater than that of the source string. An expanding conversion occurs when an ASCII mixed data string containing 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. There is no remedy in FORTRAN.

Contracting Conversions

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

Chapter 3. Language Elements

Characters	45
Tokens	45
Spaces	46
Uppercase and Lowercase	46
Identifiers	46
SQL Identifiers	46
Location Identifiers	48
Host Identifiers	48
Naming Conventions	48
Aliases and Synonyms	51
Authorization IDs and Authorization-names	52
Authorization IDs and Statement Preparation	53
Authorization IDs and Dynamic SQL	54
Authorization IDs and Remote Execution	55
Data Types	57
Character Strings	57
Graphic Strings	60
Numbers	61
Datetime Values	62
Assignment and Comparison	65
Numeric Assignments	66
String Assignments	69
Datetime Assignments	71
Numeric Comparisons	72
String Comparisons	72
Datetime Comparisons	74
Constants	74
Integer Constants	74
Floating-Point Constants	75
Decimal Constants	75
Character String Constants	75
Datetime Constants	76
Graphic String Constants	77
Special Registers	78
General Rules for Special Registers	78
CURRENT DATE	79
CURRENT DEGREE	79
CURRENT PACKAGESET	80
CURRENT PRECISION	80
CURRENT RULES	81
CURRENT SERVER	82
CURRENT SQLID	82
CURRENT TIME	82
CURRENT TIMESTAMP	83
CURRENT TIMEZONE	83
USER	83
Column Names	84
Qualified Column Names	84
Correlation Names	84
Column Name Qualifiers to Avoid Ambiguity	85

#

Language Elements

	Column Name Qualifiers in Correlated References	86
	Resolution of Column Name Qualifiers	87
	Referencing Host Variables	89
	Host Structures in PL/I, C, and COBOL	90
	Expressions	92
	Without Operators	92
	With the Concatenation Operator	92
	With Arithmetic Operators	93
	Arithmetic with Two Integer Operands	94
	Arithmetic with an Integer and a Decimal Operand	94
	Arithmetic with Two Decimal Operands	94
	Arithmetic with Floating-Point Operands	97
	Datetime Operands and Durations	97
	Datetime Arithmetic in SQL	98
	Precedence of Operations	102
	CASE Expressions	103
	Predicates	106
	Basic Predicate	106
	Quantified Predicate	107
	BETWEEN Predicate	109
	EXISTS Predicate	109
	IN Predicate	111
	LIKE Predicate	112
	NULL Predicate	116
	Search Conditions	118
#	Options Affecting SQL	119
	Precompiler Options for Dynamic Statements	121
	Decimal Point Representation	121
	Apostrophes and Quotation Marks in String Delimiters	122
	Katakana Characters for EBCDIC	123
	Mixed Data in Character Strings	123
	Formatting of Datetime Strings	124
	SQL Standard Language	124
	Positioned Updates of Columns	126

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. 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, excluding the blank character, and excluding 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 433.

Examples:

, 'string' "fld1" = .

Identifiers

Spaces

A *space* is a sequence of one or more blank characters. Tokens, other than string constants and delimited identifiers, must not include a space. Any token can be followed by a space. Every ordinary token must be followed by a delimiter token or a space. If the syntax does not allow an ordinary token to be followed by a delimiter token, a space must follow that ordinary token.

Uppercase and Lowercase

Any token can include lowercase letters, but a lowercase letter in an ordinary token is folded to uppercase unless the SQL statement is embedded in a C program. Delimiter tokens are never folded to uppercase.

Example: The statement:

```
select * from DSN8510.EMP where lastname = 'Smith';
```

is equivalent, after folding, to:

```
SELECT * FROM DSN8510.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” on page 505 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 a table, column, alias, synonym, view, statement, cursor, or correlation 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 rules for forming EBCDIC DBCS SQL ordinary identifiers are as follows:

- 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.

The rules for forming ASCII DBCS SQL ordinary identifiers are as follows:

- The maximum length is 18 bytes.
- DBCS blanks are not acceptable.
- 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 E, “SQL Reserved Words” on page 621.

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 bind option DYNAMICRULES(RUN) applies and the field SQL STRING DELIMITER on installation panel DSNTIPF is set to the quotation mark ("). Here the escape character is the apostrophe (').

However, if the dynamic SQL statements are processed in a COBOL program and the bind option DYNAMICRULES(BIND) applies, a COBOL compiler option specifies whether the escape character is the quotation mark or apostrophe.

- 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.

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

“SYNONYM”

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 48 describes what

Naming Conventions

identifiers can represent and whether those representing a given type of entity are long or short.

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.

The table or view designated by an alias need not be at the current server and 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 those for forming a table name or view name, as explained below. A fully qualified alias name can refer to an alias defined at a server that is not the current server. But the table or view identified by such an alias must exist at the DB2 identified by the first part of the three-part alias name.

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 52 for the distinction between an authorization name and an authorization ID.

bpname	A name that identifies a buffer pool. The 4KB buffer pools are named BP0, BP1, BP2, ..., BP49. The 32KB buffer pools are named BP32K, BP32K1, BP32K2, ..., BP32K9.
catalog-name	A short identifier that designates an integrated catalog facility catalog.
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 <i>Command Reference</i> for naming conventions.
column-name	A qualified or unqualified name that designates a column of a table or view. The unqualified form of a column name is a long identifier. The qualified form 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.
constraint-name	A short identifier that designates a referential constraint on a table, or a long identifier that designates 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.
descriptor-name	A host identifier that designates an SQL descriptor area (SQLDA). See “Referencing Host Variables” on page 89 for a description of a host identifier. A descriptor name never includes an indicator variable.
host-variable	A sequence of tokens that designates a host variable. A host variable includes at least one host identifier, as explained in “Referencing Host Variables” on page 89.
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. The unqualified name is implicitly qualified by an authorization ID determined by the rules set forth in “Unqualified object names” on page 51.
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 <i>Command Reference</i> for naming conventions.
plan-name	A short identifier that identifies an application plan. Refer to Chapter 1 of <i>Command Reference</i> for naming conventions.

Naming Conventions

procedure-name	A qualified or unqualified name that designates a stored procedure. Each part of the name must be a long identifier that must be composed of SBCS characters.
program-name	A short identifier that designates an exit routine.
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.
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	<p>A qualified or unqualified name that designates a table.</p> <p>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.</p> <p>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.</p> <p>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 “Unqualified object names” on page 51.</p>
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.
version-id	An identifier ³ 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 <i>Command Reference</i> for naming conventions.

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

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 “Unqualified object names” on page 51.

Unqualified object names: Unqualified table, view, index, and alias 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 option is not used on BIND PLAN, the implicit qualifier is the authorization ID of the owner of the plan. If this option is not used on BIND PACKAGE, the implicit qualifier is the authorization ID of the owner of the package.
- For dynamic SQL statements:
 - If the bind option DYNAMICRULES(RUN) applies, the implicit qualifier is the identifier contained in the CURRENT SQLID special register. DYNAMICRULES(RUN) is the default.
 - If the bind option DYNAMICRULES(BIND) applies, the qualifier is the identifier implicitly or explicitly specified in the QUALIFIER option of the BIND subcommand, as explained above for static SQL statements. One exception to this rule is that the qualifier of PLAN_TABLE (output from the EXPLAIN statement) is always the value in special register CURRENT SQLID.

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

Authorization IDs and Authorization-names

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.
- 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 designate an alias defined at another server if that alias represents a table or view at the other server.) 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 the names of 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 and you execute the following statements interactively:

```
CREATE TABLE TDEPT LIKE DSN8510.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 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 secondaries), 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 *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 55.

If VALIDATE(BIND) is specified, the privileges required to manipulate tables and views 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 tables and views and the privileges

Authorization IDs and Authorization-names

required to use these tables and views 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 347.

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 55.

The bind option DYNAMICRULES determines, for authorization and qualification purposes, whether dynamic SQL statements are processed at run time with run-time rules, DYNAMICRULES(RUN), or with bind-time rules, DYNAMICRULES(BIND). DYNAMICRULES(RUN) is the default.

DYNAMICRULES(RUN) and run-time rules: 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.

When an SQL statement is dynamically prepared, the SQL authorization ID is used as the implicit qualifier for all tables, views, and indexes. If the 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 that applies to these statements is the union of the privileges designated by each authorization ID of the process.

If the dynamic SQL statement is a CREATE, GRANT, or REVOKE statement, the only authorization ID that is used for authorization checking is the SQL authorization ID. Therefore, each privilege required for the statement must be a privilege designated by that single authorization ID.

DYNAMICRULES(BIND) and bind-time rules: The rules are the same as the rules used for authorization checking and object qualification of embedded or static SQL statements, as explained in “Authorization IDs and Statement Preparation” on page 53 and “Unqualified object names” on page 51.

DB2 uses the primary authorization ID of the owner of the package or plan for authorization checking of dynamic SQL statements.

When DYNAMICRULES(BIND) is in effect, you cannot use the following SQL statements:

- The static or dynamic statement SET CURRENT SQLID
- The dynamic statements ALTER, CREATE, DROP, GRANT, RENAME, and REVOKE

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 server and requester
- DRDA access with a server and requester other than DB2
- DB2 private protocol access

DRDA Access with DB2 for OS/390 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. 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.

Before an application that has a plan can use the package, the owner of the
application's plan must have the EXECUTE privilege on the package. Again, this
privilege must be recorded in the server's DB2 catalog. The plan needs no other
privilege to execute the package.

A user who invokes an application that has a plan must have the EXECUTE
privilege on the plan. This allows the execution of the static SQL statements in the
package, and the execution of dynamic SQL statements if the bind option
DYNAMICRULES(BIND) is in effect. If DYNAMICRULES(RUN) 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 Section 3 (Volume 1) of *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, which is described next
under "DRDA Access with a Server or Requester Other Than DB2 for OS/390."

DRDA Access with a Server or Requester Other Than DB2 for OS/390

DB2 as the server: If the application requester is not a DB2 for OS/390 subsystem, there is no DB2 application plan involved. In this case, the EXECUTE privilege is on the package and is derived from the user's primary authorization ID or secondary authorization ID. These authorization IDs are also used, as applicable, for dynamic SQL statements if the bind option DYNAMICRULES(RUN)

Authorization IDs and Authorization-names

applies. If DYNAMICRULES(BIND) applies, the package owner's authorization ID is used for dynamic SQL statements.

DB2 as the requester: The authorization rules for remote execution are those of the server.

DB2 Private Protocol Access

Any statement referring to a table or view at a DB2 subsystem other than the current server 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 option DYNAMICRULES(RUN) applies, JONES, who is executing the dynamic statement at the server, is used. If the option DYNAMICRULES(BIND) 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 Section 3 (Volume 1) of *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 based on RACF considerations. Developing such security measures is discussed in Section 3 (Volume 1) of *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:

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

Figure 4 illustrates the data types supported by DB2.

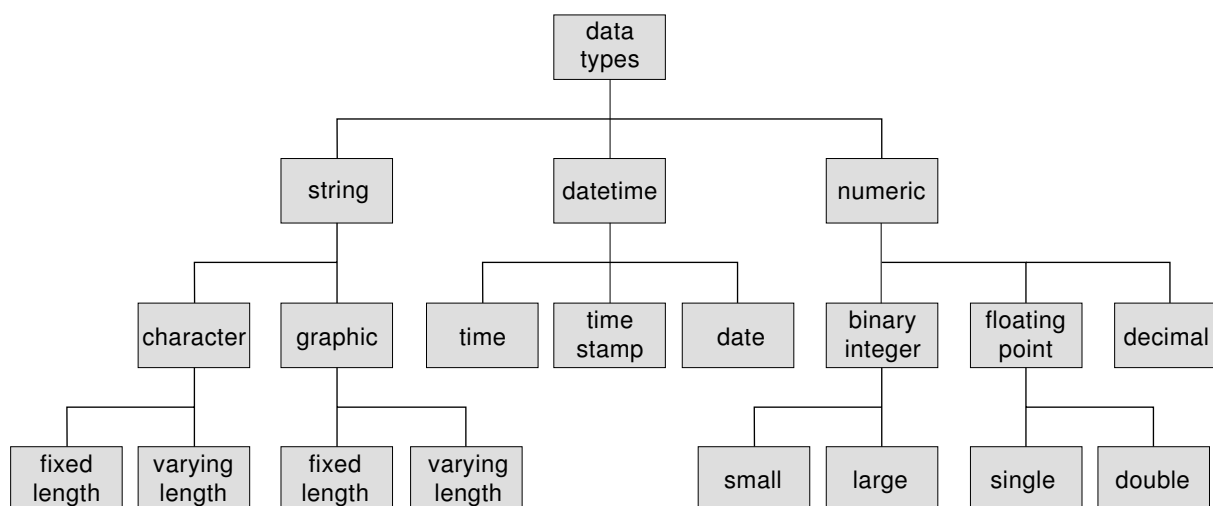


Figure 4. Data Types Supported by DB2

Nulls: All data types include the null value. The null value is a special value that is distinct from all nonnull values and thereby denotes the absence of a (nonnull) value. Although all data types allow for the null value, some sources of values cannot contain null values. For example, all constants, columns defined as NOT NULL, special registers, and the COUNT function cannot contain null values.

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, which is a small integer that precedes that string. For varying-length character strings, the length control field specifies the number of bytes. For varying-length graphic strings, the length control field specifies the number of DBCS characters.

All character strings have the subtype SBCS, MIXED, or BIT:

- If the subtype of a character string is SBCS, its bytes are assumed to represent characters from a single-byte character set. Such strings are called SBCS data.

- If the subtype of a character string is MIXED, it may contain both SBCS and DBCS characters. EBCDIC mixed data may contain shift bytes, which represent neither SBCS nor DBCS data. Strings that may contain both SBCS and DBCS characters are called *mixed data*.
- If the subtype of a character string is BIT, its bytes do not represent characters and therefore should never be converted. Such strings are called BIT data.

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 is SBCS or MIXED depending on the value of the field MIXED DATA on installation panel DSNTIPF.

- If the subtype is BIT, the CCSID is X'FFFF' (65535).
- If the subtype is SBCS, the CCSID is the system CCSID for SBCS data.
- If the subtype is MIXED, the CCSID is the system CCSID for mixed data.

The subtype of a character string column is recorded in the FOREIGNKEY column of the SYSCOLUMNS catalog table. 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 purposes other than character conversion.

DBCS Characters and ASCII and EBCDIC

The method of representing DBCS characters within a mixed string differs between ASCII and EBCDIC.

- 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.
- 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.

Examples

$\bar{u}_{gen} \bar{x}_{ki}$ CHAR(9) in ASCII.

$s_0 \bar{u}_1 s_{gen} s_0 \bar{x}_1 s_{ki}$ CHAR(13) in EBCDIC.

Because of these differences, 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 systems.

SBCS Sites

An SBCS site is a DB2 in which the subtype of character strings is SBCS or BIT. The value of field MIXED DATA on installation panel DSNTIPF is NO. The values of fields ASCII CODED CHAR SET and EBCDIC CODED CHAR SET determine the system CCSID that identifies the SBCS coded character set used at that site. The default subtype is SBCS data.

DBCS Sites

A DBCS site is a DB2 in which the subtype of character strings can be SBCS, BIT, or MIXED. The value of field MIXED DATA on installation panel DSNTIPF is YES. The values of fields ASCII CODED CHAR SET and EBCDIC CODED CHAR SET determine the system CCSIDs used for SBCS data, mixed data, and graphic data. The default subtype is mixed data.

A mixed data string can have zero or more sequences of SBCS characters and zero or more sequences of DBCS characters. Each EBCDIC DBCS 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 DBCS sites (the field MIXED DATA is YES). 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 the bind option DYNAMICRULES(BIND) applies and the GRAPHIC precompiler option is implicitly or explicitly specified

Fixed-Length Character Strings

All values of a fixed-length string column 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*.

Varying-Length Character Strings

The values of a varying-length string column can have different lengths. The maximum length is determined by the length attribute of the 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 324 in the description of the CREATE TABLE statement.

A varying-length character string column with a length attribute greater than 255 is a *long string column*. Long string columns cannot be referenced in:

- A function other than SUBSTR, LENGTH, or VALUE
- A GROUP BY clause
- An ORDER BY clause
- A CREATE INDEX statement
- A SELECT DISTINCT statement
- A subselect of a UNION without the ALL keyword
- A predicate other than EXISTS or LIKE (for LIKE, you can use a long string column for the first operand but not for the second operand)
- A *result-expression* in a CASE expression
- Primary, unique, and foreign keys

Character String Host Variables

Fixed-length string variables can be defined in all host languages. (In C, fixed-length string variables are limited to a length of 1.) Varying-length string variables can be defined in all host languages except FORTRAN. In Assembler, C, and COBOL, varying-length string variables are simulated as described in Section 3 of *Application Programming and SQL Guide*. In C, varying-length string variables can also be represented by NUL-terminated strings.

Character string variables with a maximum length greater than 255 are long string variables. Long string variables are subject to the same restrictions as long string columns. In addition, long string variables and long string columns (as opposed to short strings variables and short string columns) cannot be used to represent datetime values.

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. Every graphic string has a CCSID that identifies a double-byte coded character set. At a DBCS site, the CCSID of every graphic string column is the system CCSID for GRAPHIC data.

Fixed-Length Graphic Strings

All values of a fixed-length graphic column have the same length, given 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.

Varying-Length Graphic Strings

The values of a varying-length string column can have different lengths. The maximum length is determined by the length attribute of the 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 324 in the description of the CREATE TABLE statement. A varying-length graphic string column with a length attribute greater than 127 is a long string column. Long graphic string columns are subject to the same restrictions as long character string columns. In all cases, the length control field of a varying-length graphic string indicates the number of characters, not bytes.

Graphic String Host Variables

Graphic string variables with a maximum length greater than 127 are long string variables. Long string variables are subject to the same restrictions as long string columns.

Graphic variables can be defined in all host languages except FORTRAN.

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

A *small integer* is a System/370 binary integer with a precision of 15 bits. The range of small integers is -32768 to +32767.

Large Integer

A *large integer* is a System/370 binary integer with a precision of 31 bits. The range of large integers is -2147483648 to +2147483647.

Single Precision Floating-Point

A *single precision floating-point* number is a System/370 short (32 bits) floating-point number. The range of magnitude is about 5.4E-79 to 7.2E+75.

Double Precision Floating-Point

A *double precision floating-point* number is a System/370 long (64 bits) floating-point number. The range of magnitude is about 5.4E-79 to 7.2E+75.

Decimal

A *decimal* number is a System/370 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 largest positive number that can be represented with the applicable precision and scale. The maximum range is $1 - 10^{31}$ to $10^{31} - 1$.

Numeric Host Variables

Binary integer and floating-point variables can be defined in all host languages. 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 63.

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.⁴ 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.

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 Appendix B (Volume 2) of *Administration Guide*.) In that case, the string format of a date can be up to 255 bytes in length. Accordingly, DCLGEN⁵ 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

⁴ 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.

⁵ 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.

|

DB2 subsystem was installed. (Writing a date exit routine is described in Appendix B (Volume 2) of *Administration Guide*.) In that case, the string format of a time can be up to 255 bytes in length. Accordingly, DCLGEN⁵ 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. DCLGEN⁵ 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 character 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.

For retrieval, datetime values must be assigned to character 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 character string. The following sections describe the string formats that are recognized by DB2.

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

Date strings: A string representation of a date is a character 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.

Valid string formats for dates are listed in Table 2 on page 64. 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 2. 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 different (DSNXVDTA versus DSNXVDTX) than the exit for EBCDIC data.

Time strings: A string representation of a time is a character 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.

Valid string formats for times are listed in Table 3. 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.

Table 3. Formats for String Representations of Times

Format Name	Abbreviation	Time Format	Example
International Standards Organization ⁶	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 time exit for ASCII data is different (DSNXVTMA versus DSNXVTMX) than the exit for EBCDIC data.

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.

⁶ This is an earlier version of the ISO format. JIS can be used to get the current ISO format.

- 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 character 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. To have date and time host variables returned in another 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 65.

Assignment and Comparison

The basic operations of SQL are assignment and comparison. Assignment operations are performed during the execution of CALL, INSERT, UPDATE, FETCH, and SELECT INTO statements. 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 such as UNION and concatenation. The compatibility matrix for data types is shown in the following table.

Assignment and Comparison

Table 4. Compatibility of Data Types

Operands	Binary Integer	Decimal Number	Floating Point	Character String	Graphic String	Date	Time	Time-stamp
Binary Integer	Yes	Yes	Yes	No	No	No	No	No
Decimal Number	Yes	Yes	Yes	No	No	No	No	No
Floating Point	Yes	Yes	Yes	No	No	No	No	No
Character String	No	No	No	Yes	No	*	*	*
Graphic String	No	No	No	No	Yes	No	No	No
Date	No	No	No	*	No	Yes	No	No
Time	No	No	No	*	No	No	Yes	No
Time-stamp	No	No	No	*	No	No	No	Yes

Note: * The compatibility of datetime values is limited to assignment and comparison:

- Datetime values can be assigned to character string columns and to character string variables, as explained in “Datetime Assignments” on page 71.
- 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.

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 “Referencing Host Variables” on page 89 for a discussion of indicator variables.

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	2.0000045E6
assigned to an integer	
column or host variable is:	2000000

Example 2:

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

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

Example 1:

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

Example 2:

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

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

Example 1:

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

Example 2:

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

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:

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:	.0000012345678910000000000000000

Example 2:

The floating-point number	1.2339999999999E+01
in decimal notation is:	12.33999999999900 +5
Rounding adds 5 in the 16th position	12.33999999999905
and truncates the result to	12.3399999999990
Zeros are then added to the end of a 31-digit result:	12.33999999999900000000000000000

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 71.

The basic rule for string assignments is that the length of a string assigned to a column must not be greater than the length attribute of the column. (Trailing blanks are included in the length of the string.)

Assignment and Comparison

When a string is assigned to a fixed-length string column or host 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 SBCS or DBCS blanks, even when the source or target is BIT data.

When a string of length n is assigned to a varying-length string variable with a maximum length greater than n , the characters after the n th character of the variable are undefined and might or might not be set to blanks.

When a string is assigned to a variable and the string is longer than the length attribute of the variable, the string is truncated on the right by the necessary number of 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.

Assigning a mixed data string: A mixed data string containing DBCS characters cannot be assigned to an SBCS column or an SBCS 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.

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 column or a varying-length string column:

- If the source is a fixed-length string column, 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 column, 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 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.
- The SYSSTRINGS catalog table indicates that conversion is required.

An error occurs if:

- The SYSSTRINGS table is used but contains no information about the pair of CCSIDs.
- 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 containing 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 value assigned to a DATE column must be a date or a valid string representation of a date. A date can only be assigned to a DATE column, a character string column, or a character string variable. A value assigned to a TIME column must be a time or a valid string representation of a time. A time can only be assigned to a TIME column, a character string column, or a character string variable. A value assigned to a TIMESTAMP column must be a timestamp or a valid string representation of a timestamp. A timestamp can only be assigned to a TIMESTAMP column, a character string column, or a character string variable. A datetime value cannot be assigned to a column that has a field procedure.

When a datetime value is assigned to a character 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 character column, 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

Assignment and Comparison

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.

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 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. When it occurs, conversion takes place after any application of a field procedure. Conversion is necessary only if all of the following are true:

- The CCSIDs of the two strings are different.
- Neither CCSID is X'FFFF' (neither string is defined as BIT data).
- The string selected for conversion is neither null nor empty.
- The SYSSTRINGS catalog table indicates that conversion is required.

The conversion that occurs when SBCS data is compared with mixed data 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.

Otherwise, 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 5. 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.

Constants

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 or the SYSSTRINGS table is used but contains no information about the pair of CCSIDs of the operands being compared. 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'
```

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.

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.

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.  +3758933333333333333333333333.33
```

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 122. 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. 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. This form of string 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.

Constants

At DBCS sites, 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 application server. A mixed string constant can be continued from one line to the next only if the break occurs between single byte characters.

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 75. For information about the valid string formats, see “String Representations of Datetime Values” on page 63.

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 57.)

In EBCDIC environments, the forms of graphic string constants are⁷:

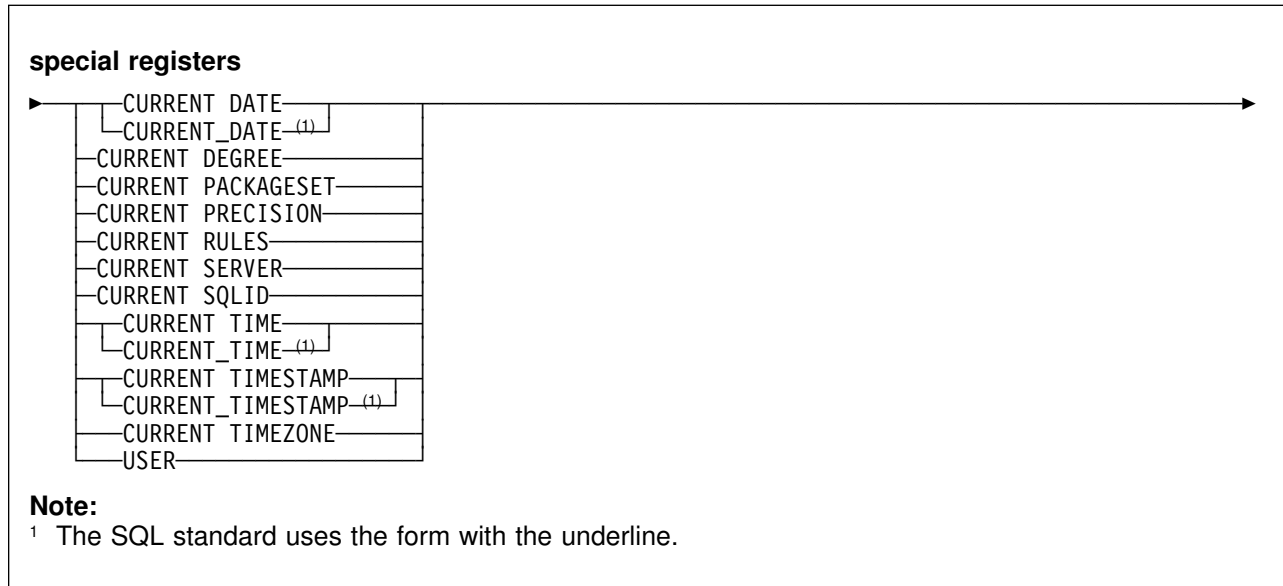
Context	Graphic String Constant	Empty String	Example
PL/I	$\text{S}_0 \text{ / d b c s - s t r i n g } \text{ / G } \text{S}_1$ $\text{S}_0 \text{ / d b c s - s t r i n g } \text{ / S}_1 \text{ G}$	$\text{S}_0 \text{ ' ' G } \text{S}_1$ $\text{S}_0 \text{ ' ' S}_1 \text{ G}$	$\text{S}_0 \text{ ' 元 氣 ' G } \text{S}_1$
	<p>G represents a DBCS G (X'42C7')</p> <p>/ represents a DBCS apostrophe (X'427D')</p>		
All other contexts	$\text{G ' S}_0 \text{ d b c s - s t r i n g } \text{ S}_1 \text{ '}$	$\text{G ' S}_0 \text{ S}_1 \text{ '}$ G '' $\text{g ' S}_0 \text{ S}_1 \text{ '}$ g ''	$\text{G ' S}_0 \text{ 元 氣 } \text{S}_1 \text{ '}$
	$\text{N ' S}_0 \text{ d b c s - s t r i n g } \text{ S}_1 \text{ '}$	$\text{N ' S}_0 \text{ S}_1 \text{ '}$ N '' $\text{n ' S}_0 \text{ S}_1 \text{ '}$ 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.

⁷ The PL/I form of graphic string constants is supported only in static SQL statements.

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:



General Rules for Special Registers

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

Changing register values: A commit or 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 DEGREE, CURRENT PACKAGESET, CURRENT PRECISION, CURRENT RULES, and CURRENT SQLID.
- 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 DEGREE, CURRENT PACKAGESET, CURRENT PRECISION, CURRENT RULES, CURRENT SERVER, CURRENT SQLID, and USER. The CCSID that is associated with these registers is either the one named in the ASCII CODED CHAR SET or EBCDIC CODED CHAR SET 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 or EBCDIC 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 or ASCII.

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.

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 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 DATE⁸.

CURRENT DATE

CURRENT DATE, or equivalently CURRENT_DATE, specifies the current date. The data type is DATE. 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 78.

Example: Display the average age of employees.

```
SELECT AVG(YEAR(CURRENT DATE - BIRTHDATE))
FROM DSN8510.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 Section 5 (Volume 2) of *Administration Guide* for a description of query parallelism.

⁸ 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

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. You can change the value of the register by executing the statement SET CURRENT DEGREE.

CURRENT DEGREE is a register at the application server. Its value applies to queries that are dynamically prepared at that application 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 application server and that DB2 subsystem.

Example: The following statement inhibits parallelism:

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

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. See “SET CURRENT PACKAGESET” on page 470 for details about this statement.

Example: Before passing control to another program, identify the collection ID for its package as ALPHA.

```
EXEC SQL SET CURRENT PACKAGESET = 'ALPHA';
```

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), and the only valid values are 'DEC15' and 'DEC31'. 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.

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. You can change the value of the register by executing the
statement SET CURRENT PRECISION.

CURRENT PRECISION only affects dynamic SQL. If the value of CURRENT
PRECISION is DEC15 when an SQL statement is dynamically prepared, DEC15
rules will apply. If the value of CURRENT PRECISION is DEC31 when an SQL
statement is dynamically prepared, 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. For more information, see “Arithmetic with Two
Decimal Operands” on page 94.

Example: Set CURRENT PRECISION so that subsequent statements that are
prepared use DEC31 rules for decimal arithmetic:

```
# SET CURRENT PRECISION = 'DEC31';
```

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 application 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. You can change the value of the register by executing the statement SET CURRENT RULES.

CURRENT RULES affects the statements listed in Table 6. 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 6. Summary of Statements Affected by CURRENT RULES

Statement	What is Affected	Details on Page
ALTER TABLE	Enforcement of check constraints added. Default value of the delete rule for referential constraints.	217
CREATE TABLE	Default value of the delete rule for referential constraints.	308
DELETE	Authorization requirements for searched DELETE.	357
GRANT	Granting privileges to yourself.	400
REVOKE	Granting privileges to yourself.	443
UPDATE	Authorization requirements for searched UPDATE.	477

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 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. 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 78.

Example: Display information about all project activities and include the current date and time in each row of the result.

```
SELECT DSN8510.PROJACT.*, CURRENT DATE, CURRENT TIME  
FROM DSN8510.PROJACT;
```

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 78.

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.

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 Section 3 (Volume 1) of *Administration Guide*.

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;
```

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
- 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 5. Queries, which begins on page 167.) 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.

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 ON 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 85 and “Column Name Qualifiers in Correlated References” on page 86 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 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 or view only within the context in which it is defined. Hence, the same correlation name can be defined for different purposes in different statements, or in different clauses of the same statement.

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.

Column Name Qualifiers to Avoid Ambiguity

In the context of a function, a GROUP BY clause, ORDER BY clause, an expression, or a search condition, a column name refers to values of a column in some table or view. The tables and views 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 DSN8510.EMP Z, DSN8510.EMPPROJACT
WHERE WORKDEPT = 'D11'
AND EMPTIME > 0.5
AND Z.EMPNO = DSN8510.EMPPROJACT.EMPNO;
```

This example illustrates how to establish table designators in the FROM clause:

- A name that follows a table name, view name, alias, or synonym is both a correlation name and a table designator. Thus, Z is a table designator and qualifies the first column name after SELECT.
- A table name, view name, alias, or synonym that is *not* followed by a correlation name is a table designator. Thus, the qualified table name, DSN8510.EMPPROJACT 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.

Column 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 DSN8510.EMP X, DSN8510.EMP Y
       WHERE Y.JOB = 'MANAGER'
              AND X.WORKDEPT = Y.WORKDEPT
              AND X.JOB <> 'MANAGER';
```

Column Name Qualifiers in Correlated References

A *subselect* is a form of a query that can be used as a component of various SQL statements. Refer to “Chapter 5. Queries” on page 167 for more information on subselects. A subselect 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 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 DSN8510.EMP at the level of the first FROM clause (which establishes X as a correlation name for DSN8510.EMP.). The statement lists employees who make less than the average salary for their department.


```

SELECT EMPNO, LASTNAME, WORKDEPT
FROM DSN8510.EMP X
WHERE SALARY < (SELECT AVG(SALARY)
                FROM DSN8510.EMP
                WHERE WORKDEPT = X.WORKDEPT);

```

Resolution of Column Name Qualifiers

A name in a FROM clause can be an alias, a synonym, a table name, a view name, or a correlation name. If it is a correlation name, or a name that is not followed by a correlation name, it is called an *exposed* name.

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. In DB2 SQL, those two rules are only guidelines; they are not enforced. Hence, in DB2, the qualifier of a column name is not required to be a unique exposed name, even though good SQL coding practice would always make it so.

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 the bind option DYNAMICRULES(RUN) applies, 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) applies 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 except that a column name in ORDER BY can only identify a column of the result table. Thus, qualification is necessary only if the result table has duplicate column names. For example, if the FROM clause refers to two tables, both of which have a column named C, and if the SELECT clause includes a single reference to C, then a reference to C in the ORDER BY clause need not be qualified.

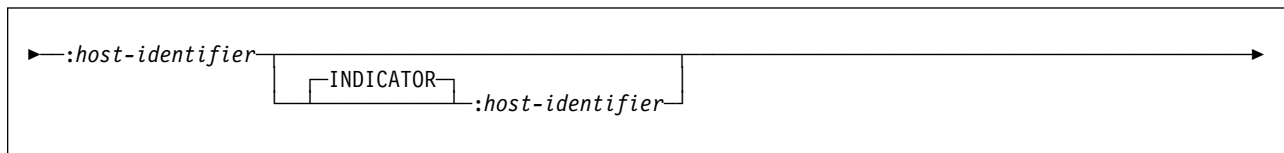
Referencing Host Variables

A *host variable* is a PL/I variable, C variable, FORTRAN variable, COBOL data item, or Assembler language storage area that is referred to in an SQL statement. Host variables are defined by statements of the host language as described in Section 3 of *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 90. 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 *host-variable* statement and the INTO clause of a FETCH or SELECT 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. 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.

Host Structures in PL/I, C, and COBOL

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 and the column contains nulls, 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 should be preceded by a colon. The colon:

- Makes it clear that the name identifies a host variable rather than a column.
- Increases the portability of the program because IBM SQL, ANSI/ISO SQL, and the SQL of other IBM relational database products require the colon.
- Is required when using a precompiler other than the DB2 precompiler.
- Is required by CALL and CONNECT statements.

In some contexts, it is possible to specify a host variable in an SQL statement without a preceding colon. These contexts are not documented. When such a statement is precompiled, a warning message is issued, and the statement is processed as if the missing colon were present. Moreover, to allow for extensions to SQL, it is possible that future versions of DB2 will not allow the omission of the colon.

In a context in which either a host variable or column can be referenced, the use of an unqualified name without a colon is interpreted by the precompiler as a column name. However, if a qualified name such as S.V is used and S is a host structure that contains V, S.V is interpreted by the precompiler as a reference to a host variable. Thus, you must avoid declaring host structures with a name that is the same as any possible qualifiers of a column name specified in your program.

A COBOL host variable beginning with a digit is treated as a number if it can be so interpreted. For example, in the following WHERE clause, the value of column XYZ is compared to the current value of the host variable 123E1. Without the identifying colon, the value of XYZ would be compared to the floating-point representation of 1230.

```
WHERE XYZ > :123E1
```

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 Section 3 of *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:

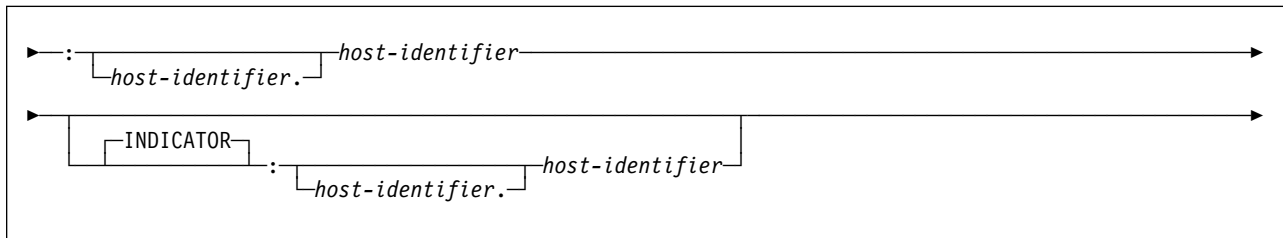
```
EXEC SQL FETCH CURSOR1 INTO :V1, :V2, :V3;
```


#

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.

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:



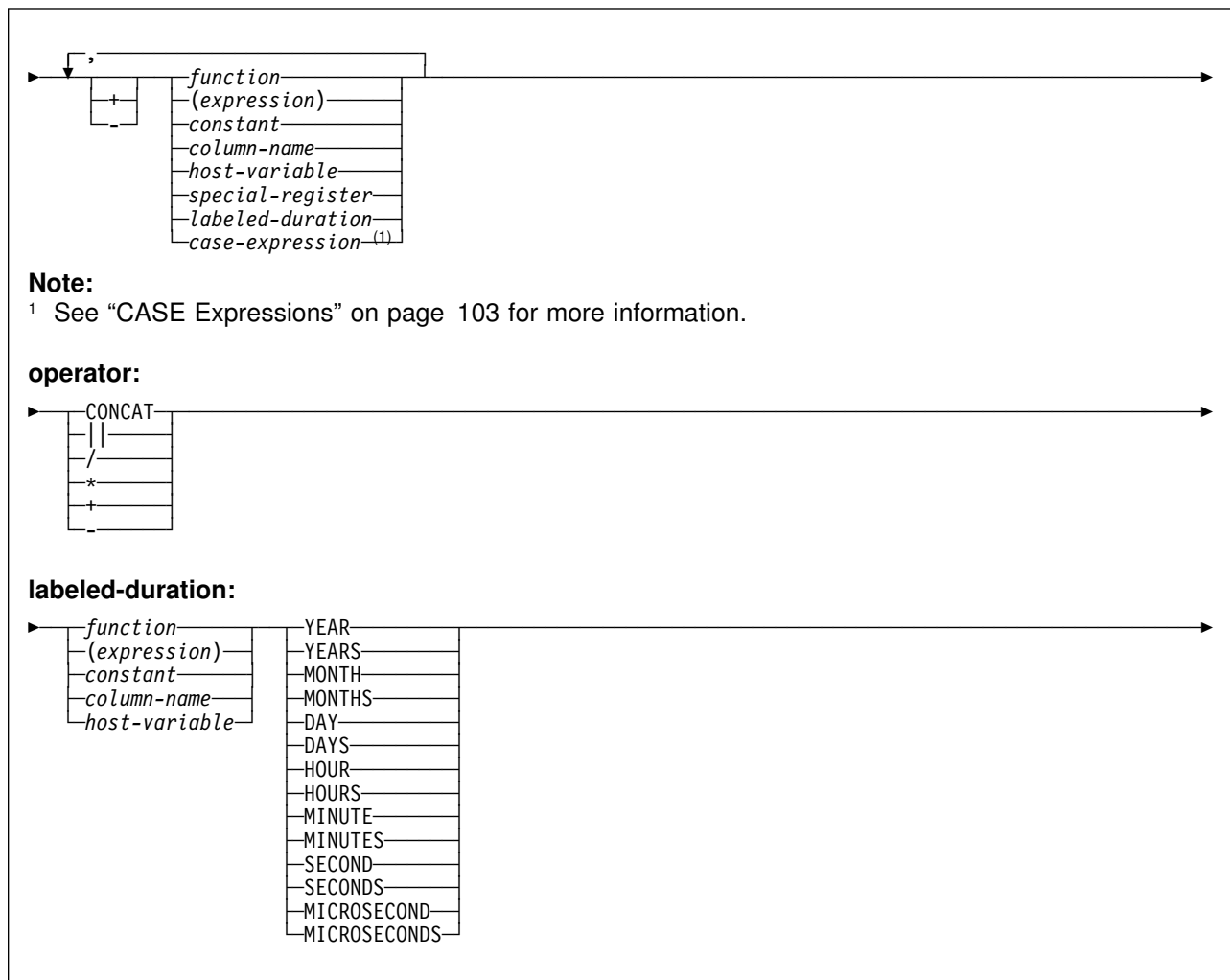
A *host-variable* in an expression must identify a host variable (not a structure) described in the program according to the rules for declaring host variables.

The following examples show references to host variables and host structures:

```
:V1   :S1.V1   :S1.V1:V2   :S1.V2:S2.V4
```

Expressions

An *expression* specifies a value. The form of an expression is as follows:



Note:

¹ See "CASE Expressions" on page 103 for more information.

operator:

labeled-duration:

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 countries⁹) can cause parsing errors in statements passed from one DBMS to another. The problem occurs if the statement undergoes character conversion with

⁹ 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.

certain combinations of source and target CCSIDs⁹. Thus, CONCAT is the preferable operator.

When concatenation is used, the result of the expression is a string. The operands of concatenation must be compatible strings. 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.

The length of the result is the sum of the lengths of the operands, unless both operands are mixed EBCDIC data, in which case two bytes are eliminated from the length of the result.

If both operands are character strings, the sum of their lengths must not exceed 32764. The result is a fixed-length character string if both operands are fixed-length, neither operand is mixed data, and the length of the result is not greater than 255. Otherwise, the result is a varying-length character string whose maximum length is the sum of the maximum lengths of the operands, or 32764, whichever is less.¹⁰ If the maximum length is greater than 255, the result is subject to the restrictions that apply to long strings.

#

If both operands are graphic strings, the sum of their lengths must not exceed 16382. The result is a fixed-length graphic string if both operands are fixed-length graphics strings and the length of the result is less than 128. Otherwise, the result is a varying-length graphic string whose maximum length is the sum of the maximum lengths of the operands, or 16382, whichever is less. If the maximum length is greater than 127, the result is subject to the restrictions that apply to long strings.

The CCSID of the result is determined by the rules set forth in “Character Conversion in Unions and Concatenations” on page 185. Some consequences of those rules are the following:

- If either operand is BIT data, the result is BIT data.
- If one operand is mixed data and the other is SBCS data, the result is:
 - Mixed data if the MIXED DATA option at the server is YES¹¹
 - SBCS data if the MIXED DATA option at the server is NO.

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.

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. If any operand can be null, or the expression is used in an outer SELECT list, the result 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

¹⁰ If both character strings are EBCDIC mixed data, the result will not have the redundant “shift-in” character (X'0F') ending the first operand and the “shift-out” character (X'0E') beginning the second operand at “the seam.” Thus, the length of the result is two less the sum of the lengths of the operands.

¹¹ The result is not necessarily well-formed mixed data.

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:

- # • The field DECIMAL ARITHMETIC applies if bind option DYNAMICRULES(RUN)
 # is in effect and the application has not set CURRENT PRECISION. The value
 # of the field also applies if bind option DYNAMICRULES(BIND) is in effect,
 # installation parameter DYNRULS is YES, and the application has not set
 # CURRENT PRECISION.
- # • The precompiler option applies if bind option DYNAMICRULES(BIND) is in
 # effect, installation parameter DYNRULS is NO, and the application has not set
 # CURRENT PRECISION.
- # • The special register applies if the application sets the register.

The value of DECIMAL ARITHMETIC is the default value for both 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.

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

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 (see the diagram on page92)

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 seven duration keywords: YEARS, MONTHS, DAYS, HOURS, MINUTES, SECONDS, or MICROSECONDS¹². 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. In this case, 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*, and *ss* represent, respectively, the number of years, months, days, hours, minutes, and

¹² The singular form of these keywords is also acceptable: YEAR, MONTH, DAY, HOUR, MINUTE, SECOND, and MICROSECOND.

seconds, and `zzzzz` represents the number of 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 433.

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 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 $RESULT = DATE1 - DATE2$.

Date Subtraction: $RESULT = DATE1 - DATE2$

- If $DAY(DATE2) \leq DAY(DATE1)$
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 $MONTH(DATE2) \leq MONTH(DATE1)$
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).

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. Section 3 of *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).

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

$$\text{TIME1} + \text{HOUR}(X) \text{ HOURS} + \text{MINUTE}(X) \text{ MINUTES} + \text{SECOND}(X) \text{ SECONDS}$$

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 $RESULT = TS1 - TS2$.

Timestamp Subtraction: $RESULT = TS1 - TS2$

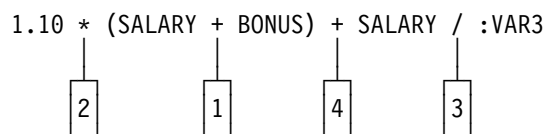
- If $MICROSECOND(TS2) \leq MICROSECOND(TS1)$
then $MICROSECOND(RESULT) = MICROSECOND(TS1) - MICROSECOND(TS2)$.
- If $MICROSECOND(TS2) > MICROSECOND(TS1)$
then $MICROSECOND(RESULT) = 1000000 + MICROSECOND(TS1) - MICROSECOND(TS2)$
and $SECOND(TS2)$ is incremented by 1.
- The seconds and minutes part of the timestamps are subtracted as specified in the rules for subtracting times.
- If $HOUR(TS2) \leq HOUR(TS1)$
then $HOUR(RESULT) = HOUR(TS1) - HOUR(TS2)$.
- If $HOUR(TS2) > HOUR(TS1)$
then $HOUR(RESULT) = 24 + HOUR(TS1) - HOUR(TS2)$
and $DAY(TS2)$ 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.

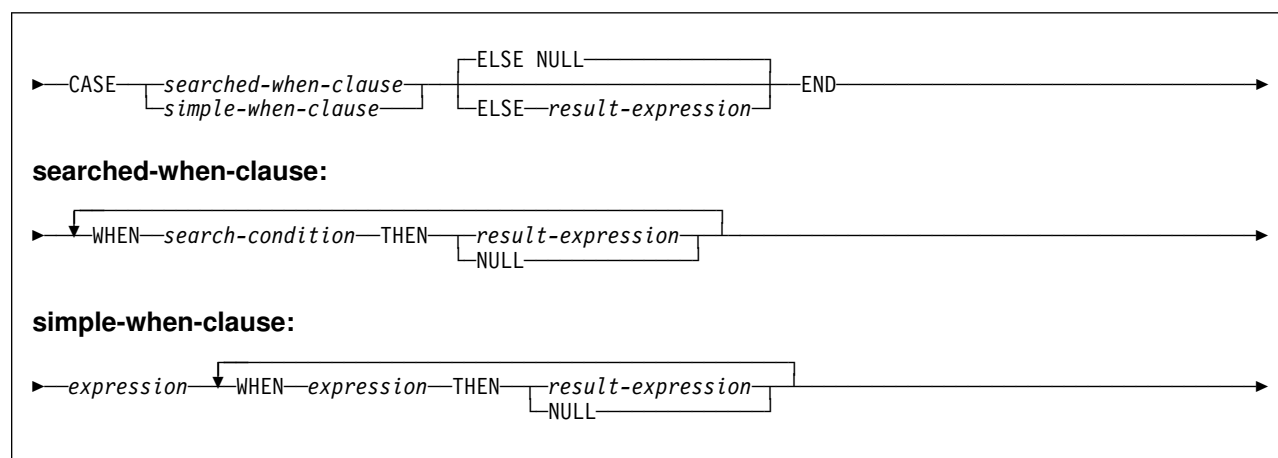
Example 1:



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.

```
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. Specifies the result for each WHEN keyword when the expressions are equal.

result-expression

Specifies the result of a *case-expression* if no case is true. Specifies the result of a *searched-when-clause* or a *simple-when-clause* that is true.

search-condition

Specifies a condition that is true, false, or unknown about a row or group of table data.

END

Ends a *case-expression*.

When using CASE expressions, consider the following usage information:

- If the CASE expression is in a select list, an IN predicate, or a SET clause of an UPDATE statement, then the *search-condition* in a searched-when-clause cannot be a quantified predicate, an IN predicate, or an EXISTS predicate.
- The *search-condition* in a searched-when-clause cannot contain a subselect and is diagnosed as a syntax violation.
- When using the simple-when-clause, the value of the *expression* prior to the first WHEN keyword is tested for equality with the value of the *expression* that follows the WHEN keyword. 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.
- A *result-expression* is an *expression* that follows the THEN or ELSE keywords. There must be at least one *result-expression* in the CASE expression, and NULL cannot be specified for every case.
- All *result-expressions* must be compatible. The attributes of the result are determined according to the rules for UNION and the VALUE function as described in “Data Type Rules for UNION and the VALUE Function” on page 183. 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 39.

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: This example shows how to group the results of a query by a CASE expression without having to retype the expression. Using the sample employee table, find the maximum, minimum, and average salary. Find these values by

```

#           department, but 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 DSN8510.EMP) X
#           GROUP BY CASE_DEPT;

```

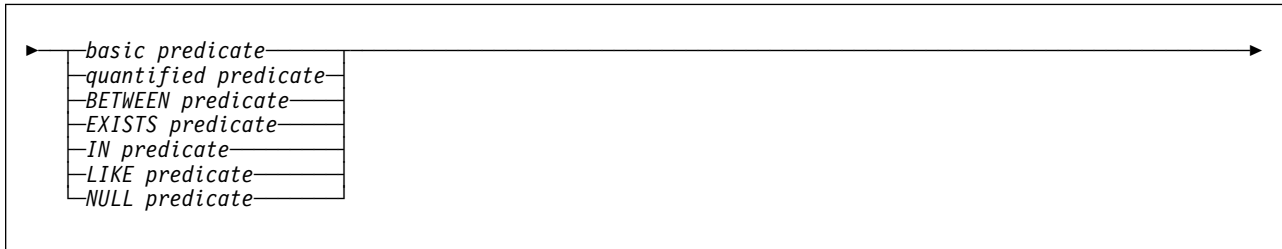
There are two scalar functions, NULLIF and COALESCE, that are specialized to handle a subset of the functionality provided by CASE. Table 8 shows the equivalent expressions using CASE or these functions.

Table 8. 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)

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 subselect in a predicate must specify a single column.

Basic Predicate



A *basic predicate* compares two values. If the value of either operand is null or the result of the subselect is empty, the result of the predicate is unknown. Otherwise, the result is either true or false.

A subselect in a basic predicate must not return more than one value, whether null or not null.

For values x and y:

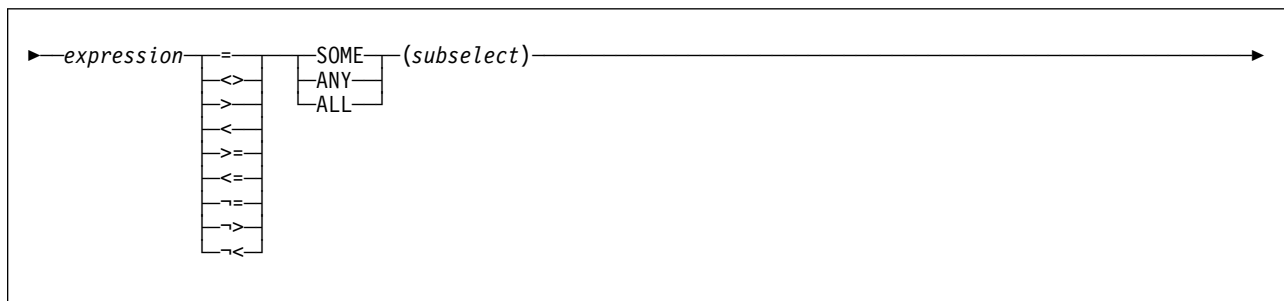
Predicate	Is true if and only if...
x = y	x is equal to y
x <> y	x is not equal to y
x < y	x is less than y
x > y	x is greater than y
x >= y	x is greater than or equal to y
x <= y	x is less than or equal to y
x \neq y	x is not equal to y
x \nless y	x is not less than y
x \ngtr y	x is not greater than y

A not sign (\neq), or the character that must be used in its place in certain countries¹³, 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 CCSIDs¹³. To avoid this problem, substitute an equivalent operator for any operator that includes a not sign. For example, substitute '<>' for ' \neq ', '<=' for ' \ngtr ', and '>=' for ' \nless '.

Examples:

```
EMPNO = '528671'
SALARY < 20000
PRSTAFF <> :VAR1
SALARY > (SELECT AVG(SALARY) FROM DSN8510.EMP)
```

Quantified Predicate



A *quantified predicate* compares a value with the set of values produced by the subselect. The subselect must specify a single result column and can return any number of values, whether null or not null.

When ALL is specified, the result of the predicate is:

- True if the result of the subselect is empty or if the specified relationship is true for every value returned by the subselect.
- False if the specified relationship is false for at least one value returned by the subselect.

¹³ DB2 supports code point combinations X'5F7E', X'BA7E', X'B07E', and X'EC7E' to mean not equal to. X'BA7E', X'B07E', and X'EC7E' are interpreted to mean not equal to only on single byte character set DB2 subsystems.

Predicates

- Unknown if the specified relationship is not false for any values returned by the subselect 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 subselect.
- False if the result of the subselect is empty or if the specified relationship is false for every value returned by the subselect.
- Unknown if the specified relationship is not true for any of the values returned by the subselect and at least one comparison is unknown because of a null value.

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:	<table border="1"><thead><tr><th>COLA</th></tr></thead><tbody><tr><td>1</td></tr><tr><td>2</td></tr><tr><td>3</td></tr><tr><td>4</td></tr></tbody></table>	COLA	1	2	3	4	TBLB:	<table border="1"><thead><tr><th>COLB</th><th>COLC</th></tr></thead><tbody><tr><td>2</td><td>2</td></tr><tr><td>3</td><td>-</td></tr></tbody></table>	COLB	COLC	2	2	3	-
COLA														
1														
2														
3														
4														
COLB	COLC													
2	2													
3	-													

Example 1: In the following predicate, the subselect 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)
```

Example 2: In the following predicate, the subselect 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)
```

Example 3: In the following predicate, the subselect 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)
```

Example 4: In the following predicate, the subselect 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)
```

Example 5: In the following predicate, the subselect returns an empty result column. Hence, the predicate is true for all rows of TBLA.

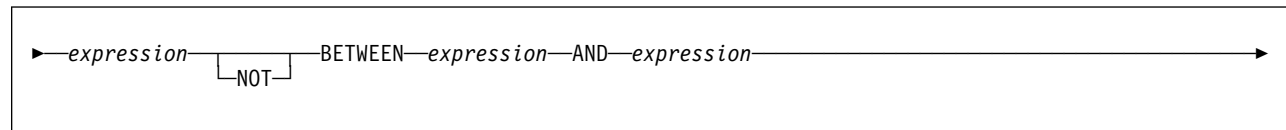
```
COLA < ALL(SELECT COLB FROM TBLB WHERE COLB>3)
```

Example 6: In the following predicate, the subselect returns an empty result column. Hence, the predicate is false for all rows of TBLA.

```
COLA < ANY(SELECT COLB FROM TBLB 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:

The predicate: `value1 BETWEEN value2 AND value3`
 is equivalent to: `value1 >= value2 AND value1 <= value3.`

The predicate: `value1 NOT BETWEEN value2 AND value3`
 is equivalent to: `NOT(value1 BETWEEN value2 AND value3)`
 and therefore also to: `value1 < value2 OR value1 > value3.`

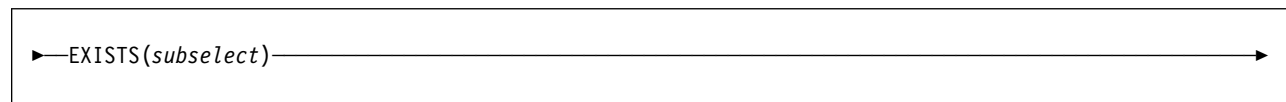
Search conditions are discussed in “Search Conditions” on page 118.

Example: The predicate:

`A BETWEEN B AND C`

- Is true when B is 1, C is 3, and A is 1, 2, or 3
- Is false when B is 1, C is 3, and A is 0 or 4
- Is false when A is 0, B is 1, and C is null
- Is false when A is 4, C is 3, and B is null
- Is unknown when any one of the following is true:
 - A is null
 - A is 2, B is 1, and C is null
 - A is 3, C is 4, and B is null.

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 subselect contains at least one row. Otherwise, the result is false.

The SELECT clause in the subselect can specify any number of columns because the values returned by the subselect are ignored. For convenience, use:

```
SELECT *
```

Unlike the NULL, LIKE, and IN predicates, the EXISTS predicate has no form containing the word NOT. To negate an EXISTS predicate, precede it with the logical operator NOT, as follows:

```
NOT EXISTS (subselect)
```

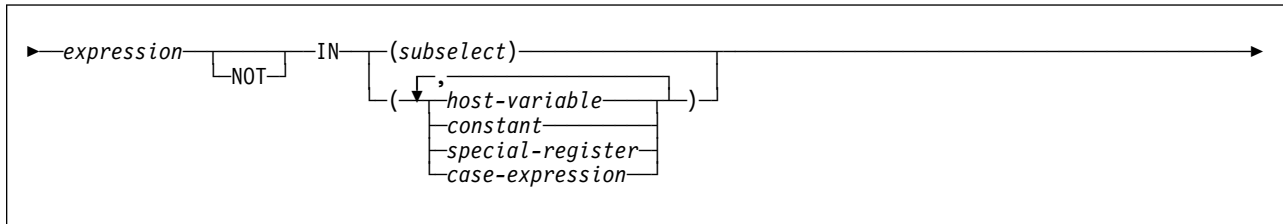
Predicates

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 118.

Example: The following query lists the employee number of everyone represented in DSN8510.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 DSN8510.EMP X
WHERE EXISTS (SELECT * FROM DSN8510.EMP
              WHERE X.WORKDEPT=WORKDEPT AND SALARY<20000);
```


IN Predicate



The IN predicate compares a value with a set of values.

In the subselect form, the subselect 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:

IN predicate	Quantified Equivalent
expression IN (subselect)	expression = ANY (subselect)
expression NOT IN (subselect)	expression <> ALL (subselect)

In the non-subselect form of the IN predicate, the second operand is a set of one or more values specified by any combination of host variables, constants, or special registers. 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
```

Each host variable must identify a structure or variable that is described in accordance with the rules for declaring host structures or variables.

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 DSN8510.EMP
WHERE WORKDEPT = 'E11')
```

Predicates

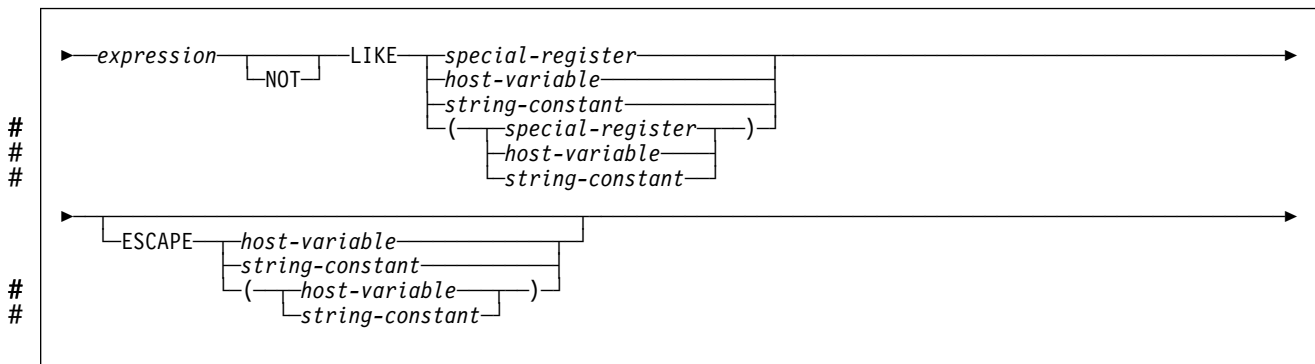
Example 3: The following example obtains the phone number of an employee in DSN8510.EMP where the employee number (EMPNO) is a value specified within the COBOL structure defined below.

```

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 DSN8510.EMP
      WHERE EMPNO IN
#          (:EMPNO-STRUCTURE.CHAR-ELEMENT-1,
#           :EMPNO-STRUCTURE.CHAR-ELEMENT-2,
#           :EMPNO-STRUCTURE.CHAR-ELEMENT-3)
      END-EXEC.
(:EMPNO-STRUCTURE)
      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 pattern is specified by a string in which the underscore and percent sign characters have special meanings, as discussed under “The Pattern” on page 114.

Evaluating the Predicate

The first operand specifies the string to be tested. A parameter marker must not be specified for or within the expression and the result of the expression must be a string. The second operand specifies the pattern. Let *x* denote the string to be tested and let *y* denote the pattern.

The following rules apply to predicates of the form “*x* LIKE *y*...”. Predicates of the form “*x* NOT LIKE *y*...” are equivalent to “NOT(*x* LIKE *y*...)”:

- When *x* and *y* are both neither empty nor null, the result of the predicate is true if *x* matches the pattern in *y* and false if *x* does not match the pattern in *y*. Matching the pattern is described beginning with “The Pattern” on page 114.

- When *x* or *y* is null, the result of the predicate is unknown.
- When *y* is empty and *x* is not, the result of the predicate is false.
- When *x* is empty and *y* is not, the result of the predicate is false unless *y* consists only of one or more percent signs.
- When *x* and *y* are both empty, the result of the predicate is true.

The Pattern String

The pattern string and the string to be tested must be of the same type, that is, both *x* and *y* must be character strings or both *x* and *y* must be graphic strings. When *x* and *y* are graphic strings, a character is a DBCS character. When *x* and *y* are character strings and *x* is not mixed data, a character is an SBCS character and *y* is interpreted as SBCS data regardless of its subtype. The rules for mixed data patterns are described later.

The pattern string can be specified as follows:

- “LIKE *special-register*” indicates that the string is in the named special register.
- “LIKE *host-variable*” indicates that the string is in the indicated host variable. The host variable must be defined in accordance with the rules for declaring string host variables and must not be a structure or a long string 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 Section 3 of *Application Programming and SQL Guide*.

- “LIKE *string-constant*” includes the pattern as a string constant within the predicate.

#

Although not required, parentheses can be used to enclose the string specified in *special-register*, *host-variable*, or *string-constant*, for example, LIKE ('ABC').

The Optional ESCAPE Clause

Within the pattern, a percent sign or underscore can have a special meaning, or it can represent the literal occurrence of a percent sign or underscore. To have its literal meaning, it must be preceded by an *escape character*. If it is not preceded by an escape character, it has its special meaning.

The ESCAPE clause designates a single character. That character—and only that character—can be used multiple times within the pattern as an escape character. When the ESCAPE clause is omitted, no character serves as an escape character, so that percent signs and underscores in the pattern always have their special meanings.

The following rules apply to the use of the ESCAPE clause:

- The ESCAPE clause cannot be used if *x* is mixed data.
- The escape character can be specified by a string constant or a host variable. If a host variable is used, it must be defined in accordance with the rules for

declaring fixed-length string host variables.¹⁴ If the host variable has a negative indicator variable, the result of the predicate is unknown.

- If *x* is a character string, the data type of the string constant or host variable must be character string. If *x* is a graphic string, the data type of the string constant or host variable must be graphic string. In both cases, the length of the string constant or host variable must be 1.¹⁴
- 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.

#

Although not required, parentheses can be used to enclose the character specified in *host-variable* or *string-constant*, for example, LIKE ('+').

The Pattern

When the pattern does not include escape characters, a simple description of its meaning is:

- The underscore sign (`_`) represents a single arbitrary character.
- The percent sign (`%`) represents a string of zero or more arbitrary characters.
- Any other character represents a single occurrence of itself.

A more rigorous description follows:

The string *y* is interpreted as a sequence of the minimum number of substring specifiers such that each character of *y* 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 percent sign.

The string *x* matches the pattern *y* if there exists a partitioning of *x* into substrings such that:

- A substring of *x* is a sequence of zero or more contiguous characters and each character of *x* is part of exactly one substring.
- If the *n*th substring specifier is an underscore, the *n*th substring of *x* is any single character.
- If the *n*th substring specifier is a percent sign, the *n*th substring of *x* 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 *x* is equal to that substring specifier and has the same length as that substring specifier.
- The number of substrings of *x* is the same as the number of substring specifiers.

When escape characters are present in the pattern string, an underscore, percent sign, or escape character represents a single occurrence of itself if and only if it is preceded by an odd number of successive escape characters.

¹⁴ If it is NUL-terminated, a C character string variable of length 2 can be specified.

Mixed data patterns: If *x* is mixed data, the pattern is assumed to be mixed data and its special characters are interpreted as follows:

- A single-byte underscore refers to one single-byte character; a double-byte underscore refers to one double-byte character.
- A percent sign, either single-byte or double-byte, refers to any number of characters of any type, either single-byte or double-byte.
- Redundant shift bytes in *x* or *y* are ignored.

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: This example illustrates the effect of successive occurrences of the escape character, which in this case is the plus sign (+).

When the pattern string is...	The pattern itself is...
+%	A percent sign
++%	A plus sign followed by zero or more arbitrary characters
+++%	A plus sign followed by a percent sign

Predicates

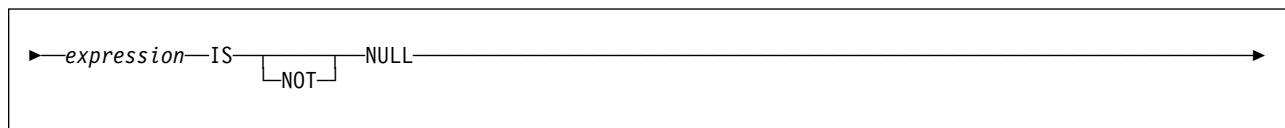
Example 5: In the following table, assume COL1 is a column containing mixed EBCDIC data. The table shows the results when the predicate in the first column is evaluated using the COL1 value in the second column:

Predicates	COL1 Values	Result
WHERE COL1 LIKE 'aaa AB% C'	'aaa ABDZC'	True
WHERE COL1 LIKE 'aaa AB% dzx C'	'aaa AB dzx C'	True
WHERE COL1 LIKE 'a% C'	'a C'	True
	'ax C'	True
	'ab DE fg C'	True
WHERE COL1 LIKE 'a _ C'	'a% C'	True
	'a X C'	False
WHERE COL1 LIKE 'a _ C'	'a X C'	True
	'ax C'	False
WHERE COL1 LIKE ' ' C'	Empty string	True
WHERE COL1 LIKE 'ab C _'	'ab C d'	True
	'ab C d'	True

Example 6: In the following table, assume COL1 is a column containing 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 'aaaAB%C'	'aaaABDZC'	True
WHERE COL1 LIKE 'aaaAB% C'	'aaaAB dzx C'	True

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.

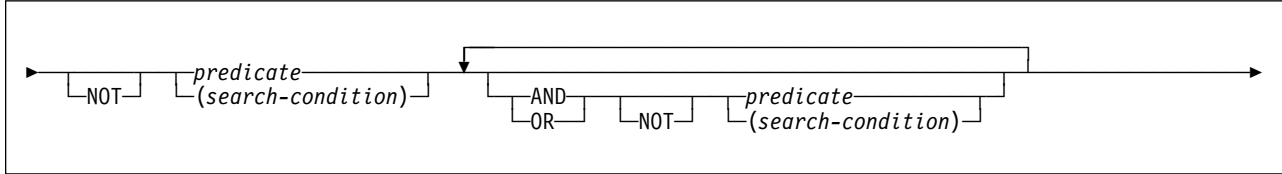
A parameter marker must not be specified for or within the expression.

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 9. 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, NOT(false) is true, and NOT(unknown) is 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)
```


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 10 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 10 (Page 1 of 2). Summary of Items Affecting Composition and Processing of SQL Statements

Precompiler Option	Field on Install Panel	Bind Option or Other	Affects
# # # # # # #		DYNRULS installation parameter	Use of precompiler options for dynamic statements when DYNAMICRULES(BIND) applies. For details, see "Precompiler Options for Dynamic Statements" on page 121.
COMMA PERIOD	DECIMAL POINT IS		Representation of decimal points in SQL statements. For details, see page 121.
APOSTSQL QUOTESQL	SQL STRING DELIMITER		Representation of string delimiters in SQL statements. For details, see page 122.
 	EBCDIC CODED CHAR SET		Use of a numeric value indicating the EBCDIC CCSID. Use of Katakana characters in ordinary identifiers. For details, see page 123.
	ASCII CODED CHAR SET		Use of a value in ASCII format. For details, see page 123.
GRAPHIC NOGRAPHIC	MIXED DATA		Use of character strings with a mixture of SBCS and DBCS characters. For details, see page 123.
DATE TIME	DATE FORMAT TIME FORMAT LOCAL DATE LENGTH LOCAL TIME LENGTH		Formatting of datetime strings. For details, see page 124.
 	STDSQL		Conformance with SQL standard. For details, see page 124.

Options Affecting SQL

Table 10 (Page 2 of 2). Summary of Items Affecting Composition and Processing of SQL Statements

Precompiler Option	Field on Install Panel	Bind Option or Other	Affects
NOFOR			Whether the FOR UPDATE OF clause must be specified (in the SELECT statement of the DECLARE CURSOR statement). For details, see page 126.
CONNECT			Whether the rules for a type 1 or a type 2 CONNECT statement apply. See "CONNECT" on page 259 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 261.
		DYNAMICRULES bind option	Whether dynamic SQL statements are processed with bind-time rules or run-time rules. For details, see "Authorization IDs and Dynamic SQL" on page 54. DYNAMICRULES also determines whether DB2 applies precompiler options or application programming defaults to dynamic SQL statements, as in the case for decimal point representation, string delimiters, and mixed data. For details, see page 121, 122, and 123..
		SQLRULES bind option	Whether a type 2 CONNECT statement is processed with DB2 rules or SQL standard rules. In a searched DELETE or UPDATE, the SELECT privilege is required. For details, see "DELETE" on page 357 or "UPDATE" on page 477.
		CURRENT RULES special register	Whether the statements ALTER TABLE, GRANT and REVOKE are processed with DB2 rules or SQL standard rules. For details, see "CURRENT RULES" on page 81. In a searched DELETE or UPDATE, the SELECT privilege is required. For details, see "DELETE" on page 357 or "UPDATE" on page 477.

For further details on precompiler options, see Section 5 of *Application Programming and SQL Guide*. For more details on bind options, see Chapter 2 of *Command Reference*.

Precompiler Options for Dynamic Statements

Generally, dynamic statements use the application programming defaults specified
 # on installation panel DSNTIPF. However, if installation parameter DYNRULS is NO
 # and DYNAMICRULES(BIND) applies, 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 installation parameter DYNRULS 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, string delimiters, and mixed
 # data, see page 121, 122, and 123.

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 application 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:

Options Affecting SQL

```
#           – If DYNAMICRULES(RUN) 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.
#
#           – If DYNAMICRULES(BIND) applies and installation parameter
#           DYNRULS is YES, the decimal point is the value of field DECIMAL
#           POINT IS on installation panel DSNTIPF.
#
#           If DYNAMICRULES(BIND) applies and installation parameter
#           DYNRULS 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 constant, a comma intended as a separator must be followed by 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.

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.

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 application requester
 - The value of the field SQL STRING DELIMITER on installation panel DSNTIPF at the DB2 where the package is bound
- Otherwise:

- 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) 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.
 - If DYNAMICRULES(BIND) applies and installation parameter DYNRULS is YES, the decimal point is the value of field SQL STRING DELIMITER.
- If DYNAMICRULES(BIND) applies and installation parameter DYNRULS 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 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. The value YES indicates that character strings can contain a mixture of SBCS and DBCS characters. The value NO indicates that they cannot. 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 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) applies, field MIXED DATA is used.
- If DYNAMICRULES(BIND) applies and installation parameter DYNRULS is NO, the precompiler option is used. If DYNRULS is YES, field MIXED DATA is used instead.

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 92. A

Options Affecting SQL

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: EBCDIC and ASCII ones for SBCS data, EBCDIC and ASCII ones for MIXED data, and EBCDIC and ASCII ones for GRAPHIC (DBCS) data. The CCSIDs for SBCS and DBCS data are derived from the value of the ASCII CODED CHAR SET or EBCDIC CODED CHAR SET field, whose value is a CCSID for MIXED data. 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 system CCSIDs are the EBCDIC and ASCII ones 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 63. 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 65 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)¹⁵ indicates that conformance with the SQL standard is intended.

When a program is precompiled with the STDSQL(YES) option, the following rules apply:

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
```

¹⁵ STDSQL(86) is a synonym, but STDSQL(YES) should be used.

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 *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 DSN8510.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 OF 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 11. The NOFOR Precompiler Option

When NOFOR is in effect	When NOFOR is not in effect
The use of the FOR UPDATE OF 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 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 OF clause must be specified and must identify every column specified in a positioned update.
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.

Chapter 4. Functions

Column Functions	130
AVG	131
COUNT	132
MAX	133
MIN	134
SUM	135
Scalar Functions	136
CHAR	137
COALESCE	139
DATE	140
DAY	141
DAYS	142
DECIMAL	143
DIGITS	144
FLOAT	145
HEX	146
HOUR	147
INTEGER	148
LENGTH	149
MICROSECOND	150
MINUTE	151
MONTH	152
NULLIF	153
SECOND	154
STRIP	155
SUBSTR	157
TIME	159
TIMESTAMP	160
VALUE	162
VARGRAPHIC	164
YEAR	166

A *function* is an operation denoted by a function name followed by one or more operands which are enclosed in parentheses. The operands of functions are called *arguments*. Most functions have a single argument that is specified by an *expression*. The result of a function is a single value derived by applying the function to the result of the expression.

function:	
AVG function	
COUNT function	
MAX function	
MIN function	
SUM function	
CHAR function	
COALESCE function	
DATE function	
DAY function	
DAYS function	
DECIMAL function	
DIGITS function	
FLOAT function	
HEX function	
HOUR function	
INTEGER function	
LENGTH function	
MICROSECOND function	
MINUTE function	
MONTH function	
NULLIF function	
SECOND function	
STRIP function	
SUBSTR function	
TIME function	
TIMESTAMP function	
VALUE function	
VARGRAPHIC function	
YEAR function	

Functions are classified as *scalar functions* or *column functions*. The argument of a column function is a set of values. An argument of a scalar function is a single value.

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 can be used. Additional restrictions apply to the use of column functions as specified in the following section and in “Chapter 5. Queries” on page 167.

Column Functions

The following information applies to all column functions, except for the COUNT(*) variation of the COUNT function.

The argument of a column function is a set of values derived from one or more columns. The scope of the set is a group or an intermediate result table as explained in “Chapter 5. Queries” on page 167. 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 DSN8510.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.

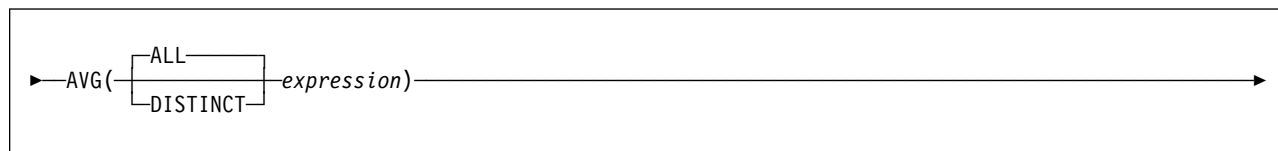
An expression in a column function must include a column name and must not include a column function. A column function can be used in a WHERE clause only if that clause is part of a subquery of a HAVING clause and the column name specified in the expression is a correlated reference to a group. If the expression includes more than one column name, each column name must be a correlated reference to the same group.

The result of the COUNT function cannot be the null value. As specified in the description of AVG, MAX, MIN, and SUM, 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 MAX and MIN inherits the field procedure.

Following in alphabetic order is a definition of each of the column functions.

AVG



The AVG function returns the average of a set of numbers.

The argument values must be numbers 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 a null value. Otherwise, the result is the average value of the set. If the type of the result is 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 DSN8510.EMP.

```
EXEC SQL SELECT AVG(SALARY)
        INTO :AVERAGE
        FROM DSN8510.EMP
        WHERE WORKDEPT = 'D11';
```

COUNT

COUNT

▶ COUNT (~~DISTINCT~~ *expression*) ▶

The COUNT function returns the number of rows or values in a set of rows or values.

#

The argument values can be any values other than character strings with a maximum length greater than 255 or graphic strings with a maximum length greater than 127. The result of the function must be within the range of large integers and cannot be null.

The data type of the result is INTEGER.

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.

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.

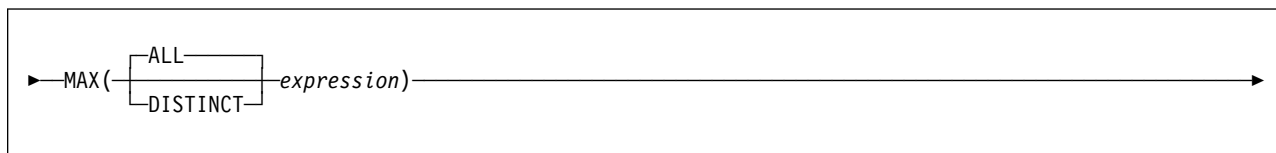
Example 1: Set the integer host variable FEMALE to the number of females represented in the sample table DSN8510.EMP.

```
EXEC SQL SELECT COUNT(*)
        INTO :FEMALE
        FROM DSN8510.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 DSN8510.EMP
        WHERE SEX = 'F';
```

MAX



The MAX function returns the maximum value in a set of values.

#

The argument values can be any values other than character strings with a maximum length greater than 255 or graphic strings with 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 a 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 DSN8510.EMP.

```
EXEC SQL SELECT MAX(SALARY) / 12
        INTO :MAX_SALARY
        FROM DSN8510.EMP;
```

Example 2: Find the surname that comes last in the collating sequence for the employees represented in the sample table DSN8510.EMP. Set the VARCHAR(15) variable LAST_NAME to that surname.

```
EXEC SQL SELECT MAX(LASTNAME)
        INTO :LAST_NAME
        FROM DSN8510.EMP;
```

MIN

MIN



The MIN function returns the minimum value in a set of values.

#

The argument values can be any values other than character strings with a maximum length greater than 255 or graphic strings with 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 a 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.

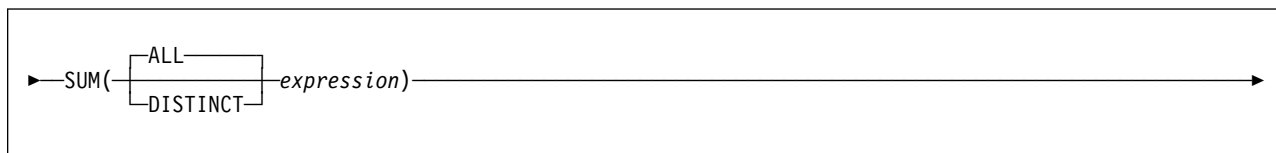
Example 1: Set the DECIMAL(15,2) variable MIN_SALARY to the minimum monthly salary of the employees represented in the sample table DSN8510.EMP.

```
EXEC SQL SELECT MIN(SALARY) / 12
        INTO :MIN_SALARY
        FROM DSN8510.EMP;
```

Example 2: Find the surname that comes first in the collating sequence for the employees represented in the sample table DSN8510.EMP. Set the VARCHAR(15) variable FIRST_NAME to that surname.

```
EXEC SQL SELECT MIN(LASTNAME)
        INTO :FIRST_NAME
        FROM DSN8510.EMP;
```


SUM



The SUM function returns the sum of a set of numbers.

The argument values must be numbers 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 a 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 INTEGER variable INCOME to the total income from all sources (salaries, commissions, and bonuses) of the employees represented in sample table DSN8510.EMP. If DEC31 is not in effect, then, because all three columns are DECIMAL(9,2), the resultant sum is DECIMAL(15,2).

```
EXEC SQL SELECT SUM(SALARY+COMM+BONUS)
        INTO :INCOME
        FROM DSN8510.EMP;
```

Scalar Functions

A scalar function can be used wherever an expression can be used. The restrictions on the use of column functions do not apply to scalar functions because a scalar function is applied to a single value rather than a set of values. The argument of a scalar function can be a function. 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 DSN8510.EMP. To obtain the ages, the scalar function YEAR is applied to the expression:

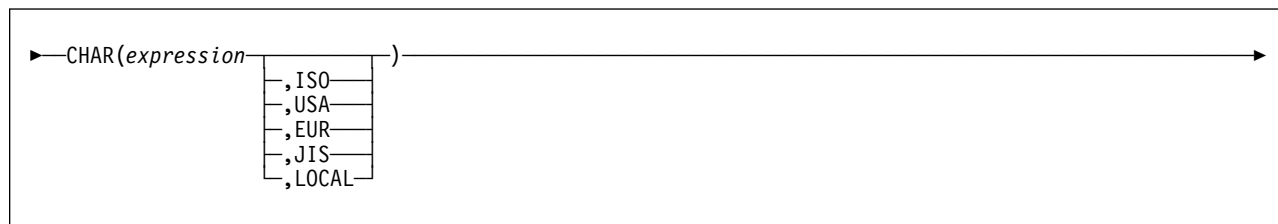
```
CURRENT DATE - BIRTHDATE
```

in each row of DSN8510.EMP for which the employee represented is in department D11:

```
SELECT EMPNO, LASTNAME, YEAR(CURRENT DATE - BIRTHDATE)
FROM DSN8510.EMP
WHERE WORKDEPT = 'D11';
```

Following in alphabetic order is the definition of each of the scalar functions.

CHAR



The CHAR function returns a string representation of a datetime value or a decimal number.

The first argument must be a date, time, timestamp, or decimal number. The second argument, if applicable, is the name of a datetime format.

The result of the function is a fixed-length character string. The CCSID of the result is the system EBCDIC or ASCII CCSID for SBCS data, depending on the encoding of other data in the SQL statement. The encoding scheme of the first argument, EBCDIC or ASCII, determines which CCSID is used. 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 other rules depend on the data type of the first argument:

If the first argument is a date, the result is the character string representation of that date in the format specified by the second argument. If the second argument is omitted, the format is specified by the DATE precompiler option, if one is provided, or else by the field DATE FORMAT on installation panel DSNTIP4. If LOCAL is the format to be used, the length of the result is specified by the field LOCAL DATE LENGTH on installation panel DSNTIP4. 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.

If the first argument is a time, the result is the character string representation of that time in the format specified by the second argument. If the second argument is omitted, the format is specified by the TIME precompiler option, if one is provided, or else by the field TIME FORMAT on installation panel DSNTIP4. If LOCAL is the format to be used, the length of the result is specified by the field LOCAL TIME LENGTH on installation panel DSNTIP4. 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.

If the first argument is a timestamp, the result is the character string representation of the timestamp. The length of the result is 26. The second argument is not applicable and must not be specified.

If the first argument is a decimal number, the result is a fixed-length character string representation of that number. The second argument is not applicable and must not be specified.

CHAR

The first character of the result is a minus sign if the argument is negative. Otherwise, the first character is blank. The result includes a decimal point and p digits, where p is the precision of the argument. The length of the result is $2+p$.

The decimal point can be a period or a comma. For details on what governs the choice, see the discussion of decimal point representation in “Options Affecting SQL” on page 119.

Example 1: HIREDATE is a DATE column in the sample table DSN8510.EMP. When it represents 15 December 1976 (as it does for employee 140):

```
EXEC SQL SELECT CHAR(HIREDATE, USA)
        INTO :DATESTRING
        FROM DSN8510.EMP
        WHERE EMPNO = '000140';
```

returns the string value '12/15/1976' in character-string variable DATESTRING.

Example 2: HOURS is a DECIMAL(6,0) variable with 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: The following SQL statement sets the CHAR(33) variable AVERAGE to the character string representation of the average employee salary:

```
EXEC SQL SELECT CHAR(AVG(SALARY))
        INTO :AVERAGE
        FROM DSN8510.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.

COALESCE



The COALESCE function is a synonym for the VALUE function. It returns the first argument that is not null. Use COALESCE to conform to the SQL standard.

For a description and examples, see “VALUE” on page 162.

DATE

DATE

▶—DATE(*expression*)—▶

The DATE function returns a date derived from its argument.

The argument must be a timestamp, a date, a positive number less than or equal to 3652059, a valid string representation of a date, or a character string of length 7. (Valid formats for string representations for dates are listed in Table 2 on page 64.)

If the argument is a character string of length 7, it must represent a valid date in the form *yyyynn*, where *yyyy* are digits denoting a year, and *nnn* are digits between 001 and 366 denoting a day of that year.

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 character string, the result is the date represented by the character string. If the CCSID of the string is not the same as the corresponding default CCSID at the application 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)
```

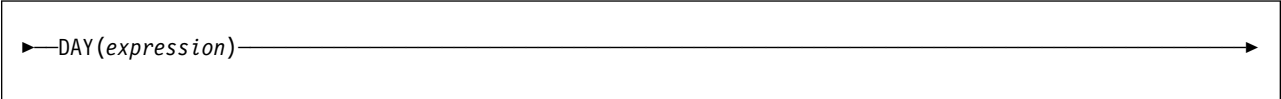
returns the internal representation of 2 March 1989.

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(*expression*) ◄

The DAY function returns the day part of its argument.

The argument must be a date, timestamp, date duration, or timestamp duration.

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 or a timestamp, 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 variable DAYVAR to the day of the month on which employee 140 in the sample table DSN8510.EMP was hired.

```
EXEC SQL SELECT DAY(HIREDATE)
        INTO :DAYVAR
        FROM DSN8510.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 respectively the dates 15 January 2000 and 31 December 1999. Then, for the given row:

```
DAY(DATE1 - DATE2)
```

returns the value 15.

DAYS

DAYS

►—DAYS(*expression*)—►

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. (Valid formats for string representations for dates are listed in Table 2 on page 64.)

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.

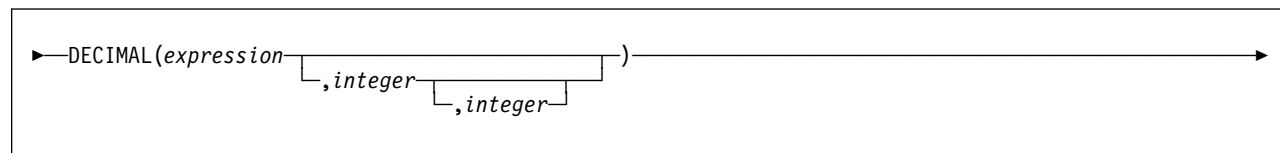
Example 1: Set the INTEGER variable DAYSVAR to the number of days employee 140 had been with the enterprise. Sample table DSN8510.EMP represents the number of days up to and including 8 January 1990.

```
EXEC SQL SELECT DAYS('1990-01-08') - DAYS(HIREDATE) + 1
          INTO :DAYSVAR
          FROM DSN8510.EMP
          WHERE EMPNO = '000140';
```

Example 2: Set the INTEGER variable DAYOFWEEK to the numerical day of the week that employee 140 was hired, where 1 represents Sunday, 2 represents Monday, ... 7 represents Saturday. HIREDATE is a column of date type in sample table DSN8510.EMP.

```
EXEC SQL SELECT (DAYS(HIREDATE) - DAYS(HIREDATE)/7 * 7) + 1
          INTO :DAYOFWEEK
          FROM DSN8510.EMP
          WHERE EMPNO = '000140';
```


DECIMAL



The DECIMAL function returns a decimal representation of a value.

The first argument must be a character string or a number. The second argument, if specified, must range in value from 1 to 31. The third argument, if specified, must range in value from 1 to p , where p is the value of the second argument.

If the second argument is omitted, its value depends on the data type of the first argument. The default is:

- 5 if the first argument is a small integer
- 11 if the first argument is a large integer
- 15 in all other cases

If the third argument is omitted, its value is zero.

#

If the first argument is a string, its length attribute must not exceed 255. Leading and trailing blanks are eliminated from the string. The resulting substring must conform to the rules for forming an SQL integer or decimal constant.

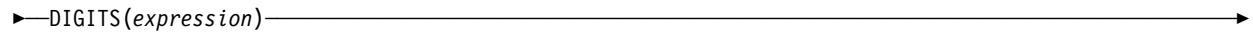
The data type of the result is `DECIMAL(p , s)`, where p and s are the second and third arguments. If the first argument can be null, the result can be null; if the first argument is null, the result is null.

Assume that N denotes the number or numeric constant specified as the first argument. The result of the function is the number that would occur if N were assigned to a decimal column with precision p and scale s . Accordingly, an error occurs if the number of significant digits required to represent the whole part of the number is greater than $p-s$.

Example: Represent the average salary of the employees in `DSN8510.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 DSN8510.EMP;
```

DIGITS



The DIGITS function returns a character string representation of its argument.

The argument must be an integer or a decimal number.

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

The CCSID of the result is the corresponding EBCDIC or ASCII CCSID for SBCS data defined at the server during system installation. The encoding scheme of the argument, EBCDIC or ASCII, determines which CCSID is used.

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;
```

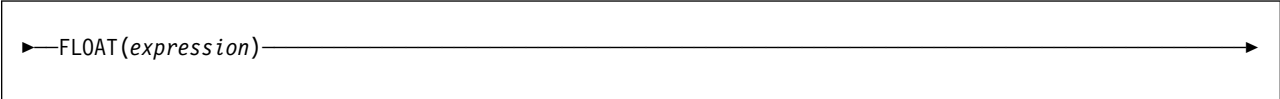
Example 2: Assume that COLUMNX has the data type DECIMAL(6,2), and that one of its values is -6.28. Then, for this value:

```
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.

FLOAT



▶—FLOAT(*expression*)—▶

The FLOAT function returns a floating-point representation of its argument.

The argument must be a number.

The result of the function is a double precision floating-point number. If the argument can be null, the result can be null; if the argument is null, the result is the null value.

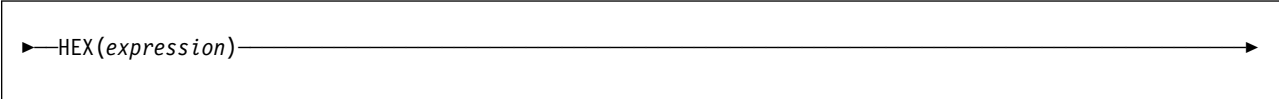
The result is the number that would occur if the argument were assigned to a double precision floating-point column or variable.

Example: Using the sample table in DSN8510.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 eliminate the possibility of out-of-range results, FLOAT is applied to SALARY so that the division is carried out in floating-point:

```
SELECT EMPNO, FLOAT(SALARY)/COMM
FROM DSN8510.EMP
WHERE COMM > 0;
```

HEX

HEX



▶—HEX(*expression*)—▶

The HEX function returns a hexadecimal representation of its argument.

#

The argument can be any value other than a character 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 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, the result is subject to the restrictions that apply to long strings.

The CCSID of the result is the corresponding EBCDIC or ASCII CCSID for SBCS data defined at the server during system installation. The encoding scheme of the argument, EBCDIC or ASCII, determines which CCSID is used.

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 HOUR function returns the hour part of its argument.

The argument must be a time, timestamp, time duration, or timestamp duration.

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 or timestamp, 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. 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 after the noon hour.

```
SELECT *
FROM CLASSES
WHERE HOUR(STARTTM) > 12;
```

INTEGER

INTEGER

▶—INTEGER(*expression*)—▶

The INTEGER function returns an integer representation of its argument.

The argument must be a number.

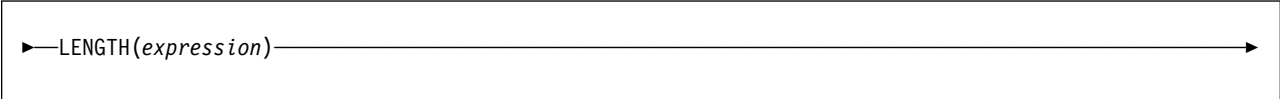
The result of the function is a large integer. If the argument can be null, the result can be null; if the argument is null, the result is the null value.

The result is the 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.

Example: Find the average salary of the employees in department 'A00', rounding the result to the nearest dollar.

```
SELECT INTEGER(AVG(SALARY)+.5)
FROM DSN8510.EMP
WHERE WORKDEPT = 'A00';
```

LENGTH



▶—LENGTH(*expression*)—▶

The LENGTH function returns the length of its argument.

The argument can be any value.

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 DBCS characters. The length of all other values is the number of bytes used to represent the value:

- The length of the string for character strings
- 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

Example 1: Assume that FIRSTNAME is a VARCHAR(12) column that contains 'ETHEL' for employee 280. The following query:

```
SELECT LENGTH(FIRSTNAME)
FROM DSN8510.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.

MICROSECOND

MICROSECOND

► MICROSECOND(*expression*) ◄

The MICROSECOND function returns the microsecond part of its argument.

The argument must be a timestamp or timestamp duration.

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:

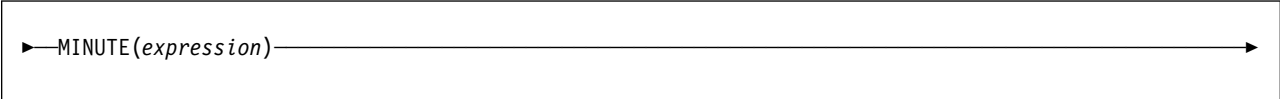
If the argument is a timestamp, the result is the microsecond part of the value, which is an integer between 0 and 999999.

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;
```


MINUTE



▶—MINUTE(*expression*)—▶

The MINUTE function returns the minute part of its argument.

The argument must be a time, timestamp, time duration, or timestamp duration.

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 or a timestamp, 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;
```

MONTH

MONTH



▶ MONTH(*expression*)

The MONTH function returns the month part of its argument.

The argument must be a date, timestamp, date duration, or timestamp duration.

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 or a timestamp, 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 DSN8510.EMP for employees who were born in May:

```
SELECT * FROM DSN8510.EMP
WHERE MONTH(BIRTHDATE) = 5;
```

NULLIF

```
▶—NULLIF—(—expression—,—expression—)————▶
```

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 “Assignment and Comparison” on page 65). The attributes of the result are the attributes of the first argument.

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 a 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)
```

SECOND

SECOND

▶—SECOND(*expression*)—▶

The SECOND function returns the seconds part of its argument.

The argument must be a time, timestamp, time duration, or timestamp duration.

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 or timestamp, 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)
```

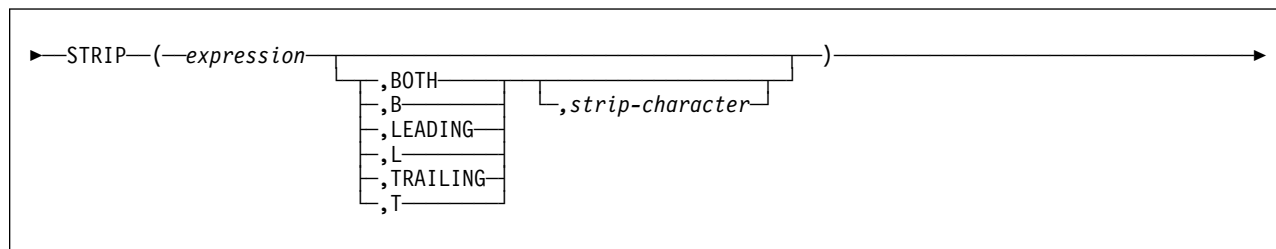
returns the value 45.

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.

STRIP



The STRIP function removes blanks or another specified character from the end, at the beginning, or at both ends of a string expression.

The first argument must be a string expression.

If you specify a second argument, it indicates whether characters are removed from the end or beginning of the string. If you do not specify a second argument, blanks are removed from both the end and the beginning of the string.

The third argument is a single-character constant that indicates the SBCS or DBCS character that is to be removed. If the first argument is a DBCS graphic or DBCS-only string, the third argument must be a graphic constant consisting of a single DBCS character. 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 DBCS 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.
- The default strip character is an SBCS blank. If the data is encoded in ASCII, then X'20' represents a blank. Otherwise, X'40' represents an EBCDIC 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 bytes removed. If all characters are stripped, 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 the host variable HELLO of type CHAR(9) has a value of

```
' Hello':
```

```
STRIP(:HELLO)
```

```
results in: 'Hello'.
```

```
STRIP(:HELLO,TRAILING)
```

```
results in: ' Hello'.
```

STRIP

```
|           Example 2: Assume the host variable BALANCE of type CHAR(9) has a value of  
|           '000345.50':  
|           STRIP(:BALANCE,L,'0')  
|           results in: '345.50'
```

SUBSTR

► SUBSTR(*string*, *start*, *length*)

The SUBSTR function returns a substring of a string.

string

Denotes an expression that specifies the string from which the result is derived. *string* must be a character string or a graphic string. If *string* 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.

A substring of *string* is zero or more contiguous characters of *string*. If *string* is a graphic string, a character is a DBCS character. If *string* is a character 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

Denotes an expression that specifies the position of the first character of the result. It must be a positive binary integer that is not greater than the length attribute of *string*. (The length attribute of a varying-length string is its maximum length.) A value of 1 would indicate that the first character of the substring is the first character of *string*.

length

Denotes an expression that specifies the length of the result. If specified, *length* must be a binary integer in the range 0 to *n*, where *n* is equal to L-S+1, L is the length attribute of *string*, and S is the value of *start*. *length* must not, however, be the integer constant 0. If *string* is a varying-length string and if *length* is explicitly specified, *string* is effectively padded on the right with the necessary number of blank characters so that the specified substring of *string* always exists. If *string* is a fixed-length string, omission of *length* is an implicit specification of LENGTH(*string*) - *start* + 1, which is the number of characters from the character specified by *start* to the last character of *string*. If *string* is a varying-length string, omission of *length* is an implicit specification of zero or LENGTH(*string*) - *start* + 1, whichever is greater.

#

If *length* is explicitly specified by an integer constant that is 255 or less, the result is a fixed-length string. If *length* is not explicitly specified, but *string* is a fixed-length string and *start* is an integer constant, the result is a fixed-length string. In all other cases, the result is a varying-length string with a maximum length that is the same as the length attribute of *string*. The result is subject to the restrictions that apply to long strings if its maximum length exceeds 255. These restrictions also apply if it is a graphic string whose maximum length exceeds 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*.

SUBSTR

Example 1: FIRSTNME is a VARCHAR(12) column in the sample table DSN8510.EMP. One of its values is the 5-character string 'MAUDE'. When FIRSTNME has this value:

```
SUBSTR(FIRSTNME,2,3)
```

returns 'AUD'.

```
SUBSTR(FIRSTNME,2)
```

returns 'AUDE'.

```
SUBSTR(FIRSTNME,2,6)
```

returns 'AUDE' followed by two blanks.

```
SUBSTR(FIRSTNME,6)
```

returns a string of length zero.

```
SUBSTR(FIRSTNME,6,4)
```

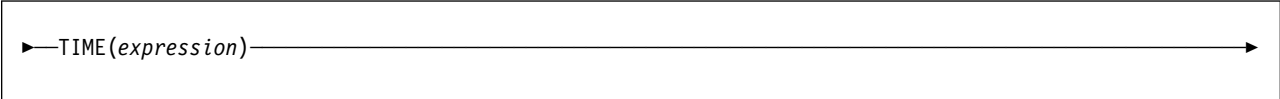
returns four blanks.

Example 2: Sample table DSN8510.PROJ contains a character string column named PROJNAME. Select all rows from that table for which the string in PROJNAME begins with 'W L PROGRAM'.

```
SELECT * FROM DSN8510.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.

TIME



►—TIME(*expression*)—►

The TIME function returns a time derived from its argument.

The argument must be a timestamp, a time, or a valid string representation of a time. (Valid formats for string representations for times are listed in Table 3 on page 64.)

The result of the function is a time. 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 time part of the timestamp.

If the argument is a time, the result is that time.

If the argument is a character string, the result is the time represented by the character string. If the CCSID of the string is not the same as the corresponding default CCSID at the application 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

The diagram shows the syntax for the TIMESTAMP function: `TIMESTAMP(expression [, expression])`. The first argument is enclosed in a box, and the second argument is optional, indicated by a bracket and a comma. Arrows point from the left and right of the function name to the arguments.

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 character string of length 14. (String representations for a timestamp are described in “String Representations of Datetime Values” on page 63.)

A character string of length 8 is assumed to be a System/370 Store Clock value.

A character 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. (Valid formats for string representations for dates and times are listed in Table 2 on page 64 and Table 3 on page 64.)

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 character string, the result is the timestamp represented by that character string. The timestamp represented by a string of length 14 has a microsecond part of zero. The interpretation of a character string as a Store Clock value will yield a timestamp with a year between 1900 to 2042 as described in *IBM System/370 ESA Principles of Operation*.

If an argument is a character string with a CCSID that is not the same as the corresponding default CCSID at the application server, the string is first converted to that CCSID.

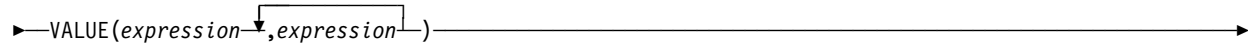
Example: Assume that table TABLEX contains a DATE column named DATECOL and a TIME column named TIMECOL. Assume also that for some row in the table, DATECOL represents 25 December 1988 and TIMECOL represents 17 hours, 12 minutes, and 30 seconds after midnight. Then, for this row:

```
TIMESTAMP(DATECOL, TIMECOL)
```

returns the value '1988-12-25-17.12.30.000000'.

VALUE

VALUE



▶—VALUE(expression, expression)—▶

The VALUE function returns the first argument that is not null. COALESCE can be used as a synonym for VALUE.

The arguments must be compatible. Character string arguments are not compatible with datetime values. Thus, if any argument is a character string, all arguments must be character strings; if any argument is a date, all arguments must be dates; if any argument is a number, all arguments must be numbers, and so forth.

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 “Data Type Rules for UNION and the VALUE Function” on page 183. If the VALUE 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 VALUE function can also handle a subset of the functions provided by CASE expressions. The result of using VALUE(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 VALUE(SCORE1,0) + SCORE2 > 100;
```

Example 2: Assume that a table named DSN8510.EMP contains a DATE column named HIREDATE, and that nulls are allowed for this column. The following query selects all rows in DSN8510.EMP for which the date in HIREDATE is either unknown (null) or earlier than 1 January 1960.

```
SELECT * FROM DSN8510.EMP
WHERE VALUE(HIREDATE,DATE('1959-12-31')) < '1960-01-01';
```

In this case, the predicate VALUE(HIREDATE, '1959-12-31') would be invalid because strings and datetime values are incompatible and strings in a VALUE predicate are not converted to datetime values. Applying the DATE function to the string does the needed conversion.

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 (or VALUE) function allows the two join columns to be combined into a single column, which enables the results to be ordered.

VARGRAPHIC

►—VARGRAPHIC(*expression*)—►

The VARGRAPHIC function returns a graphic string representation of a character string.

The argument must be an EBCDIC-encoded character string with a maximum length no greater than 255. The argument need not be mixed data, but any occurrences of X'0E' and X'0F' in the string must conform to the rules for mixed data. (See “Character Strings” on page 57 for these rules.)

The result of the function is a varying-length graphic string. If the argument can be null, the result can be null; if the argument is null, the result is the null value. If the argument is an empty string or has the value X'0E0F', the result is an empty string.

The length attribute of the result is equal to the length attribute of the argument. The actual length of the result is the number of characters in the argument, excluding shift characters. Because the result is a graphic string, its length and length attribute are measured in double-bytes, not single-bytes. If the length attribute is greater than 127, the result is a long string and is therefore subject to the restrictions that apply to long strings.

The CCSID of the result is the system EBCDIC CCSID for GRAPHIC data. If there is no system EBCDIC CCSID for GRAPHIC data, the CCSID of the result is X'FFFE'.

Before deriving the result, the argument might be converted to the native form of mixed data. Let M denote the system EBCDIC CCSID for mixed data. The argument is not converted if any of the following are 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 EBCDIC 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 converted to a new string, S, by converting its characters to the coded character set identified by M. If there is no system EBCDIC CCSID for mixed data, conversion is to the coded character set identified by the system EBCDIC CCSID for SBCS data.

The result is derived from S as follows:

- Each shift character (X'0E' or X'0F') is removed
- Each double-byte character remains as it 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'. Whether an equivalent

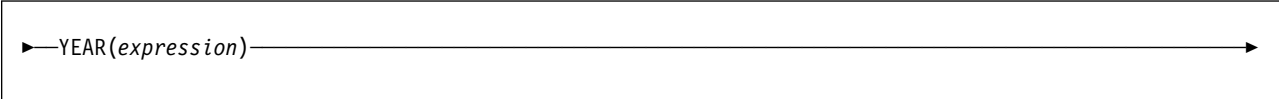
character exists depends on M. If there is no system EBCDIC CCSID for mixed data, the DBCS equivalent of X'xx' is X'42xx', except for X'40', whose DBCS equivalent is X'4040'.

Example: Assume that GRPHCOL is a GRAPHIC column in the table TABLEX, and that MIXEDSTRING is a character-string variable containing mixed data. For various rows in TABLEX, the value of GRPHCOL is being replaced with the value of MIXEDSTRING through the use of a positioned UPDATE statement. Before an update can be made, the current value of MIXEDSTRING must be converted to a GRAPHIC string. Within the UPDATE statement, this can be done using the VARGRAPHIC function:

```
EXEC SQL UPDATE TABLEX
  SET GRPHCOL = VARGRAPHIC(:MIXEDSTRING)
  WHERE CURRENT OF CRSNAME;
```

YEAR

YEAR



►—YEAR(*expression*)—►

The YEAR function returns the year part of its argument.

The argument must be a date, timestamp, date duration, or timestamp duration.

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 or a timestamp, 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 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 DSN8510.EMP select all rows for employees who were born in 1941.

```
SELECT *  
FROM DSN8510.EMP  
WHERE YEAR(BIRTHDATE) = 1941;
```

Chapter 5. Queries

Authorization	169
subselect	170
select-clause	171
from-clause	174
where-clause	177
group-by-clause	177
having-clause	178
Examples of subselects	178
fullselect	183
Data Type Rules for UNION and the VALUE Function	183
Character Conversion in Unions and Concatenations	185
Selecting the Result CCSID	185
Examples of fullselects	187
select-statement	188
order-by-clause	188
update-clause	189
read-only-clause	190
optimize-for-clause	190
with-clause	190
Examples of select statements	191

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.

There is another SQL statement called SELECT INTO, which is described in “SELECT INTO” on page 462. SELECT INTO is not a subselect, fullselect, or a select-statement.

Authorization

The privilege set 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)

If the *select-statement* is part of a DECLARE CURSOR statement, the privilege set is the privileges held by the authorization ID of the owner of the plan or package.

For dynamically prepared statements, the privilege set depends on the value of bind option DYNAMICRULES:

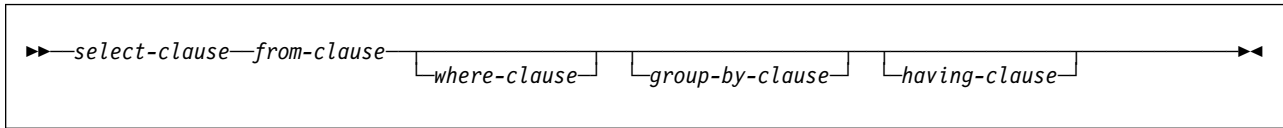
DYNAMICRULES(RUN) The privilege set is the union of the privilege sets held by each authorization ID of the process.

DYNAMICRULES(BIND) The privilege set is the privileges held by the authorization ID of the owner of the plan or package.

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 341 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



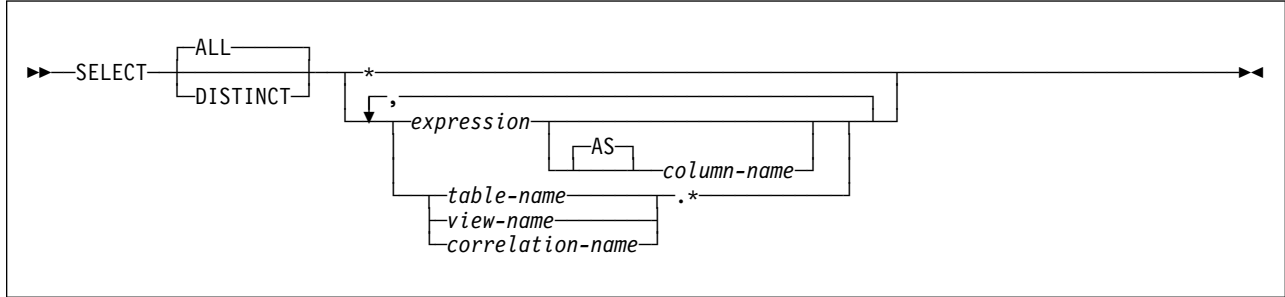
The *subselect* is a component of the fullselect, the CREATE VIEW statement, and the INSERT statement. It is also a component of certain predicates which, in turn, are components of a subselect. A subselect that is a component of a predicate is called a subquery.

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 sequence of the (hypothetical) operations is:

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 the only clauses specified are **SELECT**, **FROM**, and **WHERE**, 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.

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 described in Expressions, which begins on page 92. 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. 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:

If neither GROUP BY nor HAVING is used:

- The select list must not include column functions, or it must be entirely a list of column functions.
- 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 is a list of 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 consists of zero 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.
- 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 179 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 but COUNT
- 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.

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 12 (Page 1 of 2). Data types of result columns

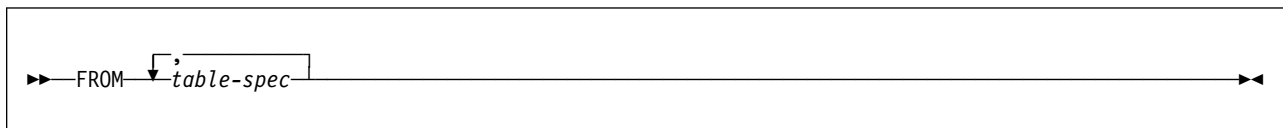
When the expression is...	The data type of the result column is...
The name of any numeric column	The same as the data type of the column, with the same precision and scale for decimal columns.
An integer constant	INTEGER.
A decimal or floating-point constant	The same as the data type of the constant, 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 92.
Any function	(See “Chapter 4. Functions” on page 127 to determine the data type of the result.)
The name of any string column	The same as the data type of the column, with the same length attribute.

subselect

Table 12 (Page 2 of 2). Data types of result columns

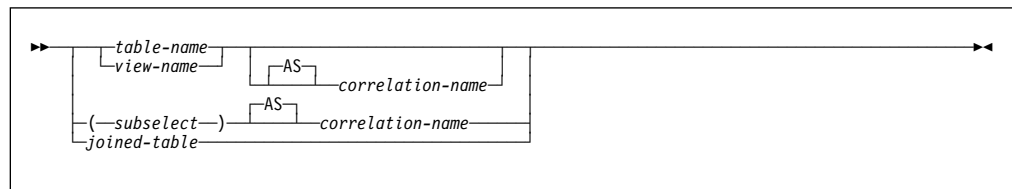
When the expression is...	The data type of the result column is...
The name of any string host variable	The same as the data type of the variable, with a length attribute equal to the length of the variable. The result is a varying-length character string if the data type of the host variable is not 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.

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.
- A *subselect* 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.
- If a *joined-table* is specified, the intermediate result table is the result of one or more join operations as explained below.

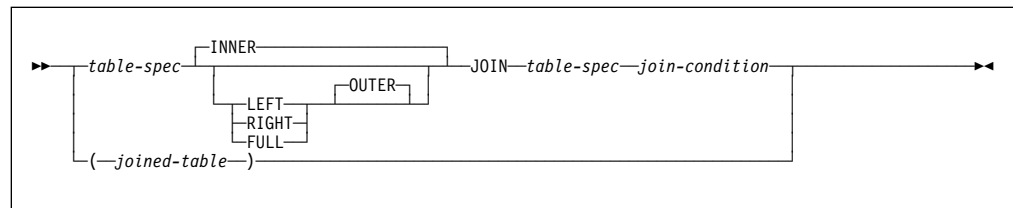
#

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. Each table or underlying table of each view that is identified must have the same encoding scheme—either all ASCII or all EBCDIC.. 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 *correlation-name* is defined as a designator of the intermediate result table specified by the immediately preceding *table-spec*. A correlation name must be specified for a nested table expression.

An exposed name is a correlation name or a 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 or view. For more information, see “Column Name Qualifiers in Correlated References” on page 86.

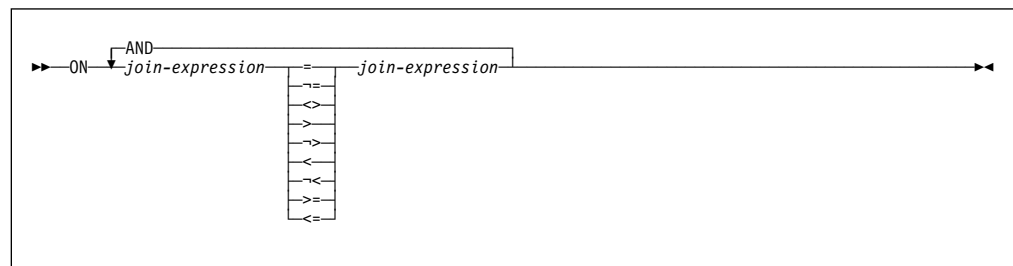
joined-table



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 or RIGHT OUTER JOIN is performed can affect the result.

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

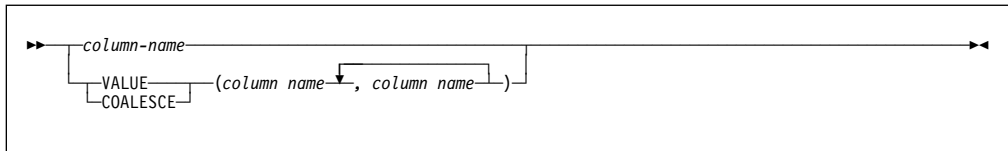


A *join-condition* is a search condition in which predicates can be combined only with AND and each predicate has the form 'expression operator expression'. The

'=' operator is the only operator allowed for a FULL OUTER JOIN or a FULL JOIN.

One expression of the predicate must reference only columns of one of the operand tables of the associated join operator, and the other expression of the predicate must reference only columns of the other operand table. Before this rule is applied, column references are resolved using the rules for resolution of column name qualifiers specified in "Resolution of Column Name Qualifiers" on page 87. As in any predicate, the values of the expressions must be comparable.

join-expression



A *join-expression* must include a column name. Only columns and the VALUE and COALESCE functions are allowed in the expression. VALUE and COALESCE are allowed only when the join operator is FULL JOIN or FULL OUTER JOIN.

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 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

▶▶ GROUP BY *column-name* ▶▶

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.

GROUP BY must not be used in a subquery of a basic predicate.

having-clause

▶—HAVING—*search-condition*—▶

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 function¹⁶, or
- 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 DSN8510.EMP.

```
SELECT * FROM DSN8510.EMP;
```

Example 2: Show the job code, maximum salary, and minimum salary for each group of rows of DSN8510.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 DSN8510.EMP
GROUP BY JOB
HAVING COUNT(*) > 1 AND MAX(SALARY) > 50000;
```

¹⁶ See “Chapter 4. Functions” on page 127 for restrictions that apply to the use of column functions.

Example 3: For each employee in department E11, get the following information from the table DSN8510.EMPPROJACT: 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 DSN8510.EMP:

```
SELECT EMPNO, ACTNO, CHAR(EMSTDATE,USA), CHAR(EMENDATE,USA)
FROM DSN8510.EMPPROJACT
WHERE EMPNO IN (SELECT EMPNO FROM DSN8510.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 DSN8510.EMP
GROUP BY WORKDEPT
HAVING MAX(SALARY) < (SELECT AVG(SALARY)
                      FROM DSN8510.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 DSN8510.EMP Q
GROUP BY WORKDEPT
HAVING MAX(SALARY) < (SELECT AVG(SALARY)
                      FROM DSN8510.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 DSN8510.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 have to retype the expression, see “Example 3:” on page 104 for CASE expressions.

#

Example 8: Get the employee number and employee name for all employees in table DSN8510.EMP, Order the results by the date of hire.

```
SELECT EMPNO, FIRSTNME, LASTNAME
FROM DSN8510.EMP
ORDER BY HIREDATE;
```

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			The PRODUCTS table		
PART	PROD#	SUPPLIER	PROD#	PRODUCT	PRICE
=====	=====	=====	=====	=====	=====
WIRE	10	ACWF	505	SCREWDRIVER	3.70
OIL	160	WESTERN_CHEM	30	RELAY	7.55
MAGNETS	10	BATEMAN	205	SAW	18.90
PLASTIC	30	PLASTIK_CORP	10	GENERATOR	45.75
BLADES	205	ACE_STEEL			

Example 9: 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:

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.

Example 10: 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 11: 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 12: 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 13: The result of “Example 10” (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

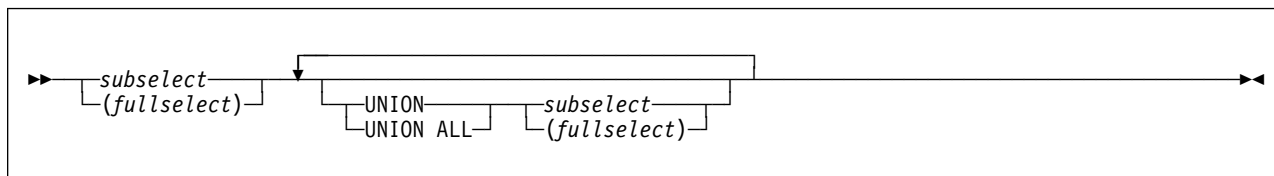
Example 14: For all parts that are used in product numbers less than 200, show the part, the part supplier, the product number, and the product name. Use a nested table expression.

```
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



fullselect specifies a result table. If UNION is not used, the result of the fullselect is the result of the specified subselect.¹⁷

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.

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. Table 13 on page 184 shows all valid combinations of operand columns and, for each combination, the description of the result column.

Data Type Rules for UNION and the VALUE Function

The following table shows the data type of the results for UNION and the VALUE function.

¹⁷ DB2 allows SELECT INTO as the operand of UNION. This is a deprecated feature with undefined results.

fullselect

Table 13. Result data types

When one operand is...	And the other operand is...	The result has the data type...
CHAR(x)	CHAR(y)	CHAR(z) where z=MAX(x,y)
VARCHAR(x)	CHAR(y) or VARCHAR(y)	VARCHAR(z) where z=MAX(x,y)
BIT data	Any subtype	BIT data
MIXED data	MIXED or SBCS	MIXED data
SBCS data	SBCS data	SBCS data
GRAPHIC(x)	GRAPHIC(y)	GRAPHIC(z) where z = MAX(x,y)
VARGRAPHIC(x)	VARGRAPHIC(y) or GRAPHIC(y)	VARGRAPHIC(z) where z = MAX(x,y)
DATE	DATE	DATE
TIME	TIME	TIME
TIMESTAMP	TIMESTAMP	TIMESTAMP
FLOAT(double)	Any numeric type	FLOAT(double)
FLOAT(single)	FLOAT(single)	FLOAT(single)
FLOAT(single)	DECIMAL, INTEGER, or SMALLINT	FLOAT(double)
DECIMAL(p,s)	DECIMAL(p',s')	DECIMAL(P,S) where P = MIN(31,MAX(s,s') + MAX(p-s,p'-s')) S = MAX(s,s')
DECIMAL(p,s)	INTEGER	DECIMAL(P,S) where P = MIN(31,s + MAX(p-s,11)) S = s
DECIMAL(p,s)	SMALLINT	DECIMAL(P,S) where P = MIN(31,s + MAX(p-s,5)) S = s
INTEGER	INTEGER	INTEGER
INTEGER	SMALLINT	INTEGER
SMALLINT	SMALLINT	SMALLINT

If neither operand column allows nulls, the result column does not allow nulls. Otherwise, the result column allows nulls. If the description of any operand column is not the same as the description of the result column, its values are converted to conform to the description of the result column.

The conversion operation is exactly the same as if the values were assigned to the result column. For example, if one operand column is CHAR(10) and the other operand column is CHAR(5), the result column is CHAR(10) and the values derived from the CHAR(5) column are padded on the right with five blanks. As another example, an error occurs if the whole part of a decimal number cannot be preserved.

Character Conversion in Unions and Concatenations

The SQL operations that combine strings are concatenation, UNION, and UNION ALL. Within an SQL statement, concatenation combines two or more strings into a new string. Within a fullselect, UNION and UNION ALL can combine two or more string columns resulting from the subselects into a 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 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.

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 14.

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 14. 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.

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 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 or SYSSTRINGS is used but contains no information about the CCSID pair. 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 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 DSN8510.EMP.

```
SELECT * FROM DSN8510.EMP;
```

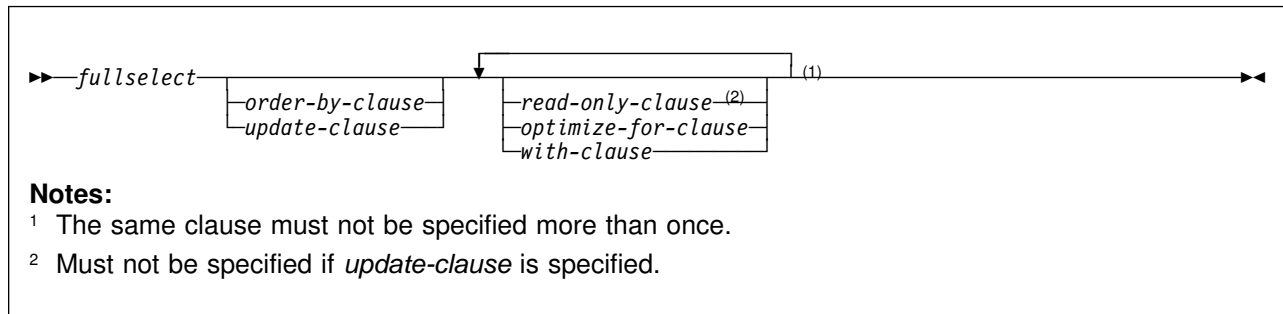
Example 3: Using sample tables DSN8510.EMP and DSN8510.EMPROJACT, list the employee numbers of all employees for which either of the following statements are true:

- Their department numbers begin with 'D'.
- They are assigned to projects whose project numbers begin with 'AD'.

```
SELECT EMPNO FROM DSN8510.EMP
WHERE WORKDEPT LIKE 'D%'
UNION
SELECT EMPNO FROM DSN8510.EMPPROJACT
WHERE PROJNO LIKE 'AD%';
```

The result is the union of two result tables, one formed from the sample table DSN8510.EMP, the other formed from the sample table DSN8510.EMPPROJACT. 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'.

select-statement

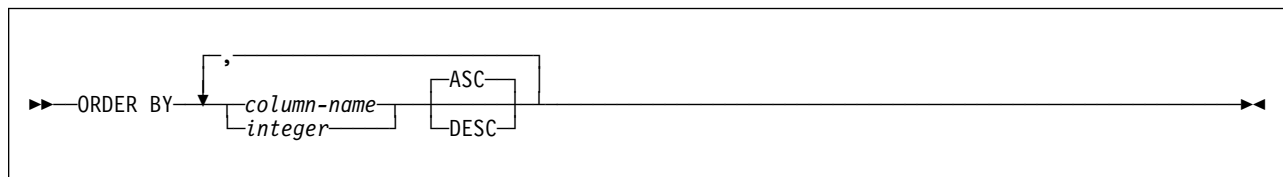


The *select-statement* is the form of a query that can be directly specified in a DECLARE CURSOR statement, or prepared and then referenced in a DECLARE CURSOR statement. It can also be issued interactively using SPUFI causing a result table to be displayed at your terminal. In any case, the table specified 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 or remote queries in which the server and requester are DB2 for OS/390, if a table is encoded as ASCII, the retrieved data is encoded in EBCDIC. For information on retrieving such data encoded in ASCII, see Section 6 of *Application Programming and SQL Guide*.

order-by-clause



The ORDER BY clause specifies an ordering of the rows of the result table. If a single column is identified, the rows are ordered by the values of that column. If more than one column is identified, the rows are ordered by the values of the first identified column, then by the values of the second identified column, and so on. A long string column must not be identified.

A named column can be identified by an integer or a column name. An unnamed column must be identified by an integer. A column is unnamed if the AS clause is not specified and it 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

Must unambiguously identify a column of the result table, with an exception. If the query is a subselect, *column-name* can identify the column name of a table, view, or nested table expression identified in the FROM clause and not in the result table when the subselect does not use:

- #
- #
- #
- DISTINCT in the select list
- Column functions in the select list
- GROUP BY

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.

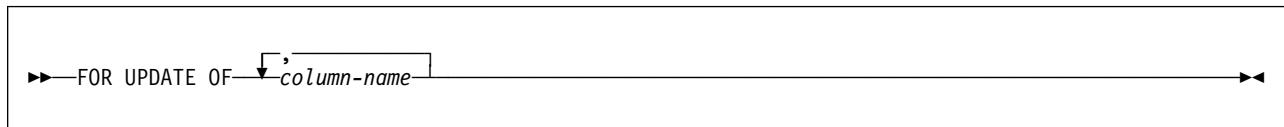
ASC

Uses the values of the column in ascending order. This is the default.

DESC

Uses the values of the column in descending order.

Ordering is performed in accordance with the comparison rules described in Chapter 3. 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 column have an arbitrary order. If the ORDER BY clause is not specified, the rows of the result table have an arbitrary order.

update-clause

The 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 347. The clause must also not be specified if a temporary table is referenced in the first FROM clause of the select-statement.

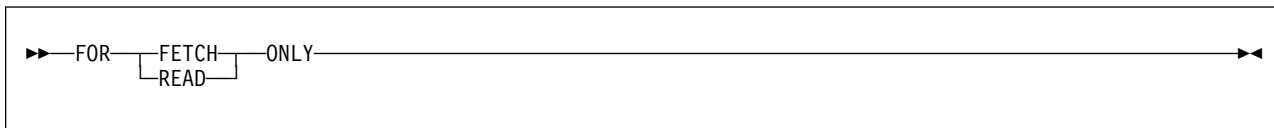
The declaration of a cursor referred to in a positioned UPDATE statement need not include an UPDATE clause if the STDSQL(YES) or NOFOR option is specified when the program is precompiled. For more on the subject, see “Positioned Updates of Columns” on page 126.

When FOR UPDATE OF 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.

For a discussion of U locks and S locks, see Section 5 (Volume 2) of *Administration Guide*.

read-only-clause

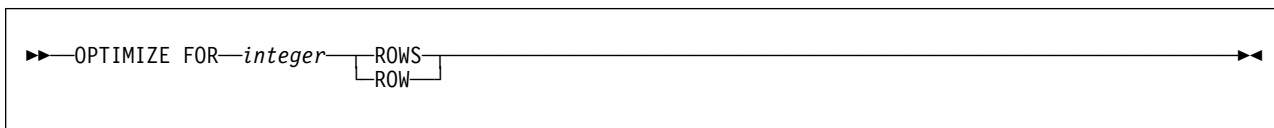


The clause `FOR FETCH ONLY`¹⁸ declares that the result table is read-only and therefore the cursor cannot be referred to in positioned `UPDATE` and `DELETE` statements.

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 possible, specifying `FOR FETCH ONLY` can possibly improve the performance of `FETCH` operations and distributed operations.

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`.

optimize-for-clause

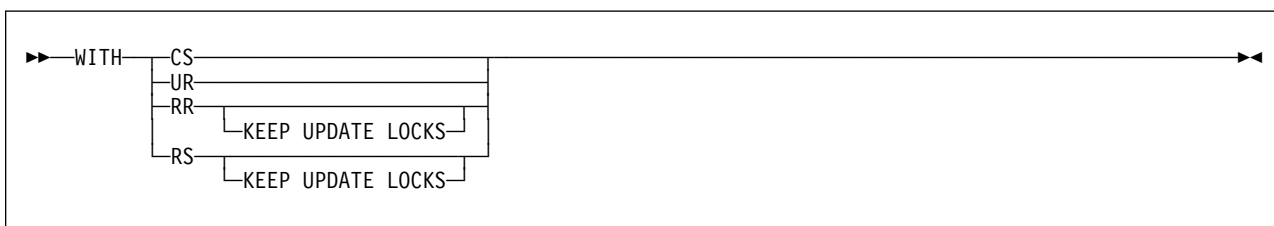


The `OPTIMIZE FOR` clause requests special optimization of the *select-statement*. If the clause is omitted, optimization is based on the assumption that all rows of the result table will be retrieved. If the clause is specified, optimization is based on the assumption that the number of rows retrieved will not exceed *n*, where *n* is the value of the integer.

The `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 *Application Programming and SQL Guide*.

|
|
|
|
|
|
#

with-clause



¹⁸ Or, `FOR READ ONLY` is equivalent.

The WITH clause specifies the isolation level at which the statement is executed.

CS	Cursor stability
UR	Uncommitted read
RR	Repeatable read
RR KEEP UPDATE LOCKS	Repeatable read update locks
RS	Read stability
RS KEEP UPDATE LOCKS	Read stability keep update locks

You can specify WITH UR only if the result table is read-only.

To specify WITH RR KEEP UPDATE LOCKS or WITH RS KEEP UPDATE LOCKS,
you must also specify the FOR UPDATE OF clause.

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
UR	Any	Read-only	UR
		Not read-only	CS
Not specified	Not specified	Any	RR
		RR	RR
		RS	RS
		CS	CS
		UR	UR
		Not read-only	CS

See “Notes” on page 349 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.

Examples of select statements

Example 1: Select all the rows from DSN8510.EMP.

```
SELECT * FROM DSN8510.EMP;
```

Example 2: Select all the rows from DSN8510.EMP, arranging the result table in chronological order by date of hiring.

```
SELECT * FROM DSN8510.EMP ORDER BY HIREDATE;
```

Example 3: Select the department number (WORKDEPT) and average departmental salary (SALARY) for all departments in the table DSN8510.EMP. Arrange the result table in ascending order by average departmental salary.

select-statement

```
SELECT WORKDEPT, AVG(SALARY)
FROM DSN8510.EMP
GROUP BY WORKDEPT
ORDER BY 2;
```

Example 4: Change various salaries, bonuses, and commissions in the table DSN8510.EMP. Confine the changes to employees in departments D11 and D21. Use positioned updates to do this with a cursor named UP_CUR. Indicate the columns to be updated in a FOR UPDATE of clause in the cursor declaration. Below is the declaration for a PL/I program.

```
EXEC SQL DECLARE UP_CUR CURSOR FOR
SELECT WORKDEPT, EMPNO, SALARY, BONUS, COMM
FROM DSN8510.EMP
WHERE WORKDEPT IN ('D11','D21')
FOR UPDATE OF SALARY, BONUS, COMM;
```

Example 5: Find the maximum, minimum, and average bonus in the table DSN8510.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.

```
EXEC SQL
SELECT MAX(BONUS), MIN(BONUS), AVG(BONUS)
INTO :MAX, :MIN, :AVG
FROM DSN8510.EMP
WITH UR;
```

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 DB2 private protocol access.

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;
```

Chapter 6. Statements

This chapter contains syntax diagrams, semantic descriptions, rules, and examples of the use of the SQL statements listed in the following table.

Table 15 (Page 1 of 3). SQL Statements

SQL Statement	Function	Page
ALLOCATE CURSOR	Defines and associates a cursor with a result set locator variable	200
ALTER DATABASE	Changes the description of a database	202
ALTER INDEX	Changes the description of an index	205
ALTER STOGROUP	Changes the description of a storage group	214
ALTER TABLE	Changes the description of a table	217
ALTER TABLESPACE	Changes the description of a table space	233
ASSOCIATE LOCATORS	Gets the result set locator value for each result set returned by a stored procedure	243
BEGIN DECLARE SECTION	Marks the beginning of a host variable declaration section	246
CALL	Calls a stored procedure	248
CLOSE	Closes a cursor	253
COMMENT ON	Replaces or adds a comment to the description of a table, view, alias, or column	255
COMMIT	Ends a unit of recovery and commits the database changes made by that unit of recovery	257
CONNECT (Type 1)	Connects the process to a server	262
CONNECT (Type 2)	Connects the process to a server	267
CREATE ALIAS	Defines an alias	270
CREATE DATABASE	Defines a database	272
CREATE GLOBAL TEMPORARY TABLE	Creates a description of a temporary table at the current server	275
CREATE INDEX	Defines an index on a table	280
CREATE STOGROUP	Defines a storage group	303
CREATE SYNONYM	Defines an alternate name for a table or view	306
CREATE TABLE	Defines a table	308
CREATE TABLESPACE	Allocates and formats a table space	327
CREATE VIEW	Defines a view of one or more tables or views	341
DECLARE CURSOR	Defines an SQL cursor	347
DECLARE STATEMENT	Declares names used to identify prepared SQL statements	352
DECLARE TABLE	Provides the programmer and the precompiler with a description of a table or view	354
DELETE	Deletes one or more rows from a table	357
DESCRIBE	Describes the result columns of a prepared statement or the columns of a table or view	362
DESCRIBE CURSOR	Puts information about the result set associated with a cursor into a descriptor	368

Statements

Table 15 (Page 2 of 3). SQL Statements

	SQL Statement	Function	Page
#	DESCRIBE INPUT	Puts information about the input parameters (markers) of a prepared statement into a descriptor	370
#	DESCRIBE PROCEDURE	Puts information about the result sets returned by a stored procedure into a descriptor	372
	DROP	Deletes an alias, database, index, package, storage group, synonym, table, table space, or view	375
	END DECLARE SECTION	Marks the end of a host variable declaration section	380
	EXECUTE	Executes a prepared SQL statement	382
	EXECUTE IMMEDIATE	Prepares and executes an SQL statement	386
	EXPLAIN	Obtains information about how an SQL statement would be executed	388
	FETCH	Assigns values of a row to host variables	397
	GRANT (Collection Privileges)	Grants authority to create a package in a collection	403
	GRANT (Database Privileges)	Grants privileges on a database	404
	GRANT (Package Privileges)	Grants authority to bind, execute, or copy a package	406
	GRANT (Plan Privileges)	Grants authority to bind or execute an application plan	408
	GRANT (System Privileges)	Grants system privileges	409
	GRANT (Table or View Privileges)	Grants privileges on a table or view	412
	GRANT (Use Privileges)	Grants authority to use specified buffer pools, storage groups, or table spaces	415
	INCLUDE	Inserts declarations into a source program	417
	INSERT	Inserts one or more rows into a table	419
	LABEL ON	Replaces or adds a label on the description of a table, view, alias, or column	424
	LOCK TABLE	Locks a table in shared or exclusive mode	426
	OPEN	Opens a cursor	428
	PREPARE	Prepares an SQL statement (with optional parameters) for execution	433
	RELEASE	Places one or more connections in the release pending state	437
	RENAME	Renames an existing table	440
	REVOKE (Collection Privileges)	Revokes authority to create a package in a collection	447
	REVOKE (Database Privileges)	Revokes privileges on a database	448
	REVOKE (Package Privileges)	Revokes authority to bind, execute, or copy a package	450
	REVOKE (Plan Privileges)	Revokes authority to bind or execute an application plan	452
	REVOKE (System Privileges)	Revokes system privileges	453
	REVOKE (Table or View Privileges)	Revokes privileges on a table or view	456
	REVOKE (Use Privileges)	Revokes authority to use specified buffer pools, storage groups, or table spaces	458

Table 15 (Page 3 of 3). SQL Statements

SQL Statement	Function	Page
ROLLBACK	Ends a unit of recovery and backs out the changes to the database made by that unit of recovery	460
SELECT INTO	Specifies a result table of no more than one row and assigns the values to host variables	462
SET CONNECTION	Establishes the application server of the process by identifying one of its existing connections	465
SET CURRENT DEGREE	Assigns a value to the CURRENT DEGREE special register	468
SET CURRENT RULES	Assigns a value to the CURRENT RULES special register	473
SET CURRENT PACKAGESET	Assigns a value to the CURRENT PACKAGESET special register	470
# SET CURRENT PRECISION	Assigns a value to the CURRENT PRECISION special register	472
SET CURRENT SQLID	Assigns a value to the CURRENT SQLID special register	474
SET <i>host-variable</i>	Assigns the current value of a named special register to a host variable	476
UPDATE	Updates the values of one or more columns in one or more rows of a table	477
WHENEVER	Defines actions to be taken on the basis of SQL return codes	483

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 CLI 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” on page 509 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 188.) 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 CLI 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 SQLCA.¹⁹ 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 “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 SQLCA.

¹⁹ SQLCODE and SQLSTATE cannot be in the SQLCA when the precompiler option STDSQL(YES) is in effect. See “SQL Standard Language” on page 124.

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 Section 6 of *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 CLI SQLPrepare function and then executed by calling the CLI 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 CLI SQLExecDirect function.

Call Level Interface 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 435 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 382 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 EXECUTE IMMEDIATE.

The successful or unsuccessful execution of the statement is indicated by setting the SQLCODE and SQLSTATE fields in SQLCA after the EXECUTE (or EXECUTE IMMEDIATE) statement. You should check the fields as described above for embedded statements.

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 SQL FETCH statement.

If the application is using CLI, 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.

Statements

The successful or unsuccessful execution of the SELECT statement is indicated by setting the SQLCODE and SQLSTATE fields in SQLCA after the OPEN. You should check the fields as described above for embedded statements.

If the application is using CLI, 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 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 433 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 428 for rules concerning this replacement.)

The successful or unsuccessful execution of the SELECT statement is indicated by the setting of the SQLCODE and SQLSTATE fields in 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 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 CLI 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 function call.

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 CLI, 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 CLI, 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

The ALLOCATE CURSOR statement defines 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

Names the cursor using the specified *cursor-name*. The name must not identify a cursor that has already been declared in the source program.

A cursor name is a long identifier.

CURSOR FOR RESULT SET *rs-locator-variable*

Names 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: When an ALLOCATE CURSOR statement is dynamically prepared, the EXECUTE statement with the USING clause must be used to execute the prepared statement. As with all dynamically prepared statements, parameter markers (question marks) must appear where the host variables would appear in the prepared statement. In the ALLOCATE CURSOR statement, *rs-locator-variable* is always a host variable. The USING clause of the EXECUTE statement should specify the host variables for which the values are to be substituted for the parameter markers in the dynamically prepared ALLOCATE CURSOR statement.

A restriction for dynamically prepared ALLOCATE CURSOR statements is that you cannot use a statement identifier for an ALLOCATE CURSOR statement if the same statement identifier has been used for 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 when it 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.

- For this statement to be successful, an application must be currently connected to the site where the stored procedure was executed.
- You cannot open an allocated cursor by using the SQL OPEN cursor statement.
- You can close an allocated cursor by using the SQL CLOSE cursor statement. This closes the cursor in the stored procedure as well.
- You can allocate only one cursor to each result set.

The life of an allocated cursor: Rollback, and an implicit and explicit close destroy allocated cursors. A commit destroys allocated cursors that are not defined WITH HOLD by the stored procedure. Destroying an allocated cursor closes the associated cursor 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

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. However, if the bind option DYNAMICRULES(BIND) applies, the statement cannot be dynamically prepared.

Authorization

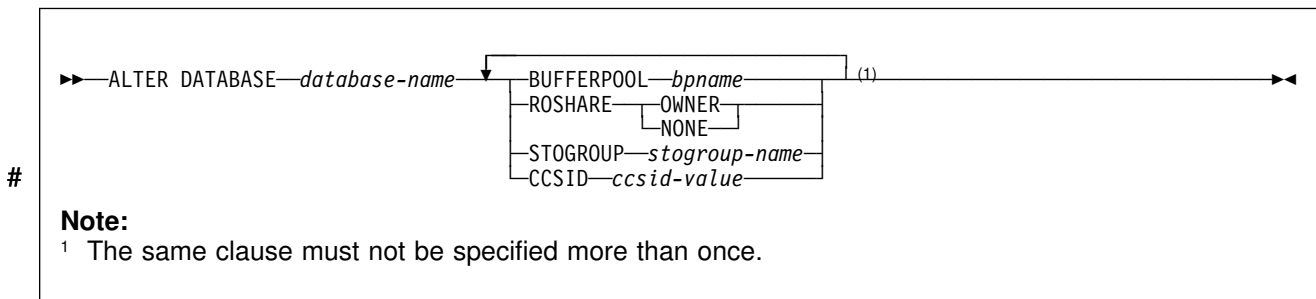
The privilege set defined below must include at least one of the following:

- The DROP privilege on the database
- Ownership of the database
- DBADM or DBCTRL authority for the database
- SYSADM or SYSCTRL authority

If ROSHARE is specified, the privilege set must include 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 held by each authorization ID of the process.

Syntax



Description

DATABASE *database-name*

Identifies the database to be altered. The name must identify database that exists at the current server. If a work file database is identified, only the BUFFERPOOL clause can be specified.

BUFFERPOOL *bpname*

Identifies the default buffer pool for the table spaces and indexes within the database. It does not apply to table spaces and indexes already existing within the database.

32KB buffer pools apply only to table spaces. If a 32KB buffer pool name is specified, the default buffer pool for indexes in the database is BP0.

See “Naming Conventions” on page 48 for more details about *bpname*.

ROSHARE

Indicates whether or not the database is to be shared with other DB2 subsystems using *shared read-only data*. Cannot be used if the database was defined with ROSHARE READ. For an explanation of shared read-only data, see Appendix F (Volume 2) of *Administration Guide*. Also, ROSHARE cannot be specified for any system database.

OWNER The database will be shared, and the current server will be the DB2 that can update the database.

NONE The database will not be shared.

STOGROUP *stogroup-name*

Is the name of 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 already existing within the database. STOGROUP cannot be specified for a work file database.

CCSID *ccsid-value*

Identifies the default CCSID for tablespaces within the database. It does not apply to existing tablespaces 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.

#

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 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 or EBCDIC), DB2 supports only one CCSID. Therefore, the CCSID 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 B of *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.

ALTER DATABASE

EBCDIC CCSIDs

```
-----  
37          1140  
273         1141  
277         1142  
278         1143  
280         1144  
284         1145  
285         1146  
297         1147  
500         1148  
871         1149
```

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 database ABCDE to BP2. Also, change the ROSHARE option for this database to NONE.

```
ALTER DATABASE ABCDE  
  BUFFERPOOL BP2  
  ROSHARE NONE;
```

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. However, if the bind option DYNAMICRULES(BIND) applies, the statement cannot be dynamically prepared.

Authorization

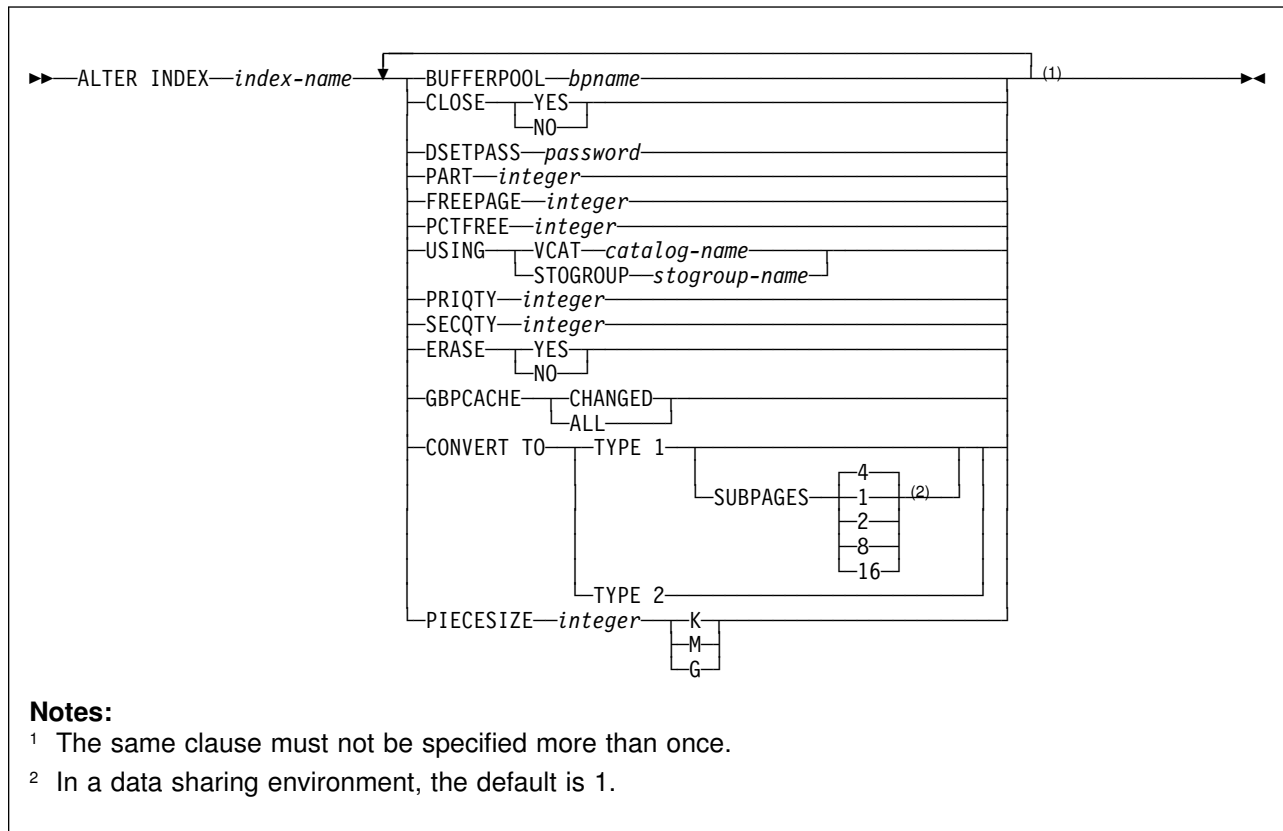
The privilege set defined below must include one of the following:

- Ownership of the index
- Ownership of the table on which the index is defined
- DBADM authority for the database containing 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.

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 held by each authorization ID of the process.

Syntax



Description

index-name

Identifies the index to be altered. The name must identify a user-created index that exists at the current server.

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 48 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.

DSETPASS *password*

Specifies a master level password sent to access method services when the data sets of the index are used by DB2. *password* is a short identifier. If delimited, *password* can contain any characters acceptable to access method services. The change to the description of the index takes effect the next time the data sets of the index space are opened.

To remove the password, use a delimited string of blanks for *password*. For example, you can use the following if the double quote is your escape character:

```
DSETPASS " "
```

If the index uses a storage group, *password* is the password that protects the data sets as well as the password that is passed to access method services when the data sets are used by DB2. If the index does not use a storage group, the password that protects the data sets must be defined using access method services.

Changing the password for the index does not change the password that protects its data sets. To change the data set password, use access method services. See Section 2 (Volume 1) of *Administration Guide* for more on changing data set passwords.

The password does not apply to the data sets managed by Storage Management Subsystem (SMS). To protect data sets defined to SMS, use RACF or a similar external security system.

PART *integer*

Identifies a partition of the index. Thus, 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 not partitioned, or if you use the BUFFERPOOL, CLOSE, DSETPASS, or CONVERT TO clause. You must use this clause if the index is partitioned and you use the FREEPAGE, PCTFREE, USING, PRIQTY, SECQTY, ERASE, or GBPCACHE clause. In this case, the alterations specified by these clauses apply only to the identified partition of the index.

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 subpage when entries are added to the index or partition as the result of executing a DB2 utility. The first entry in a page or subpage is loaded without restriction. When additional entries are placed in a nonleaf page, the percentage of free space is at least as great as *integer*. When additional entries

are placed in a leaf page, the percentage of free space is at least as great as $integer/m$, where m is the number of subpages.

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.

USING

Specifies whether a data set for the index or partition is managed by the user or managed by DB2. If the index is partitioned, USING applies to the data set for the partition identified in the PART clause. If the index is nonpartitioned, USING applies to every data set that can be used for the index. (A nonpartitioned index can have more than one data set if PRIQTY+118 × SECQTY is at least 2 gigabytes.)

If you specify USING, the index or partition must be in the stopped state when the ALTER INDEX statement is executed. See “Altering storage attributes” on page 212 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 conforming to the DB2 naming conventions set forth in Section 2 (Volume 1) of *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. However, the same *catalog-name* must be used by the subsystems when shared read-only data is used.

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 and, if the integrated catalog facility catalog is password protected, the description of the storage group must include a valid password.

If you specify USING STOGROUP and the current data set for the index or partition is managed by DB2:

- 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 and the current data set for the index or partition is managed by the user:

- Omission of the PRIQTY clause is an implicit specification of PRIQTY 12
- Omission of the SECQTY and the PRIQTY clauses is an implicit specification of SECQTY 12
- 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 index 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, or
- A USING clause is not specified.

If USING STOGROUP is specified, PRIQTY has the default specified in the description of USING STOGROUP. If PRIQTY is specified, the primary space allocation is at least *n* kilobytes, where *n* is:

12	If <i>integer</i> is less than 12
<i>integer</i>	If <i>integer</i> is between 12 and 4194304
4194304	If <i>integer</i> is greater than 4194304

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 of the index 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, or
- A USING clause is not specified.

If USING STOGROUP is specified, SECQTY has the default specified in the description of USING STOGROUP.

However, if ALTER INDEX is being used to convert from user-defined data sets to storage groups and PRIQTY is specified, the default for SECQTY is either 10% of PRIQTY or 3 times the index page size (4K), whichever is larger (if this

value exceeds 131068, the default is 131068.) If SECQTY is specified, the secondary space allocation is at least *n* kilobytes, where *n* is:

<i>integer</i>	If <i>integer</i> is not greater than 131068
131068	If <i>integer</i> is greater than 131068

If *integer* is 0, no data set for the index can be extended.

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 for the index or partition 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 managed by DB2, and if one of the following is true:

- USING STOGROUP is specified, or
- A USING clause is not specified.

If you specify ERASE, the index or partition must be in the stopped state when the ALTER INDEX statement is executed. See “Altering storage attributes” on page 212 to determine how and when changes take effect.

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.

ALL

Indicates that pages are to be cached in 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.

CONVERT TO

Specifies changing the type of index. The index is left in recover pending state, and the index change does not take place until the index is rebuilt by a LOAD REPLACE or REORG of the entire table space, or a RECOVER or RELOAD of the whole index.

TYPE 1

Specifies that the index is type 1. CONVERT TO TYPE 1 is not allowed if:

- The table space associated with the index has a LOCKSIZE value of ROW.
- The index was defined with UNIQUE WHERE NOT NULL.

SUBPAGES *n*

Gives the number of subpages for each physical page. Use 1,2,4,8, or 16. The default is 4, except in a data sharing environment when it is 1. In a data sharing environment, you must specify 1 for type 1 indexes to be shared; when there is more than one subpage, an index cannot be accessed when there is inter-DB2 R/W interest in the index.

The number of subpages for some type 1 catalog indexes is 1, regardless of what is implicitly or explicitly specified. For a list of these catalog indexes, see “SQL Statements Allowed on the Catalog” on page 532. In addition, in a data sharing environment, type 1 catalog indexes cannot have more than one subpage; therefore, the only value you can specify for SUBPAGES is 1.

TYPE 2

Specifies that the index is type 2.

PIECESIZE *integer*

Specifies the maximum addressability of each piece (data set) for a nonpartitioned 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 4 194 304.

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 4 096.

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 4.

In the above specification for piece size, spaces are permitted between the integer and K, M, or G. They are not required.

Valid values for piece size are as follows:

ALTER INDEX

```
|
|
|
# 256 K
| 512 K
| 1024 K (or 1 M)
| 2048 K (or 2 M)
| 4096 K (or 4 M)
| 8192 K (or 8 M)
| 16384 K (or 16 M)
| 32768 K (or 32 M)
| 65536 K (or 64 M)
| 131072 K (or 128 M)
| 262144 K (or 256 M)
| 524288 K (or 512 M)
| 1048576 K (or 1024 M or 1 G)
| 2097152 K (or 2048 M or 2 G)
| 4194304 K (or 4096 M or 4 G)20
```

When you alter the piece size value, the index is placed into page set recovery pending (PSRCP). You must run the RECOVER INDEX or the REORG TABLESPACE utility to remove that status.

Notes

The ALTER INDEX statement cannot be executed while a DB2 utility has control of the index or its associated table space.

To change FREEPAGE, PCTFREE, USING, PRIQTY, SECQTY, ERASE, or GBPCACHE for more than one partition, you must use separate ALTER INDEX statements.

```
# Altering the type of index: When you change the type of index, the ALTER
# INDEX statement cannot be executed during the same commit scope as other
# changes to the index. Do not execute an ALTER INDEX statement with the
# CONVERT TO clause until other changes to the index are committed or rolled back.
```

When you change a type 2 index to a type 1 index, the plans and packages associated with that index are invalidated. You must rebind those plans and packages.

```
# Altering storage attributes: The USING, PRIQTY, SECQTY, and ERASE clauses
# define the storage attributes of the index or partition. 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 Section 2 (Volume 1) of *Administration Guide*.

²⁰ Only valid for LARGE table spaces.

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 indexes on DB2 catalog tables: For details on altering options on catalog tables, see “SQL Statements Allowed on the Catalog” on page 532.

Examples

Example 1: Alter the index DSN8510.XEMP1. CLOSE NO indicates that DB2 is not to close the data sets supporting the index when there are no current users of the index.

```
ALTER INDEX DSN8510.XEMP1
  CLOSE NO;
```

Example 2: Alter the index DSN8510.XPROJ1. BP1 is the buffer pool to be associated with the index. OSESAME is the password that is passed to VSAM when the data sets are used by DB2.

```
ALTER INDEX DSN8510.XPROJ1
  BUFFERPOOL BP1
  DSETPASS OSESAME;
```

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. However, if the bind option DYNAMICRULES(BIND) applies, the statement cannot be dynamically prepared.

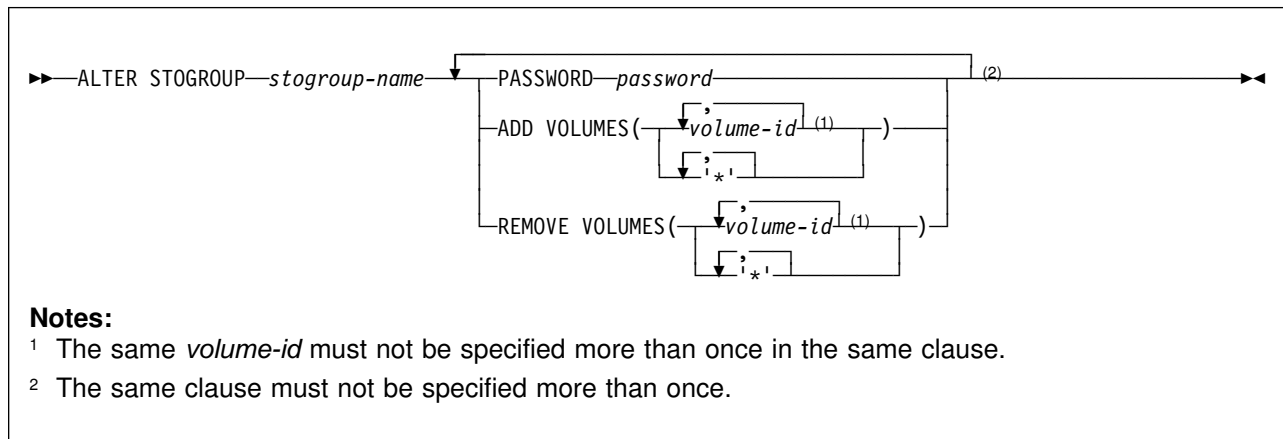
Authorization

The privilege set defined below must include one of the following:

- Ownership of the storage group
- SYSADM or SYSCTRL authority

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 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.

PASSWORD *password*

Gives a VSAM control or master level password in the form of a short identifier. If the password is a delimited identifier, it can contain any special characters acceptable to access method services. The password is used to access the integrated catalog facility catalog. The password that protects the catalog must be established by the installation of access method services.

To remove the password, use a delimited string of blanks for *password*. For example, you can use the following if the double quote is your escape character:

```
PASSWORD " "
```

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.

If the storage group is defined with one or more asterisks (*) listed after VOLUMES, listing one or more asterisks after ADD VOLUMES extends the first list. SMS uses as many volumes as there are asterisks in the concatenation of the two lists to manage extension of data sets for shared read-only data.

See "SMS dataset management" on page 216 for a description of ADD VOLUMES('*').

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. To remove volumes from a storage group that is defined with a list of asterisks, specify one asterisk for each volume you want to remove.

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.

Notes

Work file databases: If the storage group altered contains data sets in database DSNDB07 or in any other 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.

ALTER STOGROUP

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*.

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 Section 2 (Volume 1) of *Administration Guide* for more information about creating DB2 storage groups.

SMS dataset management: You can allow Storage Management Subsystem (SMS) to 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 Section 2 (Volume 1) of *Administration Guide* for considerations for using SMS to manage data sets.

Examples

Example 1: Alter storage group DSN8G510. OSESAME is the password that is used to access the integrated catalog facility catalog. DSNV04 and DSNV05 are the volumes to be added.

```
ALTER STOGROUP DSN8G510
  PASSWORD OSESAME
  ADD VOLUMES (DSNV04,DSNV05);
```

Example 2: Alter storage group DSN8G510. DSNV04 and DSNV05 are the volumes to be removed.

```
ALTER STOGROUP DSN8G510
  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. However, if the bind option DYNAMICRULES(BIND) applies, the statement cannot be dynamically prepared.

Authorization

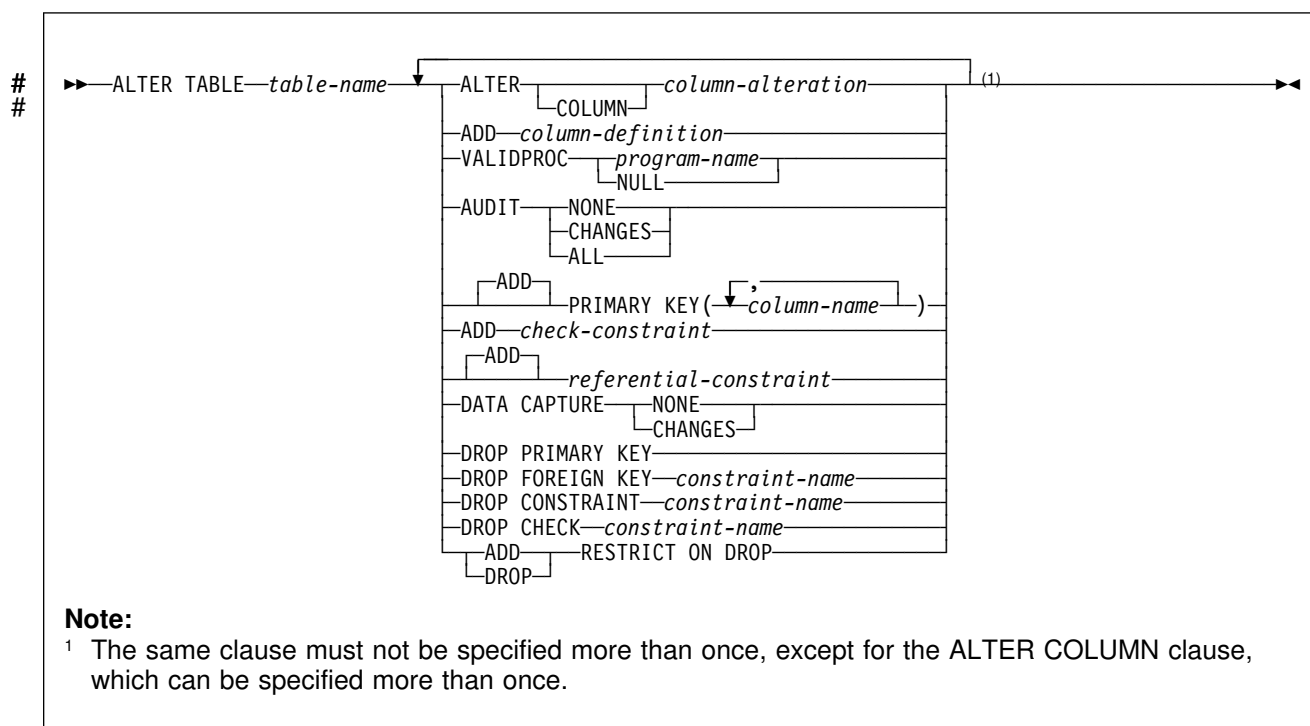
The privilege set defined below must include at least one of the following:

- The ALTER privilege on the table
- Ownership of the table
- DBADM authority for the database
- SYSADM or SYSCTRL authority

If FOREIGN KEY, DROP PRIMARY KEY, DROP FOREIGN KEY, or DROP CONSTRAINT is specified, an additional privilege could be required. More detail about this can be found in the description of the appropriate clauses.

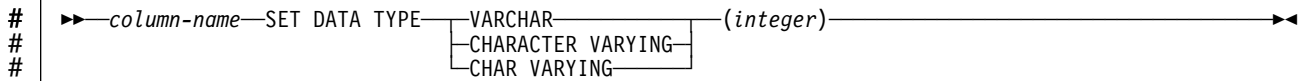
If the statement is embedded in an application program, the privilege set is the privileges held by the authorization ID of the owner of the plan or package. If the statement is dynamically prepared, the privilege set is the union of the privilege sets held by each authorization ID of the process.

Syntax

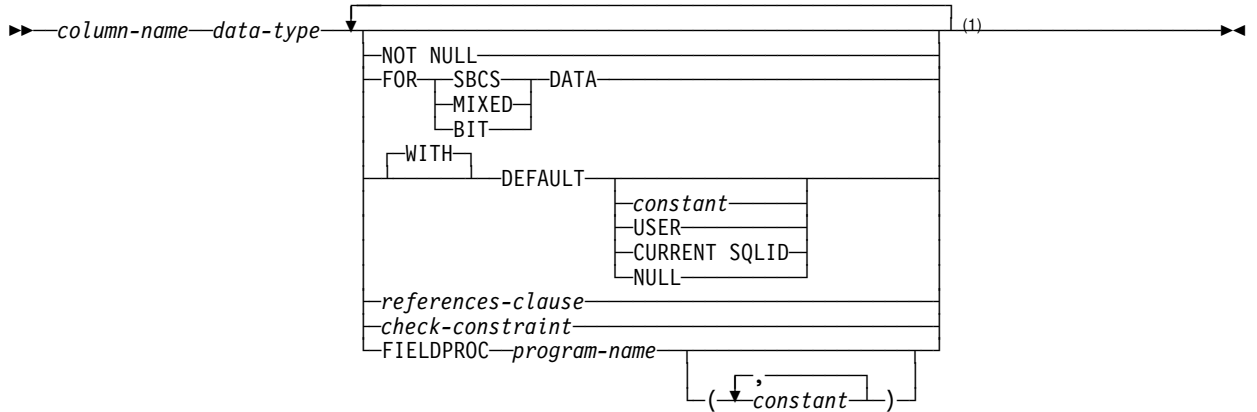


ALTER TABLE

column-alteration:



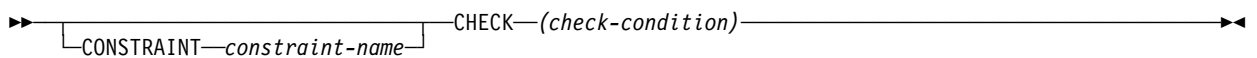
column-definition:



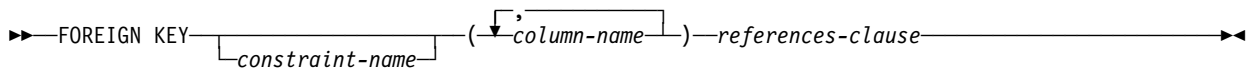
Note:

¹ The same clause must not be specified more than once.

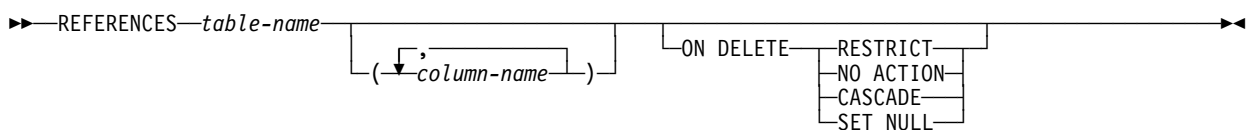
check-constraint:



referential-constraint:



references-clause:



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 a 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. See “Notes” on page 228 for a list of the restrictions of when a column cannot be altered and other information about altering columns.

#*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 324. 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.

#

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. All values of the column in existing rows are 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 324.

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*.

data-type

Specifies the data type of the column. See “data-type” on page 311 for the rules.

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.

FOR subtype DATA

Specifies the subtype of a character string column; that is, for a column with a data type of CHAR, VARCHAR, or LONG VARCHAR. The FOR DATA clause must not be used with columns of any other data type.

The next page shows what *subtype* can be.

subtype can be one of the following:

SBCS

Column holds single-byte data.

MIXED

Column holds mixed data.

BIT

Column holds BIT data.

MIXED cannot be specified when the value of field MIXED DATA on installation panel DSNTIPF is NO.

A default subtype applies if the FOR clause is not used in defining a new character string column. 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.

DEFAULT

The default value assigned to the column in the absence of a value specified on INSERT or LOAD. 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	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 value other than the one above can be specified in one of the following forms:

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 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 (SQLID) 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 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

The null value.

In a given column definition:

- NOT NULL and DEFAULT NULL cannot both be specified.
- Omission of NOT NULL and DEFAULT is an implicit specification of DEFAULT NULL.
- DEFAULT and FIELDPROC cannot both be specified.

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.

check-constraint

The *check-constraint* of a *column-definition* has the same affect 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.

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 *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.

ALTER TABLE

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 254 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 _____

VALIDPROC

Names a validation procedure for the table or inhibits the execution of any existing validation procedure.

program-name

Is the name of the new validation exit routine for the table. Validation exit routines are described in Appendix B (Volume 2) of *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. 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. The ALTER TABLE statement used to alter the table is audited only if the auditing attribute of the table is changed and the appropriate audit trace class is active. For information about audit trace classes, see Section 3 (Volume 1) of *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.

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 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 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.

check-constraint

ADD check-constraint

Designates the values that specific columns of the table can contain.

CONSTRAINT *constraint-name*

Names the table check constraint. The constraint name must be different from the names of any existing referential or check constraints 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* is a search condition, with the following restrictions:

- It can refer only to columns of table *table-name*.
- It can be up to 3800 bytes long, not including redundant blanks.
- It must not contain any of the following:
 - Subselects
 - Functions
 - Host variables
 - Parameter markers
 - Special registers
 - Columns that include a field procedure
 - CASE Expressions
 - Quantified predicates

ALTER TABLE

- 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
- 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.

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 table space containing the table is placed in a check pending status. For a description of the check pending status and the implications for utility operations, see Section 2 (Volume 1) of *Administration Guide* .

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

referential-constraint

FOREIGN KEY *constraint-name (column-name,...)* **references-clause**

Specifies a referential constraint with the specified *constraint-name*. A name is generated if a *constraint-name* is not specified. The generated name is derived from the name of the first column of the foreign key in the same way that the name of an implicitly created table space is derived from the name of a table except that the scope of uniqueness of a *constraint-name* is the table. If specified, *constraint-name* must be different from the names of any existing referential or check constraints on the table.

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 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 same column must not be identified more than once.

The list of column names must be identical to the list of column names in a unique index (UNIQUERULE in SYSINDEXES will be R, P, C, or U). The column names must be specified in the *same order* as in the unique index on T2.

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 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.²¹ A table in a segmented table space is populated if the table is not empty. A table in a nonsegmented table space is considered populated if the table space has ever contained any records.

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 Integrity” on page 24.

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 partitioned 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.

²¹ 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.

- T2 is a descendent of T3 and the deletion of rows from T3 cascades to T2.
- T2 and T3 are both descendents 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

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 *Administration Guide*

NONE

Do not record additional information to the log.

CHANGES

Write additional data about SQL updates to the log.

For details about the recording of additional data for logged updates to
catalog tables, see “Notes” on page 228.

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.

If the table has a primary index, its description 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 CONSTRAINT *constraint-name*

Drops the constraint *constraint-name*. The constraint-name must identify an existing check constraint or referential constraint defined on the table. If the constraint-name identifies a referential constraint in which the table is the dependent table, then the privilege set must include the ALTER or REFERENCES privilege on the parent table of that relationship.

ALTER TABLE

DROP CONSTRAINT must not be used on the same ALTER TABLE statement as DROP FOREIGN KEY or DROP CHECK.

DROP CHECK *constraint-name*

Drops the check constraint *constraint-name*. The constraint-name must identify an existing check constraint defined on the table.

ADD RESTRICT ON DROP

Restricts dropping the table and the database and table space that contain the table.

DROP RESTRICT ON DROP

Removes the restriction on dropping the table and the database and table space that contain the table.

Notes

Altering the length of a VARCHAR column: If you change the length of a
VARCHAR column, be aware of the following information about restrictions,
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, view, or stored
procedure.
– 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 global 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 partitioned index is created, the
highest value of the index key for each partition (also called the limit key) is
defined. DB2 generates the limit key for each partition from the constants that
are explicitly or implicitly specified for each column in the index such that the
limit key is a maximum of 40 bytes in length. For details on how the limit key is
created, see the description of the VALUES clause for “CREATE INDEX” on
page 280.

When a table is altered and the length of a column in the index is changed,
DB2 might change the length of a limit key for a partition. If the new column
length changes the value of the limit key for a partition such that data would

have to be moved from one partition to another, DB2 changes the length of the
 # limit key to prevent data relocation. When DB2 needs to change the limit key
 # length, it is increased by the same amount that the column is increased; the
 # limit key can be no longer than 255 bytes.

As the following examples help illustrate, the limit key length changes if the
 # column being altered is not the last column in the partitioned index, and the
 # sum of the lengths of the preceding columns in the index and the existing
 # length of the columns being altered is less than 40 bytes. The length of the limit
 # key is increased by the same amount that the length of column is increased.
 # When the length of a limit key changes, the table becomes release dependent.

Example of when the length of the limit key changes: Assume that the table
 # space, table, and partitioned index are created with these statements:

```
# CREATE TABLESPACE TSP1 Numparts 4;
#
# CREATE TABLE TB1
#   (COL1          CHAR(10)          NOT NULL,
#    VCOL2         VARCHAR(20)       NOT NULL,
#    COL3          CHAR(15)          NOT NULL,
#    COL4          CHAR(20)          NOT NULL)
# IN TSP1;
#
# CREATE INDEX IX1 ON TB1 (COL1, VCOL2, COL3) CLUSTER
#   (PART 1 VALUES ('DDDDDDDDDD', 'EEEEEEEEEEEEEEEEEEEE', 'YYYYYYYYYYYYYYY'),
#    PART 2 VALUES ('HHHH', 'HHHH', 'HHHH'),
#    PART 3 VALUES ('LLLL', 'LLLL', 'LLLL'),
#    PART 4 VALUES ('PPPP', 'PPPP', 'PPPP'));
```

The limit key for Partition 1 is:

```
# 'DDDDDDDDDD' || 'EEEEEEEEEEEEEEEEEEEE' || 'YYYYYYYYYYY'
```

Assume that there is a row (RX) in the table with the following values

```
# COL1 = 'DDDDDDDDDD'
# VCOL2 = 'EEEEEEEEEEEEEEEEEEEE'
# COL3 = 'ZZZZZZZZZZZZZZ'
```

Row RX is in Partition 2 because the key value for the row is greater than the
 # limit key for Partition 1. However, if the length of column VCOL2 is changed
 # from 20 bytes to 30 bytes and the length of the limit key for Partition 1 remains
 # 40 bytes, the new limit key for Partition 1 is:

```
# 'DDDDDDDDDD' || 'EEEEEEEEEEEEEEEEEEEE' || 'FFFFFFFFFFFFFFFFFFFF'X
```

This value for the limit key of Partition 1 would require that row RX be moved
 # from Partition 2 to Partition 1. Thus, to avoid moving data, DB2 increases the
 # length of the limit key to 50 bytes (40 bytes plus the length that column VCOL2
 # was increased, which was 10 bytes), which makes the value of the limit key:

```
# 'DDDDDDDDDD' || 'EEEEEEEEEEEEEEEEEEEE' || 'FFFFFFFFFFFFFFFFFFFF'X
#           || 'YYYYYYYYYYY'
```

The increase in length and change in value of the limit key allows row RX to
 # remain in Partition 2.

Example of when the length of the limit key does not change: Assume that
 # the table space, table, and partitioned index are created with these statements:

ALTER TABLE

```
# CREATE TABLESPACE TSP1 NUMPARTS 4;
#
# CREATE TABLE TB1
# (COL1 CHAR(10) NOT NULL,
# COL2 CHAR(40) NOT NULL,
# VCOL2 VARCHAR(20) NOT NULL,
# COL3 CHAR(15) NOT NULL,
# COL4 CHAR(20) NOT NULL)
# IN TSP1;
#
# CREATE INDEX IX1 ON TB1 (COL1, COL2, VCOL2, COL3) CLUSTER
# (PART 1 VALUES ('DDDDDDDDDD', 'EEEEEEEEEEEEEEEEEEEE', 'YYYYYYYYYYYYYYYY'),
# PART 2 VALUES ('HHHH', 'HHHH', 'HHHH'),
# PART 3 VALUES ('LLLL', 'LLLL', 'LLLL'),
# PART 4 VALUES ('PPPP', 'PPPP', 'PPPP'));
```

The limit key for Partition 1 is:

```
# 'DDDDDDDDDD' || 'EEEEEEEEEEEEEEEEEEEE' || 'FFFFFFFFFFFFFFFFFFFF'X
```

Because the limit key for each partition is 40 bytes in length when the index is created, notice that VCOL2 does not contribute to the limit key. Thus, increasing the length of VCOL2 has no effect on the limit key; DB2 would never have to change the length of the limit key to prevent data relocation when the length of VCOL2 is increased.

- *Check constraints.* If a table check constraint refers to the 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.

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.

Invalidation of plans and packages: When a table is altered, all plans and packages that refer to the table are invalidated if one or more of the following 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 table is a 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.

Order of processing of clauses: With the exception that the ADD *check-constraint* clause is processed last, the other clauses of an ALTER TABLE statement are processed in the order in which they are specified.

Views: Adding a column to a table has no effect on existing views.

Restrictions: When using ALTER TABLE, you cannot:

- Use NOT NULL without specifying a nonnull default value for the column;
- Add a column if an edit procedure exists for the table.
- Specify DROP CONSTRAINT on the same statement as DROP FOREIGN KEY or DROP CHECK.
- Change a temporary table except to add a column. The added column must be defined with a default value of NULL.

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.

Running utilities: You cannot execute ALTER TABLE while a utility has control of the table space that contains the table.

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.

Activity to the catalog that is caused by DB2 utilities is not captured. For example,
log records from executing a utility on a catalog table, to record the event of
executing a utility, or for catalog changes that result from executing the RUNSTATS
utility on a user table will not have data capture information.

Examples

Example 1: Column DEPTNAME in table DSN8510.DEPT was created as a
VARCHAR(36). Increase its length to 50 bytes. Also, add a column named BLDG
to the table. Describe the new column as a character string column that holds
SBCS data.

```
ALTER TABLE DSN8510.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 DSN8510.EMP.

```
ALTER TABLE DSN8510.EMP
  VALIDPROC DSN8EAEM;
```

Example 3: Disassociate the current validation procedure from the table DSN8510.EMP. After the statement is executed, the table no longer has a validation procedure.

```
ALTER TABLE DSN8510.EMP
  VALIDPROC NULL;
```

Example 4: Define ADMRDEPT as the foreign key of a self-referencing constraint on DSN8510.DEPT.

```
ALTER TABLE DSN8510.DEPT
  FOREIGN KEY(ADMRDEPT) REFERENCES DSN8510.DEPT ON DELETE CASCADE;
```

ALTER TABLE

Example 5: Add a check constraint to the table DSN8510.EMP which checks that the minimum salary an employee can have is \$10,00.

```
ALTER TABLE DSN8510.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;
```

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. However, if the bind option DYNAMICRULES(BIND) applies, the statement cannot be dynamically prepared.

Authorization

The privilege set defined below must include at least one of the following:

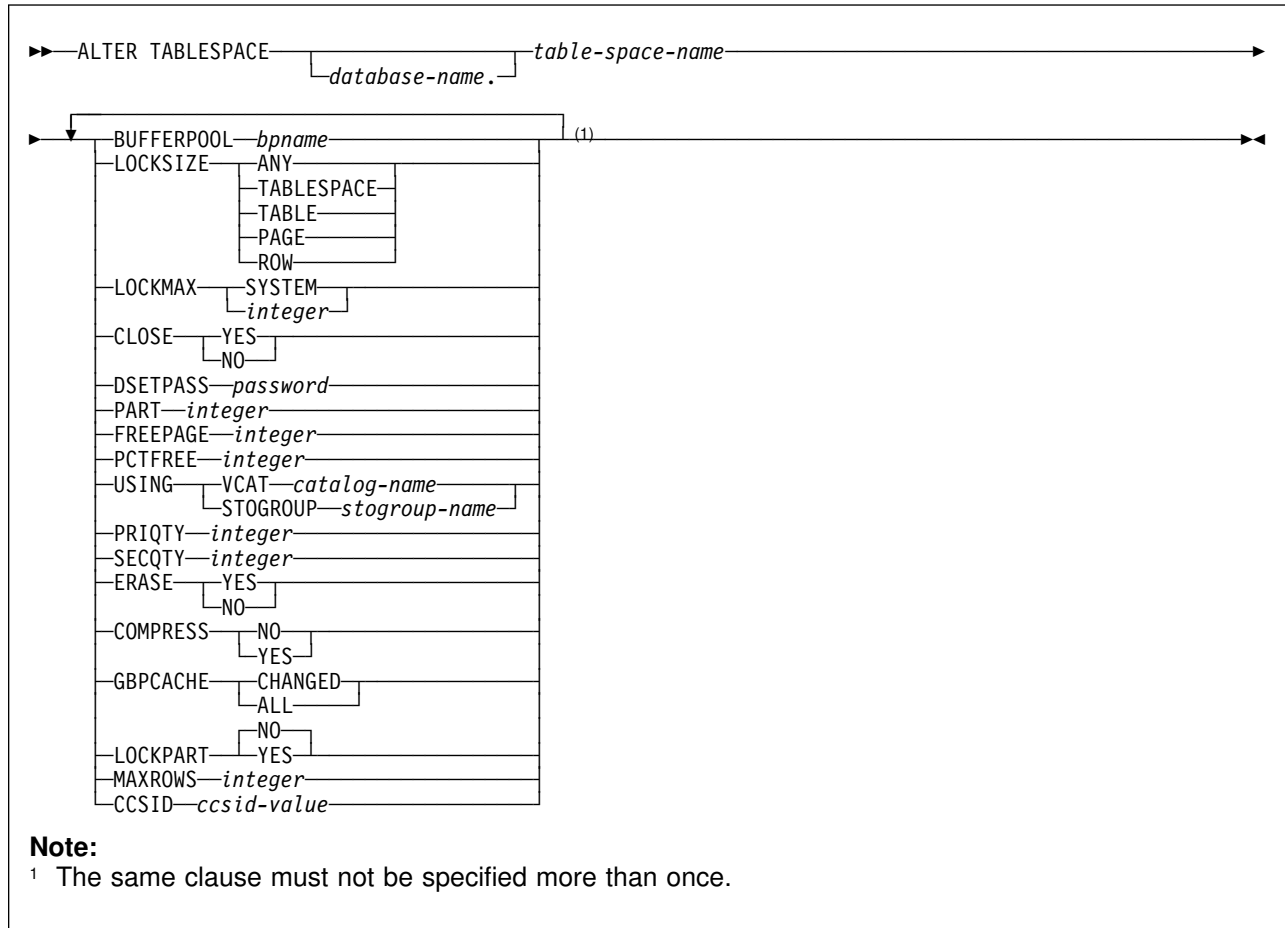
- Ownership of the table space
- DBADM authority for its database
- SYSADM or SYSCTRL authority.

If BUFFERPOOL or USING STOGROUP is specified, additional privileges could be needed, as explained in the description of those clauses.

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 held by each authorization ID of the process.

ALTER TABLESPACE

Syntax



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 DSNDB07 or of any 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 48 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. You must not use this clause for a table space in DSNDB07 or in any work file 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. However, when the number of page locks acquired for the table space exceeds the maximum number of locks allowed for a table space (an installation parameter), the page 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. TABLE can be specified only for a segmented table space.

PAGE

Specifies page locks.

ROW

Specifies row locks.

If you specify ROW, all indexes defined on tables in the table space must be type 2 indexes. If you specify LOCKSIZE ROW for a table space, you cannot create a type 1 index on any of its tables. If you attempt to alter a table space to LOCKSIZE ROW, the statement fails if a type 1 index exists on any of its tables.

Let S denote an SQL statement that refers to a table in the table space:

- The LOCKSIZE change affects S if S is prepared and executed after the change. This includes dynamic statements and static statements that are not bound because of VALIDATE(RUN).
- If the size specified by the new LOCKSIZE is greater than the size of the old LOCKSIZE, the change affects S if S is a static statement that is executed after the change.

The hierarchy of lock sizes, starting with the largest, is as follows:

- table space lock
- table lock (only for segmented table spaces)
- page lock and row lock (which are at the same level)

- In all other cases, LOCKSIZE has no effect on S until S is rebound.

LOCKMAX

Specifies the maximum number of page or row locks an application process can hold simultaneously in the table space. If a program requests more than

ALTER TABLESPACE

that number, locks are escalated. The page or row locks are released and the intent lock on the table space or segmented table is promoted to S or X mode.

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 or row 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
ROW or PAGE	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.

NO

Eligible for closing after all eligible CLOSE YES data sets are closed.

DSETPASS *password*

Specifies a master level password sent to access method services when the data sets of the table space are used by DB2. *password* is a short identifier. If delimited, *password* can contain any characters acceptable to access method services. The change to the description of the table space takes effect the next time the data sets of the table space are opened.

To remove the password, use a delimited string of blanks for *password*. For example, you can use the following if the double quote is your escape character:

```
DSETPASS " "
```

Changing the password for the table space does not change the password that protects its data sets. To change the data set password, use access method services. See Section 2 (Volume 1) of *Administration Guide* for more on changing data set passwords.

The password does not apply to data sets managed by Storage Management Subsystem. To protect data sets defined to SMS, use RACF or a similar external security system.

PART *integer*

Identifies a partition of the table space. Thus, 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 if the table space is not partitioned. You must use this clause if the table space is partitioned and you use the FREEPAGE, PCTFREE, USING, PRIQTY, SECQTY, COMPRESS, ERASE, or GBPCACHE clause. In this case, the alterations 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 use this keyword with database DSNDB07 or with any work file 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 use this keyword with database DSNDB07 or with any work file 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 $PRIQTY+118 \times SECQTY$ is at least 2 gigabytes.)

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 241 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 Section 2 (Volume 1) of *Administration Guide*.

ALTER TABLESPACE

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. However, the same *catalog-name* must be used by the subsystems when shared read-only data is used.

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 and, if the integrated catalog facility catalog is password protected, the description of the storage group must include a valid password.

If you specify USING STOGROUP and the current data set for the table space or partition is managed by DB2:

- 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 and the current data set for the table space or partition is managed by the user:

- Omission of the PRIQTY clause is an implicit specification of PRIQTY 12 for a table space with 4KB pages and PRIQTY 96 for a table space with 32KB pages.
- Omission of the SECQTY and PRIQTY clauses is an implicit specification of SECQTY 12 for a table space with 4KB pages and SECQTY 96 for a table space with 32KB pages.
- 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, or
- A USING clause is not specified.

If USING STOGROUP is specified, PRIQTY has the default specified in the description of USING STOGROUP. If PRIQTY is specified, the primary space allocation is at least *n* kilobytes, where *n* is the value of *integer*, except in these cases:

- If the page size is 4KB, and if *integer* is less than 12 or PRIQTY is omitted, then *n* is 12.
- If the page size is 32KB, and if *integer* is less than 96 or PRIQTY is omitted, then *n* is 96.
- If *integer* is greater than 4194304, then *n* is 4194304.

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.

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, or
- A USING clause is not specified.

If USING STOGROUP is specified, SECQTY has the default specified in the description of USING STOGROUP. However, if ALTER TABLESPACE is being used to convert from user-defined data sets to storage groups and PRIQTY is specified, the default for SECQTY is either 10% of PRIQTY or 3 times the page size of the table space, whichever is larger (if this value exceeds 131068, the default is 131068.) If SECQTY is specified, the secondary space allocation is at least *n* kilobytes, where *n* is the value of *integer* except in these cases:

- If the page size is 4KB and *integer* is greater than 131068, then *n* is 131068.
- If the page size is 32KB and *integer* is greater than 131040, then *n* is 131040.

If *integer* is 0, no data set can be extended.

DB2 specifies the secondary 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*.

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.

#

ALTER TABLESPACE

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 241 to determine how and when changes take effect.

COMPRESS

Specifies whether data compression applies to the rows of the table space or partition.

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 Section 2 (Volume 1) of *Administration Guide* for more information about the dictionary and data compression.

GBPCACHE

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 this option, but it is ignored. Do not use this clause for a table space in a work file database.

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.

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.

In a data sharing environment, hiperpools are not used for table spaces or partitions that are defined with GBPCACHE ALL.

LOCKPART

Indicates whether selective partition locking (SPL) is to be used when locking a partitioned table space. If LOCKPART YES is specified and all conditions required for SPL are met, only the partitions accessed will be locked. If LOCKPART YES is specified and all conditions required for SPL are not met, every partition of the table space is locked.

If you specify LOCKPART NO, 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.

LOCKPART is only valid 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.

You cannot alter the MAXROWS value of a table space in a work file database or for the following table spaces:

- DSNDB06.SYSDBASE
- DSNDB06.SYSDBAUT
- DSNDB06.SYSGROUP
- DSNDB06.SYSPLAN
- DSNDB06.SYSVIEWS

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. The “Notes” on page 203 for ALTER DATABASE contains a list that shows the CCSID to which a given CCSID can be changed and details about changing it.

Notes

The ALTER TABLESPACE statement cannot be executed while a DB2 utility has control of the table space.

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. If you specify the USING or ERASE clause 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.

ALTER TABLESPACE

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 Section 2 (Volume 1) of *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. Changes to the other storage attributes take effect the next
time you use the REORG, RECOVER, or LOAD REPLACE utility on the table
space or partition. 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.

Altering table spaces for DB2 catalog tables: For details on altering options on catalog tables, see “SQL Statements Allowed on the Catalog” on page 532.

Examples

Example 1: Alter table space DSN8S51E in database DSN8D51A. 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. OSESAME is the password that is passed to VSAM when the data sets are used by DB2.

```
ALTER TABLESPACE DSN8D51A.DSN8S51E
  CLOSE NO
  DSETPASS OSESAME;
```

Example 2: Alter table space DSN8S51D in database DSN8D51A. BP2 is the buffer pool associated with the table space. PAGE is the level at which locking is to take place.

```
ALTER TABLESPACE DSN8D51A.DSN8S51D
  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.

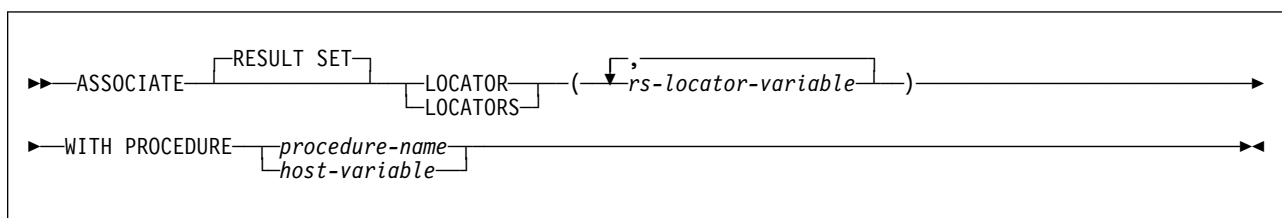
Invocation

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 is returned by the stored procedure. 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*

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 is a long SQL identifier that must be composed of SBCS characters:

- 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 next two parts identify the stored procedure. A period must separate each of the parts.
- A two-part procedure name is implicitly qualified by the location name of the current server. The name with its implicit qualifier identifies a stored procedure. A period must separate the two parts. The first part identifies the stored procedure at the server. The location name of the first part depends on the application server.
- 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 identifies the stored procedure at the

ASSOCIATE LOCATORS

server. The meaning of the second implicit qualifier depends on the application server (for MVS, it is SYSPROC). The name and its implicit qualifiers identifies a stored procedure.

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 be preceded by a colon and must not be followed by an indicator variable.
- The value of the host variable is a specification that depends on the application server. Regardless of the application 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 requestor has already invoked using the CALL statement.

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 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 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.

For this statement to be successful, the application must be currently connected to the site at which the stored procedure was executed.

Examples

The statements in the following examples are assumed to be in PL/I programs.

Example 1: Use :loc1 and :loc2 to get the result set locator values for the two result sets returned by stored procedure P1:

```
EXEC SQL ASSOCIATE RESULT SET LOCATORS (:loc1, :loc2)
      WITH PROCEDURE P1;
```

Example 2: Use :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:

```
EXEC SQL ASSOCIATE LOCATORS (:loc1, :loc2)
      WITH PROCEDURE :hv1;
```

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.

This statement cannot be included in a REXX application program.

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 380.

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 must not contain statements other than host variable declarations or SQL INCLUDE statements that include host variable declarations.

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. CALL cannot be used if the program is executing as a stored procedure.

Authorization

The application server determines the privileges that are required and the authorization ID that must have those privileges. If the server is DB2 for OS/390, the privileges and authorization ID depend on the syntax of the CALL statement and the value of bind option DYNAMICRULES:

- If *either* of the following items is true:
 - The statement is of the form `CALL procedure-name`
 - The statement was bound with option `DYNAMICRULES(BIND)`

then the owner of the package or plan that contains the CALL statement must have one or more of the following privileges on each package that the stored procedure uses:

- The EXECUTE privilege on the package
 - Ownership of the package
 - PACKADM authority for the package collection
 - SYSADM authority
- If *both* of the following items are true:
 - The statement is of the form `CALL host-variable`
 - The statement was bound with option `DYNAMICRULES(RUN)`

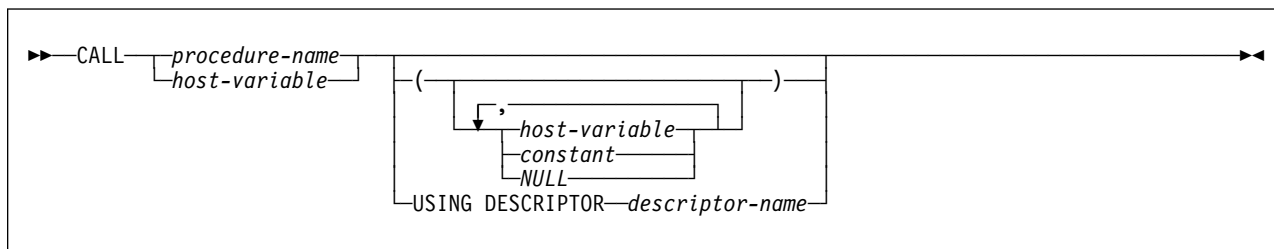
then the privilege set is the union of the privileges held by:

- The owner of the package or plan that contains the CALL statement
 - For ODBC or CLI applications, this is the owner of the package or plan associated with the ODBC or CLI driver.
- The primary SQL authorization ID of the application process
- The secondary SQL authorization IDs associated with the application process

The privilege set must include one or more of the following privileges on each package that the stored procedure uses:

- The EXECUTE privilege on the package
- Ownership of the package
- PACKADM authority for the package collection
- SYSADM authority

Syntax



Description

procedure-name or *host-variable*

Identifies the stored procedure to call 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 is a long SQL identifier that must be composed of SBCS characters:

- 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 next two parts identify the stored procedure. A period must separate each of the parts.
- A two-part procedure name is implicitly qualified by the location name of the current server. The name with its implicit qualifier identifies a stored procedure. A period must separate the two parts. The meaning of the first part depends on the application server. If the server is DB2 for OS/390, the first part must be SYSPROC.
- 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 application server. If the server is DB2 for OS/390, the implicit qualifier is SYSPROC. The name with its implicit qualifiers identifies a stored procedure.

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 be preceded by a colon and must not be followed by an indicator variable.
- The value of the host variable is a specification that depends on the application server. Regardless of the application 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

If the server is DB2 for OS/390, 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 application server. If the server is DB2 for OS/390, the last part or the only part of the name must be equal to some value of the PROCEDURE column of the catalog table SYSIBM.SYSPROCEDURES.

Parameters (*host-variable*, *constant*, or **NULL**)

Identifies a list of values to be passed as parameters to the procedure. If USING DESCRIPTOR is specified, each host variable described by the identified SQLDA is a parameter of the CALL. The *n*th parameter of the CALL corresponds to the *n*th parameter in the procedure, and the number of parameters in each must be the same.

Each parameter of a procedure is described at the server. In the case of a DB2 for OS/390 server, the information is in the SYSIBM.SYSPROCEDURES catalog table. In addition to attributes such as data type and length, the description of each parameter indicates how the procedure uses it:

- IN means only as an input value
- OUT means only as an output value
- INOUT means both as an input and an output value

When the CALL statement is executed, the value of each parameter of the CALL is assigned to the corresponding parameter of the procedure. Control is passed to the procedure according to the calling conventions of the host language. When execution of the procedure is complete, the value of each parameter of the procedure is assigned to the corresponding parameter of the CALL defined as OUT or INOUT. For details on the rules used to assign parameters, see “Rules for assigning parameters” on page 251.

host-variable

A parameter of the CALL is the identified host-variable. *host-variable* must identify a host variable (not a structure) described in the program according to the rules for declaring host variables, and the data type of the variable must be compatible with the data type of the corresponding parameter in the procedure. If an indicator variable is specified, its value must not be negative unless the description of the procedure allows for null parameters or the corresponding parameter of the procedure is defined as OUT.

constant

A parameter of the CALL is the specified value. The data type of the constant must be compatible with the corresponding parameter of the procedure and that parameter must be defined as IN.

NULL

A parameter of the CALL is the NULL value. The corresponding parameter of the procedure must be defined as IN and the description of the procedure must allow for null parameters.

USING DESCRIPTOR *descriptor-name*

Identifies an SQLDA that contains a valid description of host variables that are to be passed as parameters to the procedure. If the procedure has no parameters, an SQLDA is not used.

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 be 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 procedure.
 - SQLVAR occurrences to indicate the attributes of the variables.
- See “Identifying an SQLDA in C” on page 526 for how to represent *descriptor-name* in C.

Notes

Rules for assigning parameters: The rules for assigning the value of input and output parameters are similar.

Let S denote an input parameter of a CALL statement, let T denote the corresponding parameter of the stored procedure, and let C denote a column of a hypothetical single-row table such that the description of C is identical to the SYSIBM.SYSPROCEDURES description of T. S is assigned to T as follows:

1. The value of S is assigned to C in accordance with the “store assignment” rules of the SQL standard. These are the same rules as the DB2 rules for assigning values to columns with one exception. When the value is a string that is longer than the column, the SQL standard rules specify that an error occurs only if the excess characters are not blanks. If the excess characters are all blanks, they are discarded and the truncated string is assigned to the column without a warning.
2. The value of C is assigned to T in accordance with the DB2 rules for assigning values to host variables.

Let T denote an output parameter of a CALL statement, let S denote the corresponding parameter of the stored procedure, and let C denote a column of a hypothetical single-row table such that the description of C is identical to the SYSIBM.SYSPROCEDURES description of S. S is assigned to T as follows:

1. The value of S is assigned to C in accordance with the “store assignment” rules of the SQL standard described above for input parameters.
2. The value of C is assigned to T in accordance with the DB2 rules for assigning values to host variables.

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 application server can be the local DB2. In which case, packages are still required.

All values of all parameters are passed from the application requester to the application server. To improve the performance of this operation, host variables that

CALL

correspond to OUT parameters and have lengths of more than a few bytes should be set to null before the CALL statement is executed.

Example

A PL/I application has been precompiled on DB2 ALPHA and a package was created at DB2 BETA with the BIND subcommand. The SYSIBM.SYSPROCEDURES catalog table at BETA describes 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 in the application that runs at DB2 ALPHA are:

```
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 347 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 the DECLARE CURSOR statement. When the CLOSE statement is executed, the cursor must be in the open state.

Notes

Any open cursors of an application process not declared with the WITH HOLD 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 200.

CLOSE

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 DSN8510.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.

GET-REST-OF-DEPT.
EXEC SQL FETCH C1 INTO :DNUM, :DNAME, :MNUM END-EXEC.
```


COMMENT ON

The COMMENT ON statement adds or replaces comments in the descriptions of tables, views, aliases, or columns in the DB2 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.

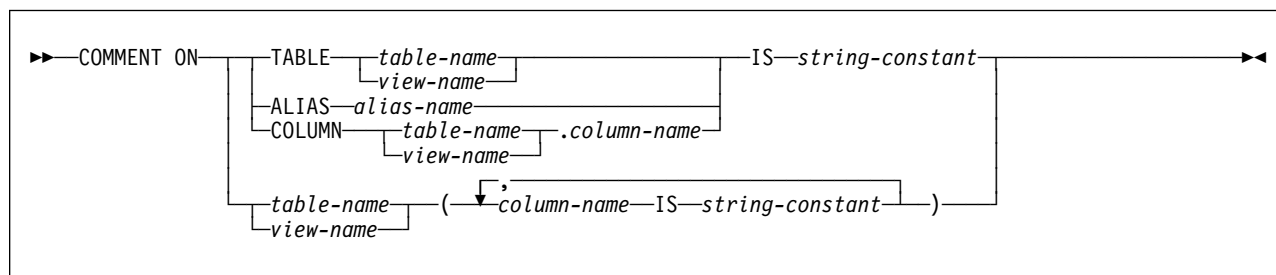
Authorization

The privilege set 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

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 and the bind option DYNAMICRULES(RUN) applies, the privilege set is the union of the privilege sets held by each authorization ID of the process. If the statement is dynamically prepared and the bind option DYNAMICRULES(BIND) applies, the privilege set is the privileges held by the authorization ID of the owner of the plan or package.

Syntax



Description

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 or view 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 table or view.

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.

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 comment is

COMMENT ON

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.

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 DSN8510.EMP.

```
COMMENT ON TABLE DSN8510.EMP  
IS 'REFLECTS 1ST QTR 81 REORG';
```

Example 2: Enter a comment on view DSN8510.VDEPT.

```
COMMENT ON TABLE DSN8510.VDEPT  
IS 'VIEW OF TABLE DSN8510.DEPT';
```

Example 3: Enter a comment on the DEPTNO column of table DSN8510.DEPT.

```
COMMENT ON COLUMN DSN8510.DEPT.DEPTNO  
IS 'DEPARTMENT ID - UNIQUE';
```

Example 4: Enter comments on two columns in table DSN8510.DEPT.

```
COMMENT ON DSN8510.DEPT  
(MGRNO IS 'EMPLOYEE NUMBER OF DEPARTMENT MANAGER',  
ADMDEPT IS 'DEPARTMENT NUMBER OF ADMINISTERING DEPARTMENT');
```

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.

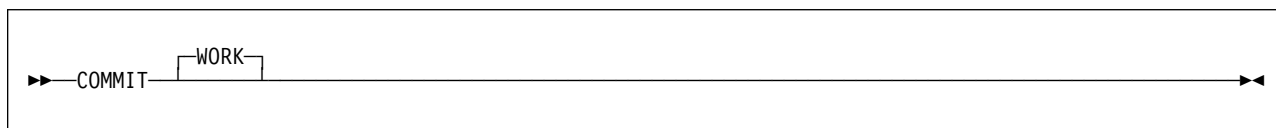
Invocation

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, or if the program is executing as a stored procedure.

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. SQL connections are ended when any of the following apply:

- The connection is in the release pending state
- The connection is not in the release pending state 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 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:

- SQL RELEASE has been issued for that site, or
- Bind option DISCONNECT(AUTOMATIC) was used, or

COMMIT

- Bind option DISCONNECT(CONDITIONAL) was used and there are no hold cursors for that site.

All implicitly acquired locks are released, except for those required for the cursors that were not closed. See “LOCK TABLE” on page 426 for an explanation of the duration of explicitly acquired locks.

All rows of every temporary table of the application process are deleted with the exception that the rows of a temporary table are not deleted if any program in the application process has an open WITH HOLD cursor that is dependent on that temporary table. In addition, if RELEASE(COMMIT) is in effect, the logical work files for those temporary tables whose rows are deleted are also deleted.

Notes

The SQL COMMIT statement cannot be used in the IMS or CICS environment. To effect 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.

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. “When an Application Process Has a Current Server” on page 260 describes what happens when the process has a current server.

CONNECT (Type 1) and CONNECT (Type 2) Differences

There are two types of CONNECT statements with the 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 262

“CONNECT (Type 2)” on page 267

The following table summarizes the differences between CONNECT (Type 1) and CONNECT (Type 2) rules:

Table 16 (Page 1 of 2). 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 for any other reason, the application process is placed in the unconnected state.	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.
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.

Table 16 (Page 2 of 2). CONNECT (Type 1) and CONNECT (Type 2) Differences

Type 1 Rules	Type 2 Rules
A CONNECT to the current application server is executed like any other CONNECT (Type 1) statement.	<p>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 application server is an error. For example, an error occurs if the first CONNECT is a CONNECT TO x where x is the local DB2.</p> <p>If the SQLRULES(DB2) bind option is in effect, a CONNECT to an existing SQL connection is not an error. Thus, if x is an existing SQL connection of the application process, CONNECT TO x makes x its current connection. If x is already the current connection, CONNECT TO x has no effect on the state of any connections.</p>

Determining the CONNECT rules that apply: The following table explains how to determine the CONNECT rules that apply:

Table 17. 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 will occur 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 which was bound at that server. However, this does not apply to SQL statements such as CONNECT and RELEASE which 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 470.

- 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 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 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 259.

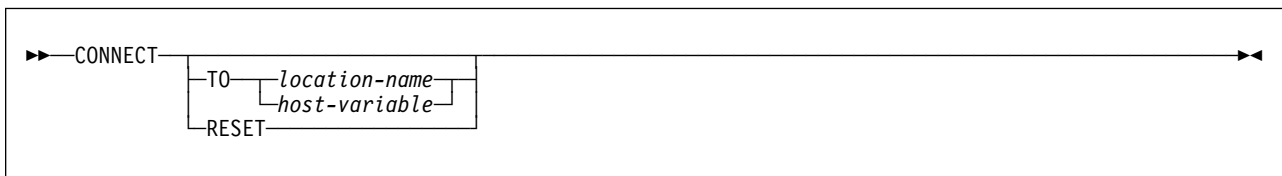
Invocation

This statement can only be embedded in an application program. It is an executable statement that cannot be dynamically prepared. CONNECT cannot be used in a program executing as a stored procedure.

Authorization

The primary authorization ID of the process must be authorized to connect to the identified server. That server performs the authorization check and determines the specific authorization required. See Section 3 (Volume 1) of *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 may be up to 17 bytes long.)
- It must be preceded by a colon and 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 264.)

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 application server is an IBM relational database product, the information has the form *pppvrrm*, where:
 - *ppp* is:
 - ARI for SQL/DS
 - DSN for DB2 for MVS
 - QSQ for OS/400
 - SQL for all other DB2 products
 - *vv* is a two-digit version identifier such as '05'.
 - *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 5 of DB2 for OS/390, the value of SQLERRP is 'DSN05010'.

If execution of the CONNECT statement is unsuccessful, the SQLERRP field of the SQLCA is set to the name of the DB2 application 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.

CONNECT RESET

CONNECT RESET is equivalent to CONNECT TO *x* where *x* is the location name of the local DB2 subsystem.

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

Connection states: In the following description of the connection states, CONNECT means CONNECT TO or CONNECT RESET, 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:

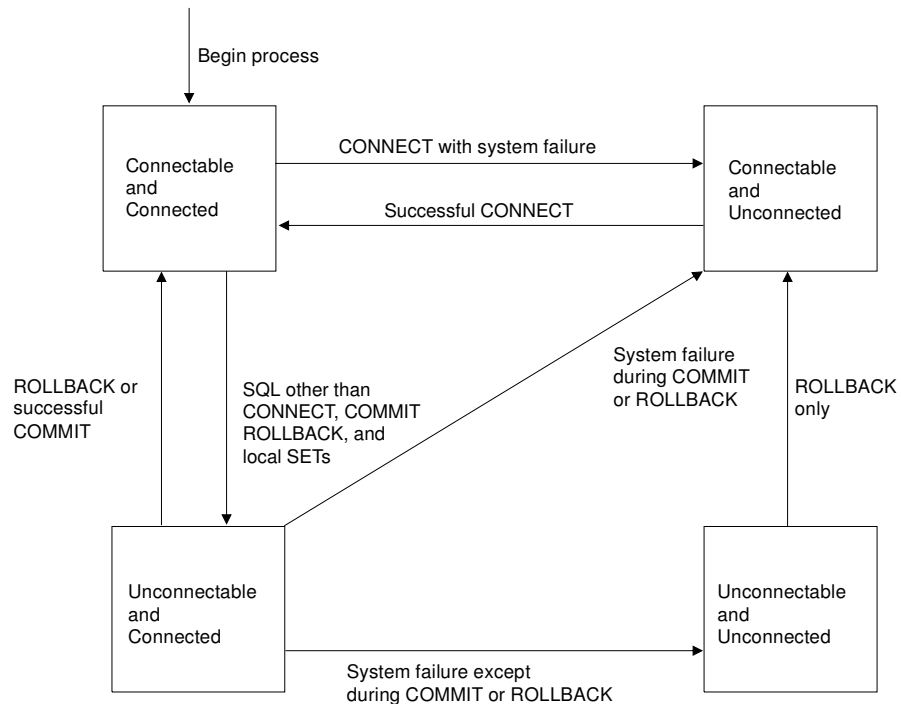


Figure 5. Connect state transitions

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.

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 will result 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 will result 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 will delay 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.

CONNECT (Type 1)

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 SQLERRP 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.

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 259. Refer to “Connection Management for DB2 Private Protocol and DRDA Access” on page 33 for more information about connection states.

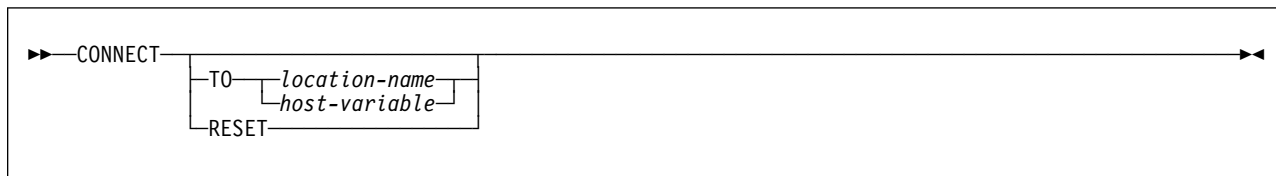
Invocation

This statement can only be embedded in an application program. It is an executable statement that cannot be dynamically prepared. CONNECT cannot be used if the program is executing as a stored procedure.

Authorization

The primary authorization ID of the process must be authorized to connect to the identified server. The authorization check is performed by the application server when the statement is executed, and the specific authorization required is determined by that server. See Section 3 (Volume 1) of *Administration Guide* for further information.

Syntax



Description

TO *location-name* or *host-variable*

Identifies the application 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 may be up to 17 bytes long.)
- It must be preceded by a colon and 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.

CONNECT (Type 2)

S must identify an application 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.

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 application 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.
- S is placed in the CURRENT SERVER special register.
- Information about application server S is placed in the SQLERRP field of the SQLCA. If the application server is an IBM relational database product, the information has the form *pppvrrm*, where:
 - *ppp* is:
 - ARI for SQL/DS
 - DSN for DB2 for MVS
 - QSQ for OS/400
 - SQL for all other DB2 products
 - *vv* is a two-digit version identifier such as '05'.
 - *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 5 of DB2 for OS/390, the value of SQLERRP is 'DSN05010'.

If the CONNECT statement is unsuccessful, the connection state of the application process and the states of its SQL connections are unchanged.

CONNECT 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.

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.

Example

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)
```

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. However, if the bind option DYNAMICRULES(BIND) applies, the statement cannot be dynamically prepared.

Authorization

The privilege set defined below must include at least one of the following:

- The CREATEALIAS privilege
- SYSADM or SYSCTRL authority

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 specified alias name includes a qualifier that is not the same as this authorization ID, the privilege set must include SYSADM or SYSCTRL authority.

If the statement is dynamically prepared, the privilege set is the privileges 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, or
- The qualifier must be the same as one of the authorization IDs of the process and the privileges held by that authorization ID must include the CREATEALIAS privilege. This is an exception to the rule that the privilege set is the privileges 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 re-bound. 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. 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 containing 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.

Example

Create an alias for a catalog table at a DB2 with location name DB2USCALABOA5281.

```
CREATE ALIAS LATABLES FOR DB2USCALABOA5281.SYSIBM.SYSTABLES;
```

CREATE DATABASE

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. However, if the bind option DYNAMICRULES(BIND) applies, the statement cannot be dynamically prepared.

Authorization

The privilege set defined below must include at least one of the following:

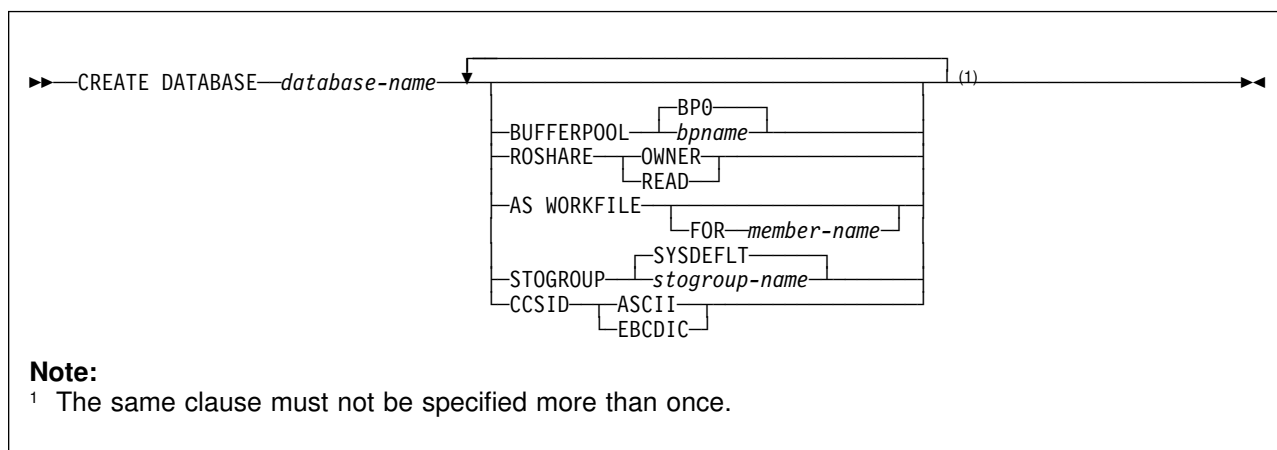
- The CREATEDBA privilege
- The CREATEDBC privilege
- SYSADM or SYSCTRL authority

If ROSHARE is specified, the privilege set must include SYSADM or SYSCTRL authority.

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 privileges held by the SQL authorization ID of the process.

See “Notes” on page 274 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 for table spaces and indexes within the database. The default is BP0.

32KB buffer pools apply only to table spaces. If a 32KB buffer pool name is specified, the default buffer pool for indexes in the database is BP0.

See “Naming Conventions” on page 48 for more details about *bpname*.

ROSHARE

Indicates how the database will be shared using *shared read-only data*. If ROSHARE is omitted, the database will not be shared. For an explanation of shared read-only data, see Appendix F (Volume 2) of *Administration Guide*.

OWNER The database will be shared, and the current server will be the DB2 that can update the database.

READ The current server is to have read-only access to the database through shared read-only data.

AS WORKFILE

Indicates that this is a work file database. In a non-data-sharing environment, the clause is ignored.

FOR *member-name*

Specifies the member for which this database is to be a work file. Only one work file database can be created for each DB2 subsystem.

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 work file databases. If you specify AS WORKFILE, do not use the CCSID clause.

STOGROUP *stogroup-name*

Names 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 specified during installation.

EBCDIC Specifies that the data must be encoded using the EBCDIC CCSIDs specified during installation.

Usually, each encoding scheme requires only a single CCSID. Additional CCSIDs are needed when mixed or graphic 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 clause.

CREATE DATABASE

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 DSN8D51P. DSN8G510 is the default storage group to be used for table spaces and indexes in the database. BP2 is the default buffer pool to be used for table spaces and indexes in the database.

```
CREATE DATABASE DSN8D51P
  STOGROUP DSN8G510
  BUFFERPOOL BP2;
```

Example 2: Create database DSN8TEMP. Use the default DB2 storage group and buffer pool and no shared read-only data.

```
CREATE DATABASE DSN8TEMP;
```

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

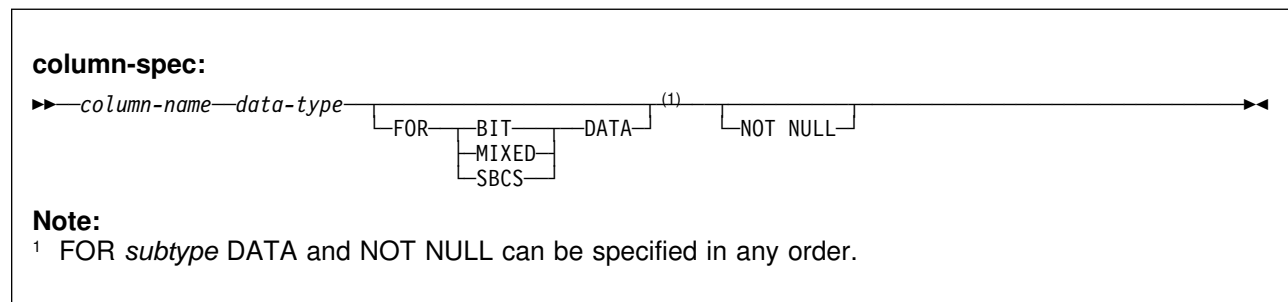
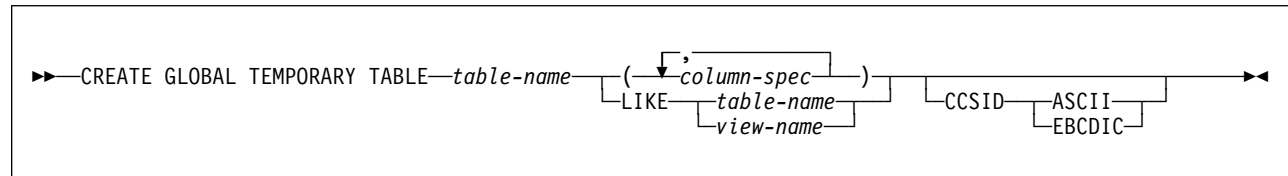
The privilege set defined below must include at least one of the following:

- The CREATETMTAB system privilege
- The CREATETAB database privilege for any database
- DBADM, DBCTRL, or DBMAINT authority for any database
- SYSADM or SYSCTRL authority

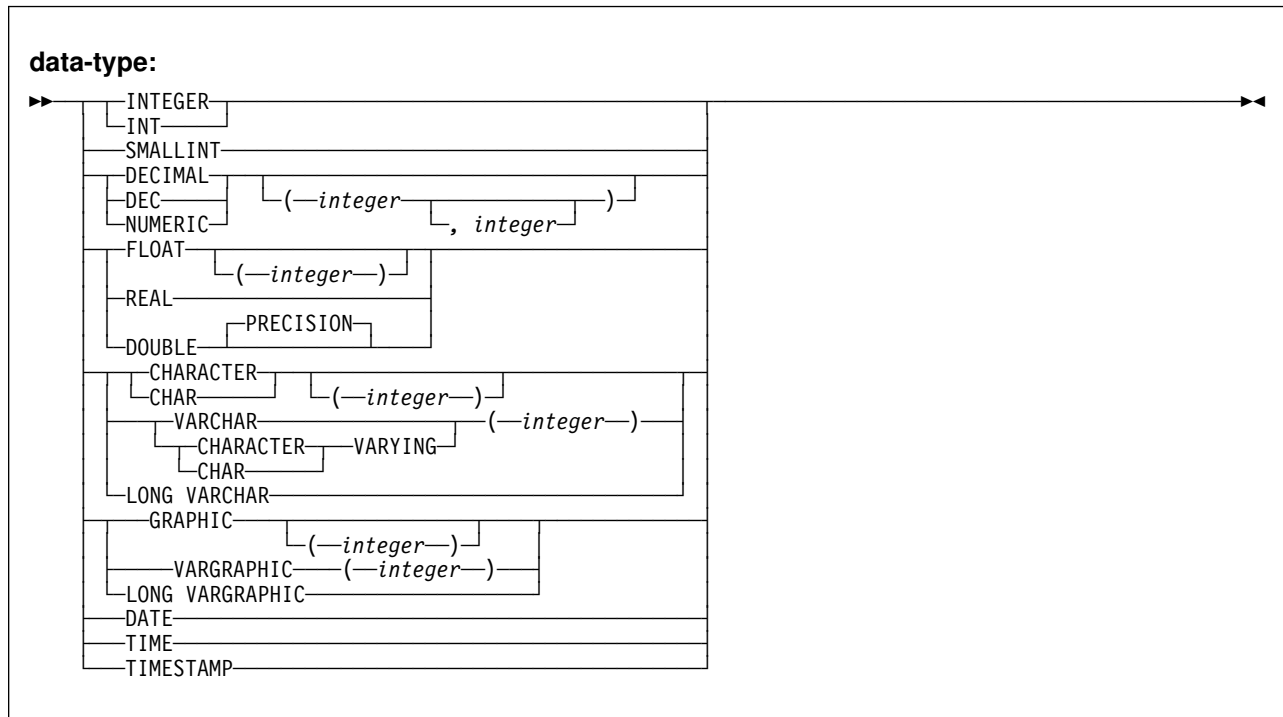
If the LIKE keyword is used, additional authorization might be required, as explained in the description of that clause.

Privilege Set: The privilege set is the same as the privilege set for the CREATE TABLE statement. See "Authorization" on page 308 for details.

Syntax



CREATE GLOBAL TEMPORARY TABLE



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 application server.

The qualification rules for *table-name* are the same as for *table-name* in the CREATE TABLE statement. (See “table-name” on page 311.)

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 types allowed and the rules are the same as those for the CREATE TABLE statement. See “data-type” on page 311.

FOR subtype DATA

Specifies a subtype for a character string column. The subtypes allowed and the rules followed are the same as those for the CREATE TABLE statement. See “FOR subtype DATA” on page 313.

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 data stored in the table.

ASCII Specifies that the data must be encoded by using the ASCII CCSIDs that are specified during installation.

EBCDIC Specifies that the data must be encoded by using the EBCDIC CCSIDs that are specified during installation.

Usually, each encoding scheme requires only a single CCSID. Additional CCSIDs are needed when mixed or graphic data is used.

The option defaults to the value of field DEF ENCODING SCHEME on installation panel DSNTIPF.²²

An error is issued if a valid ASCII CCSID has not been specified for the installation.

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.

²² When you use the LIKE clause with the CREATE GLOBAL TEMP TABLE statement, the encoding scheme of the table that you are copying is not used to create the new table.

CREATE GLOBAL TEMPORARY TABLE

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 application 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.
- 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 OF 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 (that is, through a DB2 private protocol) can be accessed by another SQL statement using the same three-part 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:

DROP TABLE	SELECT INTO	CREATE TABLE LIKE
CREATE VIEW	LABEL ON	DESCRIBE TABLE
COMMENT ON	CREATE ALIAS	DECLARE TABLE
INSERT	CREATE SYNONYM	

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 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 partitioned or nonpartitioned 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. However, if the bind option DYNAMICRULES(BIND) applies, the statement cannot be dynamically prepared.

Authorization

The privilege set 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 containing the table
- SYSADM or SYSCTRL authority

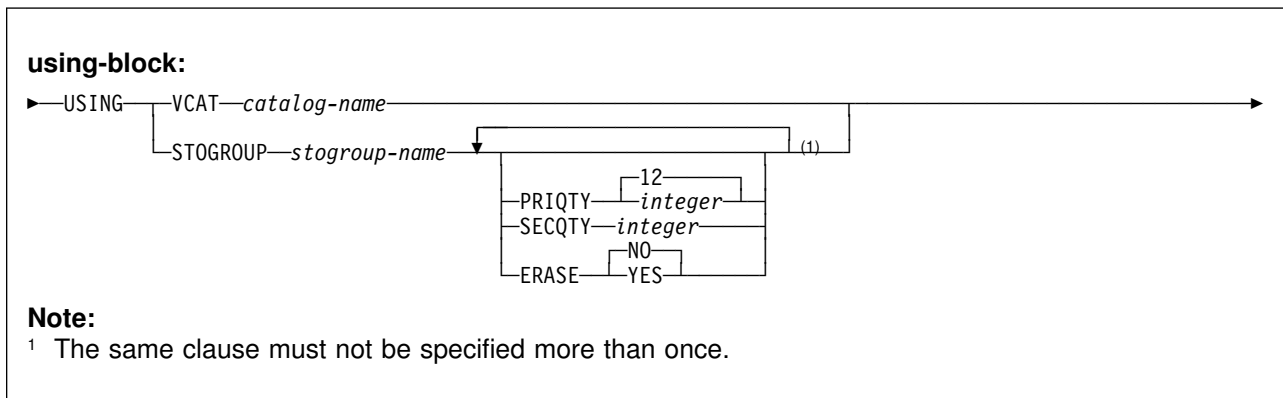
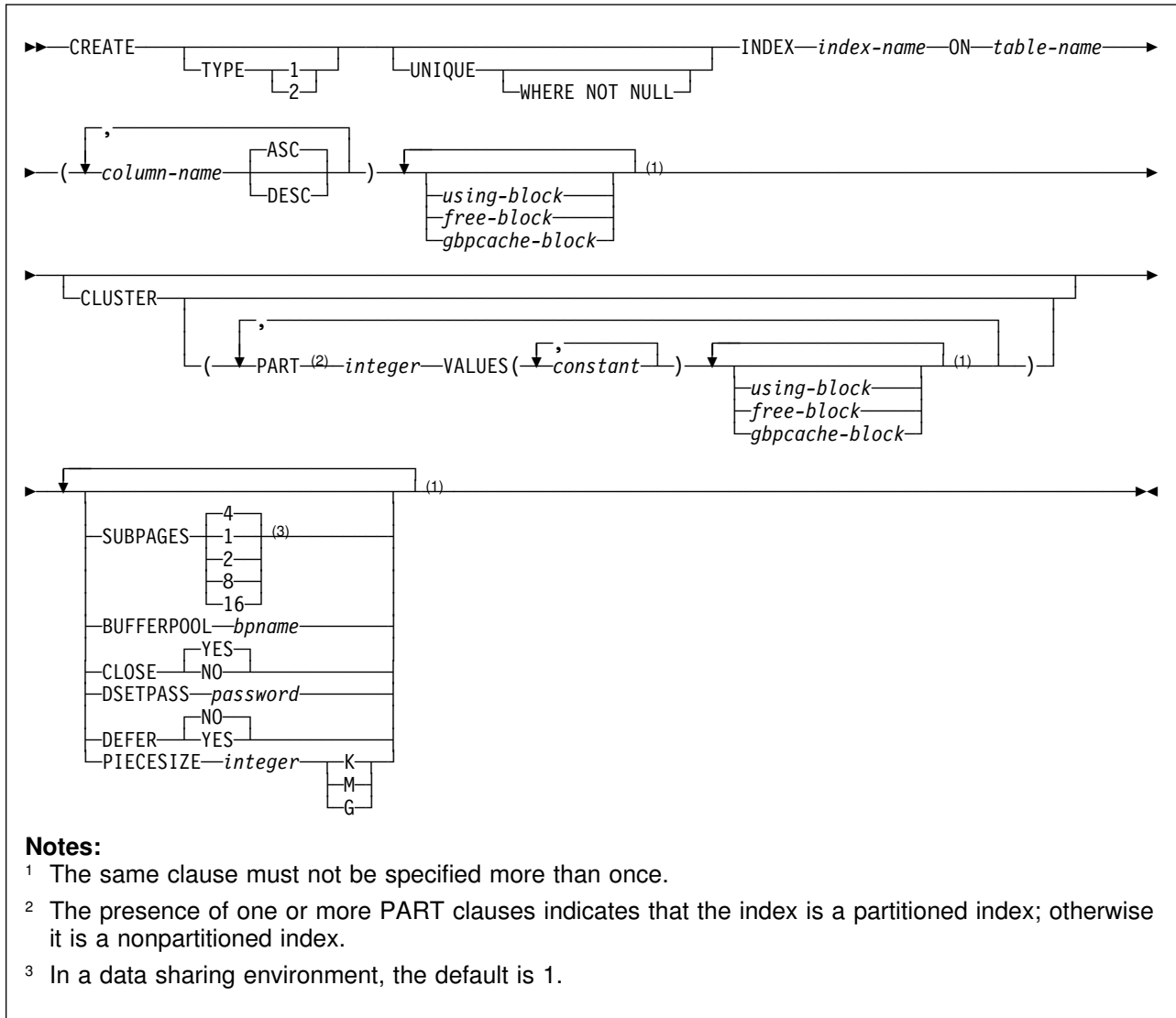
Additional privileges may be needed, as explained in the description of the BUFFERPOOL and USING STOGROUP clauses.

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 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 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:

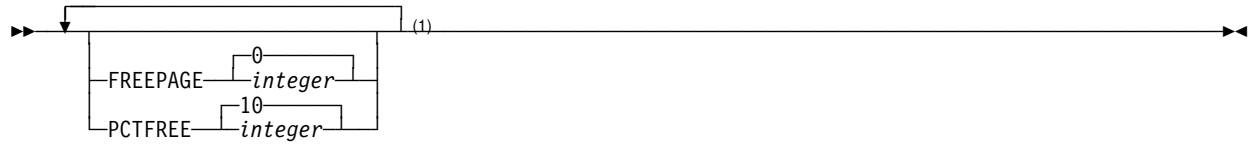
- If the privilege set includes SYSADM or SYSCTRL authority, or DBADM or DBCTRL authority for the database, any qualifier is valid.
- If the privilege set includes none of these authorities, the qualifier is valid only if it is the same as one of the authorization IDs of the process and the privilege set 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 held by the SQL authorization ID of the process.

Syntax



CREATE INDEX

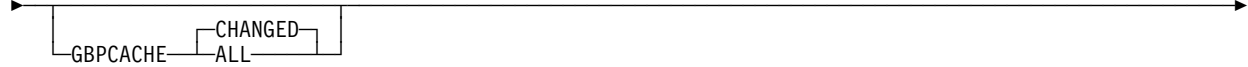
free-block:



Note:

¹ The same clause must not be specified more than once.

gbpcache-block:



Description

TYPE *n*

Specifies the type of index, 1 or 2.

Do not specify TYPE 1 if the table space containing the identified table:

- Has a LOCKSIZE of ROW
- Is a large partitioned table space
- Has an ASCII encoding scheme

If you specify TYPE 2, any specification of SUBPAGES is ignored, and a warning message is issued. If you specify TYPE 1 in a data sharing environment, a warning message is issued if you specify a value greater than 1 for SUBPAGES. A type 1 index with more than one subpage cannot be accessed when there is inter-DB2 R/W interest in the index.

If you do not specify TYPE, its default value is TYPE 2 if the table space containing the table:

- Has a LOCKSIZE of ROW
- Is a large partitioned table space
- Has an ASCII encoding scheme

In all other cases, its default value is the type specified in field DEFAULT INDEX TYPE on installation panel DSNTIPE. The default for the field on that panel is TYPE 2.

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.

- 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. You can specify WHERE NOT NULL only if TYPE is 2, either explicitly or by default.

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 Section 2 (Volume 1) of *Administration Guide* for more information about using the RECOVER 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, its status can return to incomplete only by the dropping of its primary index; in that case K is the primary key of the table and n is one.)
- 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.

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.

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 re-bound. If QUALIFIER was not used, the owner of the index is the owner of the package or plan.

CREATE INDEX

If the index name is unqualified and the statement is dynamically prepared, the SQL authorization ID is the owner of the index.

ON *table-name*

Identifies the table on which the index is created. The name must identify a table that exists at the current server. The name must not identify a temporary table.

If qualified, *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.

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 or the same column more than once. Do not qualify *column-name*.

The sum of the length attributes of the columns must not be greater than $m-n$, where:

- n is the number of columns that can contain null values
- m depends on the number of subpages and whether the index has the attribute UNIQUE:

Table 18. Values of m

SUBPAGES	UNIQUE	m^1
<8	-	254
8	Yes	241
8	No	238
16	Yes	114
16	No	111

Notes:

1. The values of m shown are for type 1 indexes. For type 2 indexes, the values of m are the values shown plus 1.

ASC

Puts the index entries in ascending order by the column. This is the default.

DESC

Puts the index entries in descending order by the column.

using-block

The components of the USING clause are discussed below, first for nonpartitioned indexes and then for partitioned indexes.

Using Clause for Nonpartitioned Indexes

For nonpartitioned 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 Section 2 (Volume 1) of *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. However, the same *catalog-name* must be used by the subsystems when shared read-only data is used.

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 $PRIQTY + 118 \times SECQTY$ 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

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
131068	If <i>integer</i> is greater than 131068
<i>integer</i>	If <i>integer</i> is not greater than 131068

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 (4K), whichever is larger. However, if this value exceeds 131068, the default is 131068.

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. 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 Indexes:

If the index is partitioned, 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 an nonpartitioned 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 Section 2 (Volume 1) of *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. However, the same *catalog-name* must be used by the subsystems when shared read-only data is used.

DB2 assumes one and only one data set for each partition.

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.

PCTFREE *integer*

Determines the percentage of free space to leave in each nonleaf page and subpage 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 or subpage is loaded without restriction. When additional entries are placed in a nonleaf page, the percentage of free space is at least as great as *integer*. When additional entries are placed in a leaf page, the percentage of free space is at least as great as $integer/m$, where *m* is the number of subpages.

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.

If the index is partitioned, 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 _____

_____ gpbcache-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.
GBPCACHE CHANGED is the default.

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.

In a data sharing environment, hiperpools are not used for indexes or partitions that are defined with GBPCACHE ALL.

If the index 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 *gpbcache-block* in any PART clause.
2. The value given in a *gpbcache-block* that is not in any PART clause.
3. The default value is CHANGED.

End of gpbcache-block

CLUSTER

Specifies that the index is the clustering index of the table. Do not use CLUSTER 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 clustering 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 clustering 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 reorgnized in cluster order.

PART *integer*

A PART clause specifies the highest value of the index key in one partition of a partitioned 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*.

If the key of a partitioned index is longer than 40 bytes, only the first 40 bytes are used to determine the high value for each partition.

VALUES(*constant*,...)

You must use at least one constant after VALUES in each PART clause. You can use as many as there are columns in the key. The concatenation

of all the constants is the highest value of the key in the corresponding partition of the index.

The use of the constants to define key values is subject to these rules:

#

- The first constant corresponds to the first column of the key, the second constant to the second column, and so on. Each constant must have the same data type as its corresponding column. For example, if a column has a decimal data type, the constant must include a decimal point (.).
- 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'.
- Using fewer constants 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. If the key is longer than 40 bytes, this rule applies only to the first 40 bytes.
- The highest value of the key in the last partition depends on the type of table space. For table spaces that are not large partitioned table spaces, the constants 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 large partitioned table spaces, the constants you specify are enforced. The value specified for the last partition 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.

SUBPAGES *integer*

Gives the number of subpages for each physical page. (The subpage is the unit of index locking.)

SUBPAGES is valid for TYPE 1 indexes. If you specify TYPE 2, SUBPAGES is ignored and a warning message is issued.

Use 1, 2, 4, 8, or 16. The default is 4, except in a data sharing environment when it is 1. In a data sharing environment, you must specify 1 for type 1 indexes to be shared; when there is more than one subpage, an index cannot be accessed when there is inter-DB2 R/W interest in the index.

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 of the database. If the default buffer pool of the database is a 32KB page buffer pool, the default is BP0.

See “Naming Conventions” on page 48 for more details about *bpname*. See Chapter 2 of *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.

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.

DSETPASS *password*

Specifies a master level password sent to access method services when the data sets of the index are used by DB2. *password* is a short identifier. If delimited, *password* can contain any characters acceptable to access method services.

If you use a storage group, *password* is the password that protects the data sets as well as the password that is passed to access method services when the data sets are used by DB2. If you do not use a storage group, you define the password that protects the data sets through access method services.

If the index occupies more than one data set, all of its data sets that are password-protected must have the same password.

The password does not apply to the data sets managed by Storage Management Subsystem. To protect data sets defined to SMS, use RACF or a similar external security system.

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 recover pending state. Refer to Section 2 (Volume 1) of *Administration Guide* for more information about using DEFER.

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 recover pending state to indicate that the index must be recovered by the RECOVER INDEX utility. A warning message is issued to inform the user that the index has been placed in a recover pending state.

PIECESIZE *integer*

Specifies the maximum addressability of each piece (data set) for a nonpartitioned index. The subsequent keyword K, M, or G, indicates the units of the value specified in *integer*.

CREATE INDEX

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 4 194 304.

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 4 096.

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 4.

In the above specification for piece size, spaces are permitted between the integer and the K, M, or G. They are not required.

Valid values for piece size are as follows:

256 K
512 K
1024 K (or 1 M)
2048 K (or 2 M)
4096 K (or 4 M)
8192 K (or 8 M)
16384 K (or 16 M)
32768 K (or 32 M)
65536 K (or 64 M)
131072 K (or 128 M)
262144 K (or 256 M)
524288 K (or 512 M)
1048576 K (or 1024 M or 1 G)
2097152 K (or 2048 M or 2 G)
4194304 K (or 4096 M or 4 G)²³

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 backed by non-LARGE table spaces and 4 G (4 GB) for indexes backed by LARGE table spaces.

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.

²³ Only valid for LARGE table spaces.

EBCDIC and ASCII encoding schemes for an index: An index has the same encoding scheme as its associated table. For an ASCII table, all the indexes must be defined as type 2; the indexes are stored in ASCII order.

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 nonpartitioned index, and your nonpartitioning index is 10 MB (and not likely to grow much), specify PIECESIZE 2 M. If your nonpartitioned index is likely to grow, choose a larger value. Remember that 32 pieces is the limit if the underlying tablespace is not defined as LARGE and that 128 is the limit if the underlying tablespace is defined as LARGE.

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: Partitioned indexes can only be dropped with a DROP of the associated table space, whereas nonpartitioned indexes can be dropped with the DROP INDEX statement.

Creating indexes on DB2 catalog tables: For details on creating indexes on catalog tables, see “SQL Statements Allowed on the Catalog” on page 532.

Examples

Example 1: Create a unique index, named DSN8510.XDEPT1, on table DSN8510.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 DSN8G510. 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.

Use 8 subpages for each physical page and associate the index with buffer pool BP1. 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. The VSAM password for the data sets is OSESAME. The maximum for each data set is 1 megabyte .

```
CREATE UNIQUE INDEX DSN8510.XDEPT1
ON DSN8510.DEPT
(DEPTNO ASC)
USING STOGROUP DSN8G510
PRIQTY 512
SECQTY 64
ERASE NO
SUBPAGES 8
BUFFERPOOL BP1
CLOSE YES
DSETPASS OSESAME;
PIECESIZE 1 M;
```

Example 2: Create a cluster index, named XEMP2, on table EMP in database DSN8510. Put the entries in ascending order by column EMPNO. Let DB2 define the data sets for each partition using storage group DSN8G510. Make the primary space allocation be 36 kilobytes, and allow DB2 to use the default value for

CREATE INDEX

SECQTY, which for this example is 3 times 4KB (12 kilobytes). 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.

Use 8 subpages for each physical page, and associate the index with buffer pool BP1. The data sets can be closed when no one is using the index. The VSAM password for the data sets is OSESAME.

```
CREATE INDEX DSN8510.XEMP2
ON DSN8510.EMP
(EMPNO ASC)
USING STOGROUP DSN8G510
PRIQTY 36
ERASE NO
SUBPAGES 8
CLUSTER
(PART 1 VALUES('H99'),
PART 2 VALUES('P99'),
PART 3 VALUES('Z99'),
PART 4 VALUES('999'))
BUFFERPOOL BP1
CLOSE YES
DSETPASS OSESAME;
```

|
Example 3: Create a nonpartitioned index, named DSN8510.XDEPT1, on table
DSN8510.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. Specify the value in terms of kilobytes.

```
|  
|  
|  
#  
|  
CREATE TYPE 2 UNIQUE INDEX DSN8510.XDEPT1  
ON DSN8510.DEPT  
(DEPTNO ASC)  
USING VCAT DSNCAT  
PIECESIZE 1048576 K;
```


CREATE PROCEDURE (SQL procedure)

The CREATE PROCEDURE statement specifies the source statements for an SQL procedure.
 #

Invocation

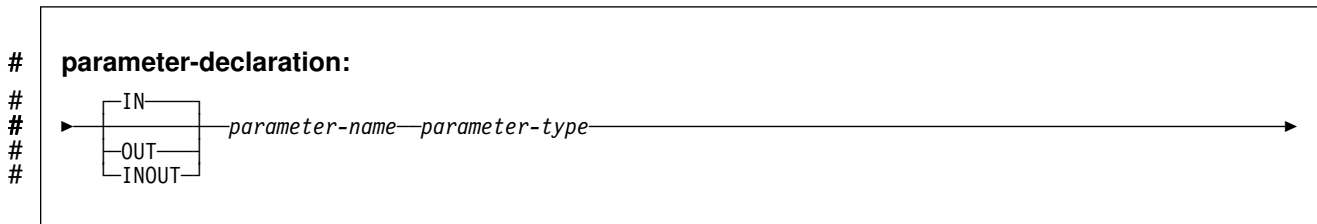
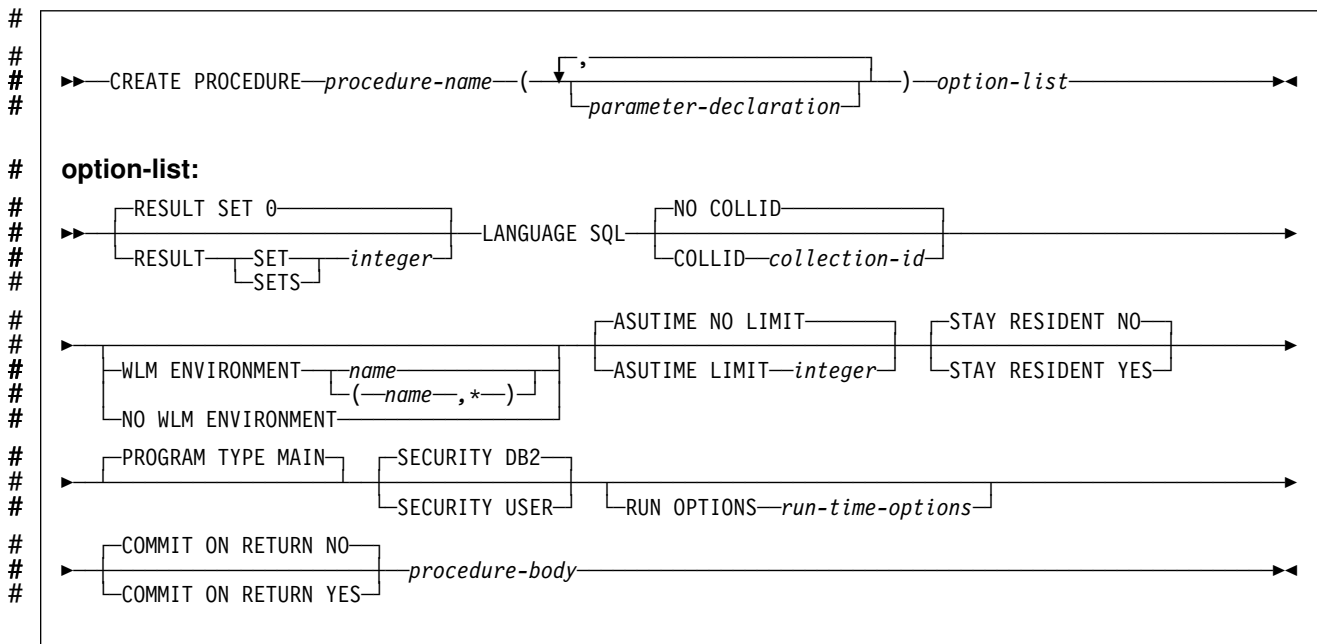
This statement cannot be embedded in an application program or dynamically prepared. This statement can appear in the following places:
 #

- # • As the only statement in a partitioned data set member that is input to the DB2 precompiler or the SQL procedure processor
- #
- # • As the only statement in a character string that is an input parameter for the SQL procedure processor
- #

Authorization

None required.

Syntax



CREATE PROCEDURE (SQL procedure)

```
# parameter-type:
```

```
# ▶—built-in-data-type—▶▶
```

```
# data-type:
```

```
# ▶—built-in-data-type—▶▶
```

```
# built-in-data-type:
```

```
# ▶—SMALLINT—▶▶  
# |  
# |—INTEGER—|  
# |—INT—|  
# |  
# |—DECIMAL—|  
# |—DEC—| (—integer—, integer—)|  
# |—NUMERIC—|  
# |—FLOAT—| (—integer—)|  
# |  
# |—REAL—|  
# |—PRECISION—|  
# |—DOUBLE—|  
# |  
# |—CHARACTER—| (—integer—)| FOR—SBCS—DATA—|  
# |—CHAR—| (—integer—)|  
# |—CHARACTER—VARYING—| (—integer—)|  
# |—CHAR—|  
# |—VARCHAR—|  
# |  
# |—GRAPHIC—| (—integer—)|  
# |—VARGRAPHIC—| (—integer—)|  
# |  
# |—DATE—|  
# |—TIME—|  
# |—TIMESTAMP—|
```

Description

```
# procedure-name
```

```
# Names the stored procedure. The name is an unqualified long SQL identifier that must not identify an existing stored procedure at the current server.
```

```
# The default load module name for the stored procedure is one of the following strings:
```

- If the stored procedure name is eight or fewer bytes, the load module name is the stored procedure name.
- If the stored procedure name is more than eight bytes, DB2 generates a load module name. You can determine that name by looking in the EXTERNAL_NAME column of SYSIBM.SYSROUTINES.

```
# If you use the default load module name, the SQL procedure name must conform to MVS naming conventions for partitioned data set members.
```

```

#           (parameter-declaration,...)
#           Specifies the number of parameters of the stored procedure and the data type
#           of each parameter. A parameter for a stored procedure can be used only for
#           input, only for output, or for both input and output. You must 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 short SQL identifier, which can
#           include only the characters A through Z, 0 through 9, or characters that
#           correspond to the EBCDIC code points X'5B', X'7B', and X'7C', which
#           correspond to $, #, and @ in code page 37 or 500. A parameter name
#           cannot be an SQL reserved word. For a list of SQL reserved words, see
#           Appendix E, "SQL Reserved Words" on page 621.
#
#           data-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.
#
#           The NUMERIC, DATE, TIME, and TIMESTAMP data types are valid in
#           a CREATE PROCEDURE statement, but they are not valid data types
#           for the PARMLIST column of SYSIBM.SYSPROCEDURES. When you
#           define your SQL procedure to DB2 by inserting a row into
#           SYSIBM.SYSPROCEDURES, you need to make the following
#           substitutions:

```

CREATE PROCEDURE (SQL procedure)

```
# Table 19. Substitutions for NUMERIC, DATE, TIME, and TIMESTAMP in
# SYSIBM.SYSPROCEDURES
#
# For this data type in CREATE PROCEDURE          Substitute this data type in
#                                                  SYSPROCEDURES
#-----
# NUMERIC                                         DECIMAL
#-----
# DATE                                             VARCHAR(10)1
#-----
# TIME                                             VARCHAR(8)2
#-----
# TIMESTAMP                                       VARCHAR(26)
#-----
#
# Notes:
#
# 1. If a date exit is installed on the DB2 subsystem, specify VARCHAR(n), where n is the
# length value from field LOCAL DATE LENGTH on installation panel DSNTIP4.
#
# 2. If a time exit is installed on the DB2 subsystem, specify VARCHAR(n), where n is the
# length value from field LOCAL TIME LENGTH on installation panel DSNTIP4.
#
#
# 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 311.
#
# 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)
#
# 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. 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.
#
# RESULT SET integer or RESULT SETS integer
# Specifies the maximum number of query result sets that the stored procedure
# can return. The default is RESULT SETS 0, which indicates that there are no
# result sets.
#
# LANGUAGE
# Specifies the application programming language in which the stored procedure
# is written.
#
# SQL
# The stored procedure is written in DB2 SQL procedure language.
#
# NO COLLID or COLLID collection-id
# Identifies the package collection that is 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 that
must not contain an underscore.
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.

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 clause.
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.

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*.

CREATE PROCEDURE (SQL procedure)

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 2 GB.
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. Only PROGRAM TYPE MAIN
is allowed for an SQL procedure.

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.

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.

#           procedure-body
#           Specifies the source code for an SQL procedure. See “Chapter 7. SQL
#           procedure statements” on page 485 for information on how to write a
#           procedure body.

```

Notes

```

#           The following restrictions apply to the use of parameters in SQL procedures:
#
#           • If IN is specified for a parameter in an SQL procedure, the parameter cannot
#           be modified within the SQL procedure body.
#
#           • If OUT is specified for a parameter in an SQL procedure, the parameter can be
#           used only as the target of an assignment in the SQL procedure body. The
#           parameter cannot be checked or used to set other variables. If the parameter is
#           not set, DB2 returns the null value to the caller.

```

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
#           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 (SQL procedure)

```
#          CREATE PROCEDURE UPDATE_SALARY_1
#          (IN EMPLOYEE_NUMBER CHAR(10),
#          IN RATE DECIMAL(6,2))
#          LANGUAGE SQL
#          WLM ENVIRONMENT PARTSA
#          COMMIT ON RETURN YES
#          RUN OPTIONS 'MSGFILE(OUTFILE),RPTSTG(ON),RPTOPTS(ON) '
#          UPDATE EMP
#          SET SALARY = SALARY * RATE
#          WHERE EMPNO = EMPLOYEE_NUMBER

#          For more examples of SQL procedures, see "Chapter 7. SQL procedure
#          statements" on page 485.
```


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. However, if the bind option DYNAMICRULES(BIND) applies, the statement cannot be dynamically prepared.

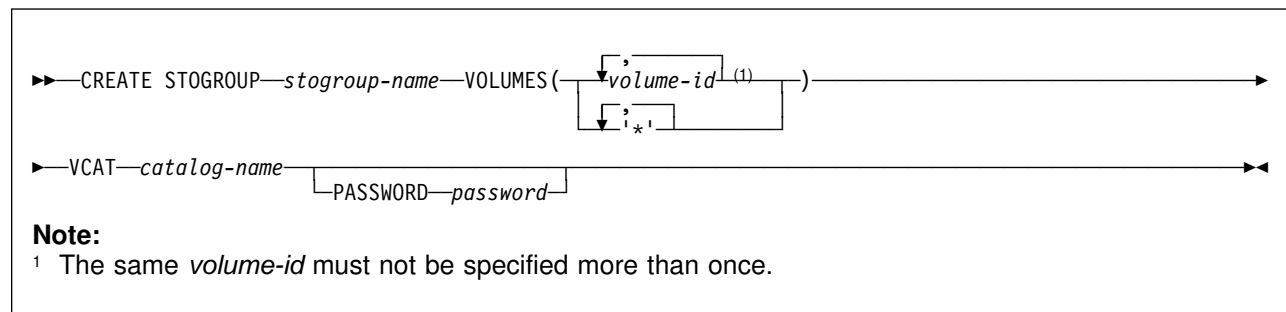
Authorization

The privilege set defined below must include at least one of the following:

- The CREATESG privilege
- SYSADM or SYSCTRL authority

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 privileges held by the SQL authorization ID of the process.

Syntax



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).
Contact your site's storage administrator to determine if the SMS Guaranteed
Space attribute applies. If SMS Guaranteed Space does not apply for
SMS-managed data sets, it is recommended that the VOLUMES clause be
specified with one asterisk, VOLUMES(' * '). If SMS Guaranteed Space does
apply, contact your site storage manager and refer to *DFSMS/MVS: Access*
Method Services for the Integrated Catalog and *DFSMS/MVS: Storage*

CREATE STOGROUP

Administration Reference for DFSMSdfp for information on how to specify the
VOLUMES clause.

To have Storage Management Subsystem (SMS) manage the extension of data sets for shared read-only data, list more than one asterisk (*) after VOLUMES, as in VOLUMES('*', '*', '*'). SMS uses one volume for each asterisk in the list. See Section 2 (Volume 1) of *Administration Guide* for considerations for using SMS to manage data sets.

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 Section 2 (Volume 1) of *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. However, the same *catalog-name* must be used by the subsystems when shared read-only data is used.

PASSWORD *password*

Gives a VSAM control or master level password in the form of a short identifier. If the password is a delimited identifier, it can contain any special characters acceptable to access method services. The password is used to access the integrated catalog facility catalog. The password that protects the catalog must be established by the installation of access method services. If this clause is not specified, no password is used by DB2 to access the integrated catalog facility catalog.

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.

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 (DFSMSdfp), which does the actual work. See Section 2 (Volume 1) of *Administration Guide* for more information about creating DB2 storage groups.

Example

Create storage group, DSN8G510, of volumes ABC005 and DEF008. DSNCAT is the integrated catalog facility catalog name, and OSESAME is the VSAM password.

```
CREATE STOGROUP DSN8G510
  VOLUMES (ABC005,DEF008)
  VCAT DSNCAT
  PASSWORD OSESAME;
```

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. However, if the bind option DYNAMICRULES(BIND) applies, the statement cannot be dynamically prepared.

Authorization

None required.

Syntax

```
►► CREATE SYNONYM synonym FOR authorization-name. table-name | view-name ◄◄
```

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 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 474 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 synonym and alias: even if the alias were dropped and redefined, the synonym would still be in effect and would name the original table or view.

Example

Define DEPT as a synonym for the table DSN8510.DEPT.

```
CREATE SYNONYM DEPT  
FOR DSN8510.DEPT;
```

This example will not work if the current SQLID is DSN8510.

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. However, if the bind option DYNAMICRULES(BIND) applies, the statement cannot be dynamically prepared.

Authorization

The privilege set 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

If IN, LIKE or FOREIGN KEY is specified, additional privileges may be required, as explained in the description of those clauses.

Additional authorization may be required when implicitly creating a table space. See the description of the IN parameter for details.

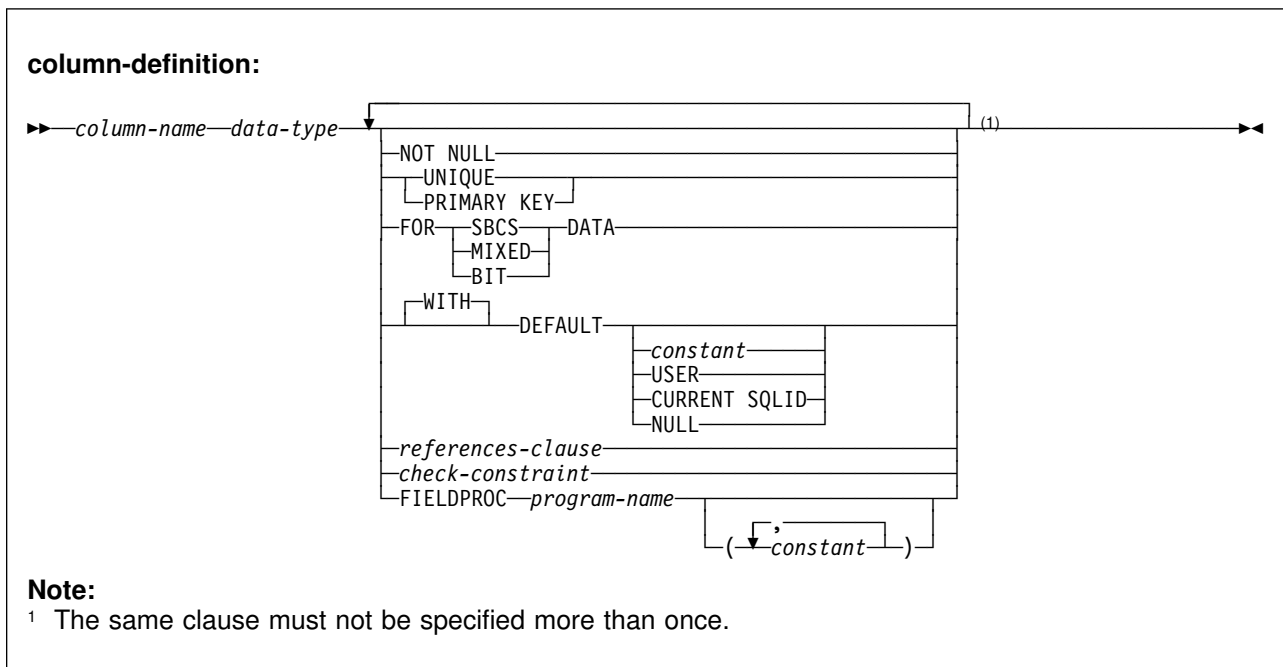
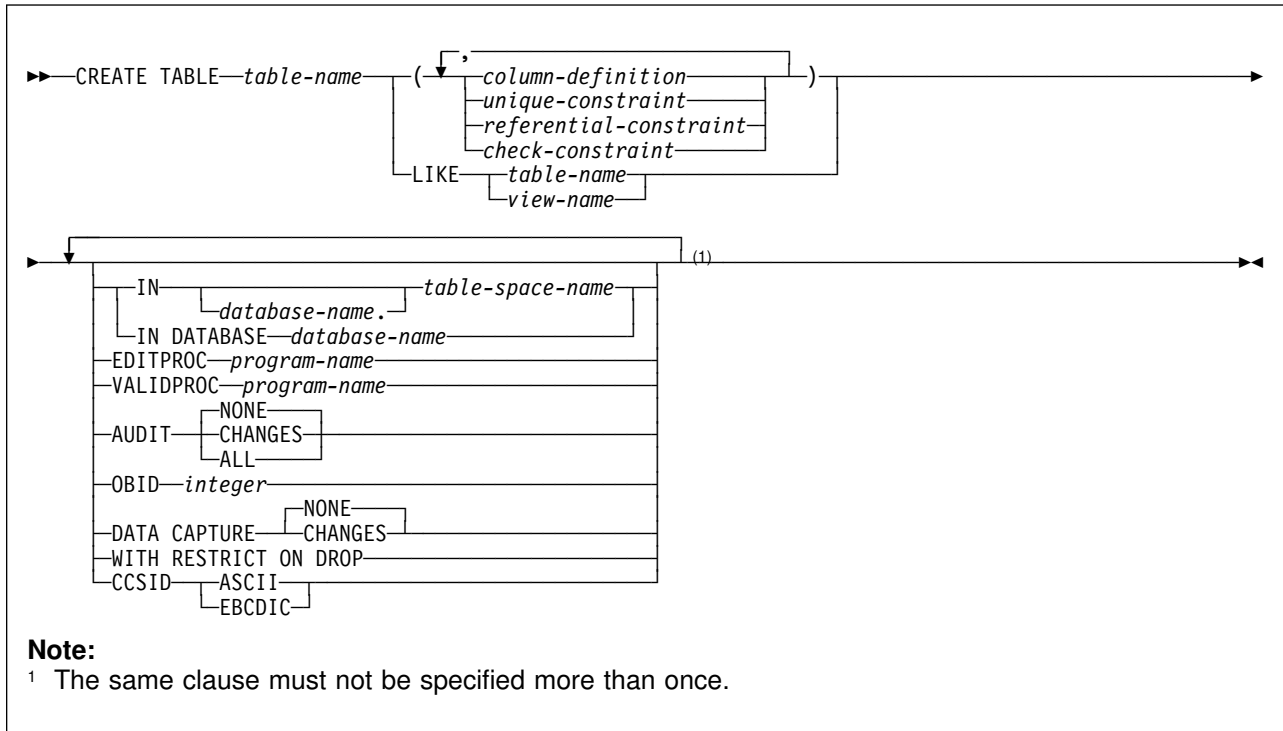
Privilege Set: If the statement is embedded in an application program, the privilege set is the privileges held by the authorization ID of the owner of the plan or package. If the specified table name includes a qualifier that is not the same as this authorization ID, the privilege set must include SYSADM 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 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 held by that authorization ID includes all²⁴ privileges needed to create the table.

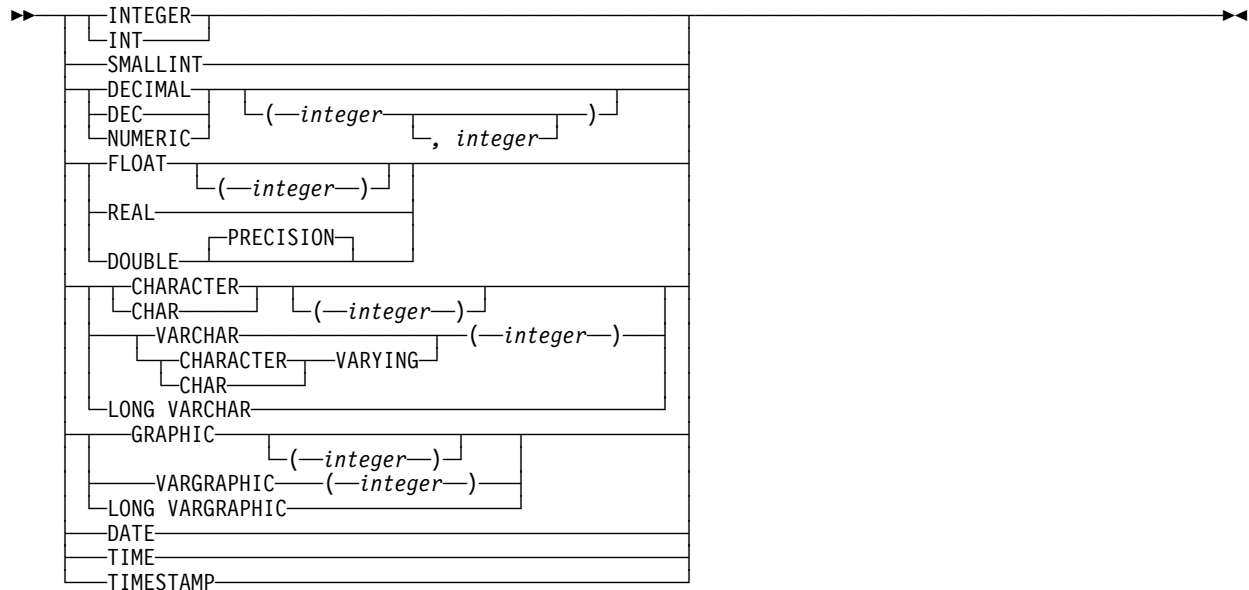
²⁴ Exception: The CREATAB privilege is checked on the SQL authorization ID of the process.

Syntax

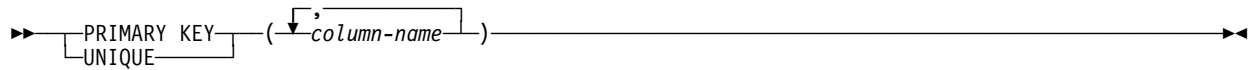


CREATE TABLE

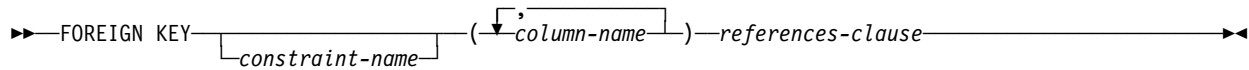
data-type:



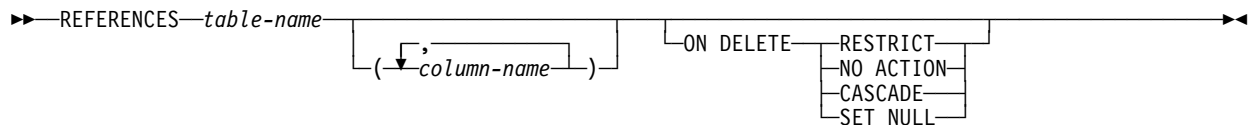
unique-constraint:

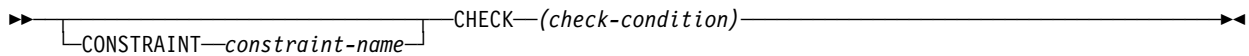


referential-constraint:



references-clause:



check-constraint:**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 re-bound. 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.

data-type

Specifies one of the types in the following list.

INTEGER or INT

For a large integer.

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.

CREATE TABLE

You can also specify:

REAL	For single precision floating-point.
DOUBLE	For double precision floating-point.
DOUBLE PRECISION	For double precision floating-point.
FLOAT	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:

DECIMAL (<i>integer</i>)	For DECIMAL(<i>integer</i> ,0)
DECIMAL	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 21 on page 324 to determine the maximum record size. An integer greater than 255 defines a long string column.

LONG VARCHAR

For a varying-length character string whose maximum length is determined by the amount of space available in a page. For information on how DB2 calculates the maximum length, see “Length of a LONG column” on page 325. If the maximum length is greater than 255, the column is a long string column.

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.

LONG VARGRAPHIC

For a varying-length graphic string whose maximum length is determined by the amount of space available in a page. For information on how DB2 calculates the maximum length, see “Length of a LONG column” on page 325.

If the maximum length is greater than 127, the column is a long string column.

DATE

For a date.

TIME

For a time.

TIMESTAMP

For a timestamp.

NOT NULL

Prevents the column from containing null values.

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 at all if the UNIQUE clause is specified in the column definition.

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 325.)

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.

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.

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 325.)

FOR *subtype* DATA

Specifies a subtype for a character string column; that is, for a column with a data type of CHAR, VARCHAR, or LONG VARCHAR. The FOR DATA clause must not be used with columns of any other data type. *subtype* can be one of the following:

SBCS

Column holds single-byte data.

MIXED

Column holds mixed data.

BIT

Column holds BIT data.

MIXED cannot be specified when the value of field MIXED DATA on installation panel DSNTIPF is NO.

A default subtype applies if the FOR clause is not used in defining a new character string column. 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.

DEFAULT

The default value assigned to the column in the absence of a value specified on INSERT or LOAD. 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

A value other than the one above can be specified in one of the following forms:

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 (SQLID) 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.
- Omission of NOT NULL and DEFAULT is an implicit specification of DEFAULT NULL. If NOT NULL is specified and DEFAULT is omitted, the column does not have a default value.
- DEFAULT and FIELDPROC cannot both be specified.

Table 20 on page 315 summarizes the effect of specifying the various combinations of the NOT NULL and DEFAULT clauses on the CREATE TABLE statement *column-description* clause.

Table 20. Effect of Specifying Combinations of NOT NULL and DEFAULT Clauses

If NOT NULL is:	And DEFAULT is:	The effect is:
Specified	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.
	<i>constant</i>	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.
	<i>constant</i>	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.

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.

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.

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 *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 72.

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.

If you omit FIELDPROC, the column has no field procedure.

_____ End of column-definition _____

_____ unique-constraint _____

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 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 325.)

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 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 325.)

The total number of columns in *all* UNIQUE clauses in the CREATE TABLE statement is limited. If the limit is reached, you can still get the effect of the UNIQUE clause by using a unique index.

End of unique-constraint

referential-constraint

FOREIGN KEY *constraint-name (column-name,...)* **references-clause**

Each specification of the FOREIGN KEY clause defines a referential constraint with the specified name. A name is generated if *constraint-name* is not specified. The generated name is derived from the name of the first column of the foreign key in the same way that the name of an implicitly created table space is derived from the name of a table, except that the scope of uniqueness of *constraint-name* is the table. If specified, *constraint-name* must be different from the names of any referential or check constraints previously specified on the table.

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 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 server²⁵, 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 table²⁵).

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.

²⁵ 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 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 same column must not be identified more than once.

The list of column names must be identical to the list of column names in a unique index (UNIQUE RULE in SYSINDEXES will be R, P, C, or U). The column names must be specified in the **same order** as in the unique index on T2.

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 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 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 Integrity” on page 24.

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 descendents 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 or check 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* is a search condition, with the following restrictions:

- It can refer only to columns of table *table-name*.
- It can be up to 3800 bytes long, not including redundant blanks.
- It must not contain any of the following:
 - Subselects
 - Functions
 - 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.

CREATE TABLE

- 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
- 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.

_____ 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, and the privilege set must implicitly or explicitly include the SELECT privilege on the identified table or view.

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. If the identified table was created in a table space with 4KB pages and the new table is created in a table space with 32KB pages, LONG VARCHAR columns of the identified table will be VARCHAR columns in the new table and LONG VARGRAPHIC columns of the identified table will be VARGRAPHIC columns in the new table. If a column of the identified table has a field procedure, the corresponding column of the new table has the same field procedure and the field description, but the field procedure is not invoked during the execution of the CREATE TABLE statement. If a view is identified, no column has a field procedure because 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.

WITH RESTRICT ON DROP

Indicates that the table cannot be dropped. Also, the database and table space that contain the table cannot be dropped.

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 DSNDB07 (or any other 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, nor be a partitioned table space that already contains a table. 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 *Administration Guide*.

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 *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. 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. The CREATE TABLE statement used to alter the table is audited only if AUDIT CHANGES or AUDIT ALL is specified and the appropriate audit trace class is active. For information about audit trace classes, see Section 3 (Volume 1) of *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 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.

OBID *integer*

Identifies the OBID to be used for this table. An OBID is the identifier for an object's internal descriptor. The OBID keyword is required if the database for the table was defined as ROSHARE READ. When so defined, the current server is using shared read-only data to share the database as a reader. If the table was not defined as ROSHARE READ, the integer must not identify an existing or previously used OBID of the database. For details on shared read-only data, see Appendix F (Volume 2) of *Administration Guide*.

#

If you omit OBID, DB2 generates a value.

The following statement, executed at the DB2 that owns the table's database, would retrieve 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 *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.

CCSID *encoding-scheme*

Specifies the encoding scheme for data stored in the table. If an IN clause is specified with a table space name, the value must agree with the encoding scheme already in use for that table space. All data stored within a table space must use the same encoding scheme.

If an IN clause is not specified with a table space name, the CREATE TABLE statement creates a table space, and the encoding scheme of that table space is the same as the table that you are creating.

If you do not specify a CCSID on a table, the CCSID of the table space is taken. If you do not specify a CCSID on the table space, the CCSID of the table is taken from the database. If you do not specify a CCSID on the database, the CCSID relies on the default encoding scheme at installation time.

ASCII Specifies that the data must be encoded by using the ASCII CCSIDs specified during installation.

EBCDIC Specifies that the data must be encoded by using the EBCDIC CCSIDs specified during installation.

Usually, each encoding scheme requires only a single CCSID. Additional CCSIDs are needed when mixed or graphic data is used.

The option defaults to the encoding scheme of the table space that contains the table, or the encoding scheme of the database if the table space is not explicitly specified.²⁶

If the table space is not explicitly specified and the database is DSNDB04, the option defaults to the value of field DEF ENCODING SCHEME on installation panel DSNTIPF.

The actual coded character set (CCSID) that is used to create the table is obtained from the table space or database. If the CCSID is zero, the CCSID is obtained from the appropriate CODED CHAR SET field on installation panel DSNTIPF.

Notes

Table design: Designing tables is part of the process of database design. For information on design, see Section 2 (Volume 1) of *Administration Guide*.

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.

²⁶ When you use the LIKE clause with the CREATE TABLE statement, the encoding scheme of the table being copied is not used.

CREATE TABLE

While a utility is running: You cannot use CREATE TABLE while a DB2 utility has control of the table space implicitly or explicitly specified by the IN clause.

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 the following table. The page size of the table space is the size of its buffer. This in turn is determined by the BUFFERPOOL clause that was explicitly or implicitly specified when the table space was created.

Table 21. Maximum Record Size, in Bytes

EDITPROC	Page Size = 4KB	Page Size = 32KB
NO	4056	32714
YES	4046	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.

The following table gives the byte counts of columns by data type, for columns that do not allow null values. For a column that allows null values the byte count is one more than shown in the table.

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.
DECIMAL	INTEGER($p/2$)+1, where <i>p</i> is the precision
CHAR(<i>n</i>)	<i>n</i>
VARCHAR(<i>n</i>)	<i>n</i> +2
LONG VARCHAR	See “Byte count of a LONG column” on page 324.
GRAPHIC(<i>n</i>)	2 <i>n</i>
VARGRAPHIC(<i>n</i>)	2 <i>n</i> +2
LONG VARGRAPHIC	See “Byte count of a LONG column” on page 324.
DATE	4
TIME	3
TIMESTAMP	10

Byte count of a LONG 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 column” on page 324.
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.
- (*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 Section 2 (Volume 1) of *Administration Guide*.

Using tables with different encoding schemes: The CCSID clause determines whether the data for a table is encoded in ASCII or EBCDIC. All tables that are referenced in a SQL statement must have the same encoding scheme—the tables must be either all ASCII or all EBCDIC.

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 DSN8510.DEPT in the table space DSN8S51D of the database DSN8D51A. 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 SBSCS.

CREATE TABLE

```
CREATE TABLE DSN8510.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 DSN8D51A.DSN8S51D;
```

Example 2: Create a table named DSN8510.PROJ in an implicitly created table space of the database DSN8D51A. Assign the table a validation procedure named DSN8EAPR.

```
CREATE TABLE DSN8510.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 DSN8D51A
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 on that unique key.

```
CREATE TABLE ACTIVITY
(PROJNO CHAR(6) NOT NULL,
ACTNO SMALLINT NOT NULL,
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 DSN8D51A.DSN8S51D;
```

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. However, if the bind option DYNAMICRULES(BIND) applies, the statement cannot be dynamically prepared.

Authorization

The privilege set defined below must include at least one of the following:

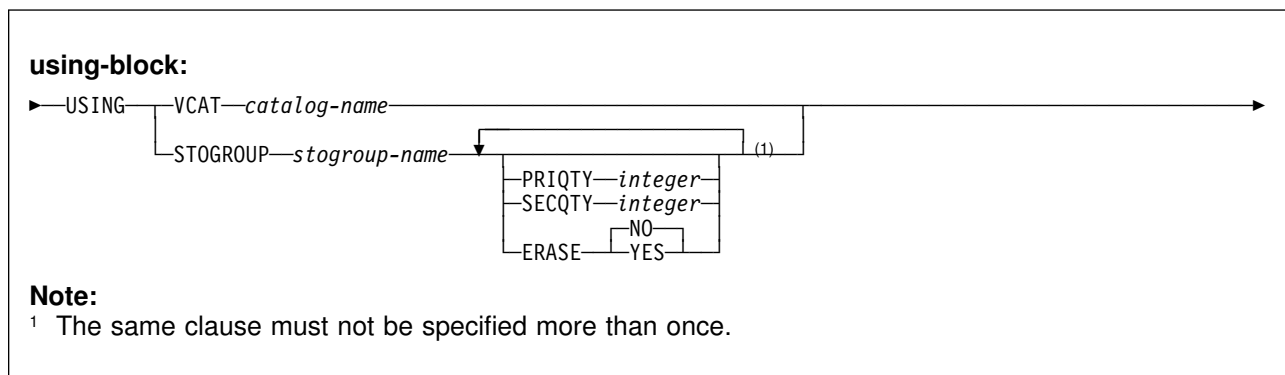
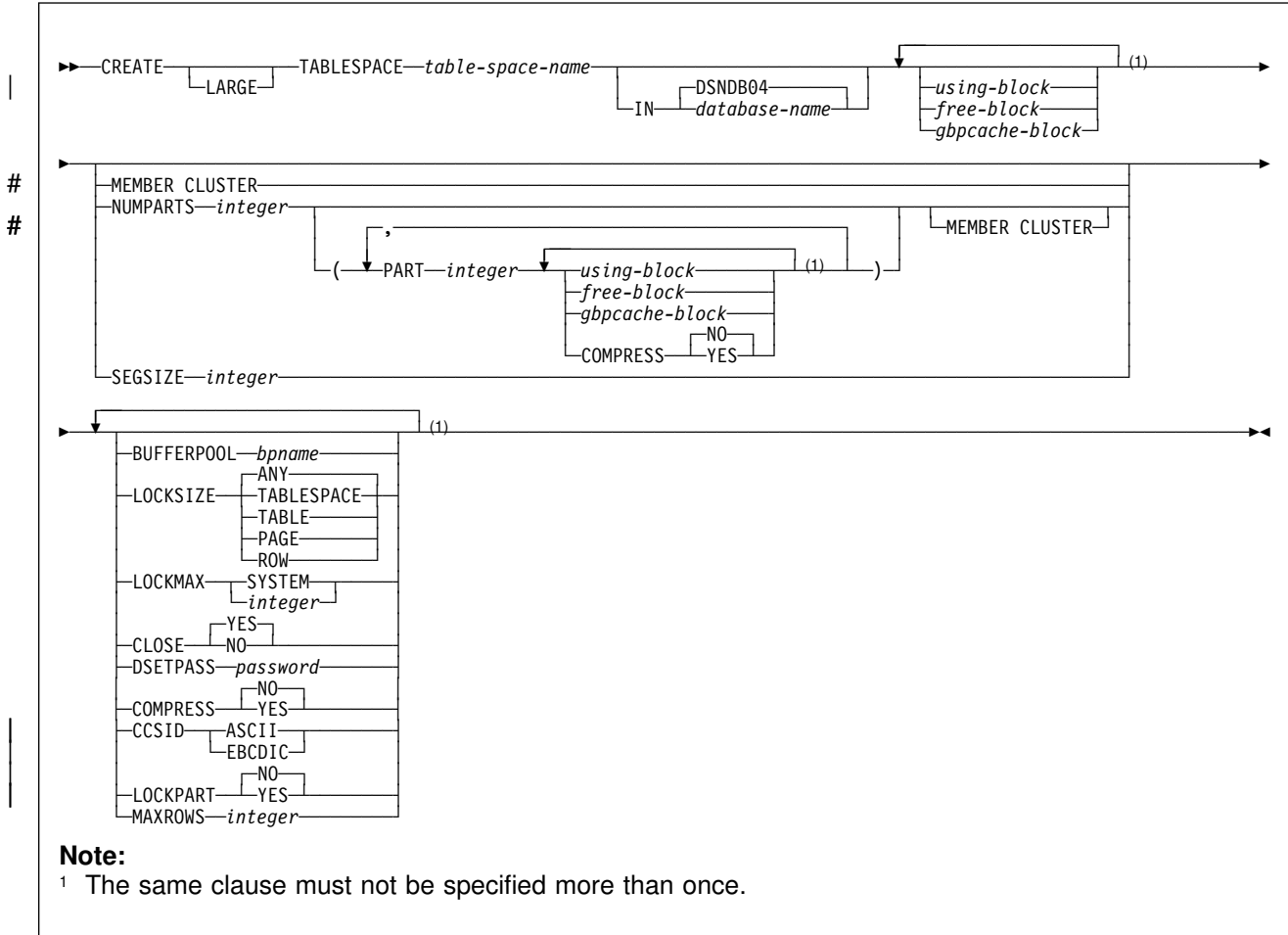
- The CREATETS privilege for the database
- DBADM, DBCTRL, or DBMAINT authority for the database
- SYSADM or SYSCTRL authority

Additional privileges may be required, as explained in the description of the BUFFERPOOL and USING STOGROUP clauses.

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 privileges held by the SQL authorization ID of the process.

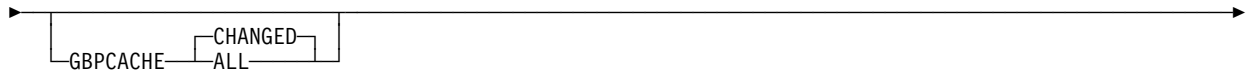
CREATE TABLESPACE

Syntax



free-block:**Note:**

¹ The same clause must not be specified more than once.

gbpcache-block:

Description

LARGE

Identifies a table space as large. A large table space is a partitioned table space that can hold more than 64GB of data, either compressed or uncompressed. A large table space can have a maximum of 254 partitions with a maximum size of 4GB per partition. If **LARGE** is specified, **NUMPARTS** must be specified. If **LARGE** is omitted and the value for **NUMPARTS** is greater than 64, or **LARGE** is specified and the value for **NUMPARTS** is less than 65, then the table space will have the attributes of a large 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 or index space that exists at the current server.

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. If **DSNDB07** or any work file database is specified, it must be in the stopped state. 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.

CREATE TABLESPACE

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 Section 2 (Volume 1) of *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. However, the same *catalog-name* must be used by the subsystems when shared read-only data is used.

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*, except in these cases:

- If the page size is 4KB, and if *integer* is less than 12 or PRIQTY is omitted, then *n* is 12.
- If the page size is 32KB, and if *integer* is less than 96 or PRIQTY is omitted, then *n* is 96.
- If *integer* is greater than 4194304, then *n* is 4194304.

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. 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. If the value of PRIQTY+118 x SECQTY is 2 gigabytes or greater, DB2 automatically defines more data sets when they are needed. DB2 uses a maximum size of 2 gigabytes for a data set. Regardless of the value in PRIQTY, when a data set reaches 2 gigabytes, DB2 creates a new one. This leaves space unused if PRIQTY is greater than 2 gigabytes. Therefore, specify PRIQTY to be less than or equal to 2 gigabytes to minimize unused space.

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* except in these cases:

- If the page size is 4KB and SECQTY and PRIQTY are omitted, then *n* is 12
- If the page size is 32KB and SECQTY and PRIQTY are omitted, then *n* is 96
- If the page size is 4KB and *integer* is greater than 131068, then *n* is 131068
- If the page size is 32KB and *integer* is greater than 131040, then *n* is 131040

If *integer* is 0, no data set can be extended. If you specify PRIQTY and do not also specify SECQTY, the default for SECQTY is either 10% of PRIQTY or 3 times the page size of the table space, whichever is larger. However, if this value exceeds 131068, the default is 131068.

#

DB2 specifies the secondary 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

CREATE TABLESPACE

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 Section 2 (Volume 1) of *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. However, the same *catalog-name* must be used by the subsystems when shared read-only data is used.

DB2 assumes one and only one data set for each partition.

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 use this keyword for a table space in DSNDB07 or in any work file 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 use this keyword for a table space in DSNDB07 or in any work file 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 _____

_____ gpbcache-block _____

GBPCACHE

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 this option, but it is ignored.

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.

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.

In a data sharing environment, hiperpools are not used for indexes or partitions that are defined with GBPCACHE ALL.

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.
3. The default value CHANGED.

End of gbpcache-block

#

MEMBER CLUSTER

Specifies that data inserted by the INSERT statement is not clustered by the 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 use this keyword for a table space in DSNDB07 or in any work file database.

NUMPARTS *integer*

Indicates that the table space will be partitioned. Do not use this keyword with database DSNDB07 or any work file database.

integer is the number of partitions, and can range from 1 to 254 inclusive. If you specify a value greater than 64, a large table space is created, even if you did not specify the LARGE keyword. If you specified the LARGE keyword, you must specify NUMPARTS.

The maximum partition size for a large table space is 4 gigabytes; otherwise, the maximum partition size depends on the number of partitions specified. The following table summarizes the values for the maximum partition size:

²⁷ 1 gigabyte is 1 073 741 824 bytes.

If LARGE is:	And Numparts is:	The maximum size in gigabytes is: ²⁷
Omitted	1 to 16	4
	17 to 32	2
	33 to 64	1
	65 to 254	4
Specified	1 to 254	4

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.

If you omit Numparts, the table space is not partitioned and initially occupies one data set.

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 page buffer pools, the page size is 4KB. Otherwise, the page size is 32 kilobytes. 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 BUFFERPOOL clause is not specified, the default buffer pool of the database is used.

See "Naming Conventions" on page 48 for more details about *bpname*. See Chapter 2 of *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. You must not use this clause for a table space in DSNDB07 or in any work file 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. However, when the number of page locks acquired for the table space exceeds the maximum number of locks allowed for a table space (an installation parameter), the page 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 not segmented, the next higher level is the table space. ANY is the default.

TABLESPACE

Specifies table space locks.

CREATE TABLESPACE

TABLE

Specifies table locks. TABLE can be specified only for a segmented table space.

PAGE

Specifies page locks.

ROW

Specifies row locks.

If you specify ROW, all indexes defined on tables in the table space must be type 2 indexes. If you specify LOCKSIZE ROW for a table space, you cannot create a type 1 index on any of its tables. If you attempt to alter a table space to LOCKSIZE ROW, the statement fails if a type 1 index exists on any of its tables.

LOCKMAX

Specifies the maximum number of page or row locks an application process can hold simultaneously in the table space. If a program requests more than that number, locks are escalated. The page or row locks are released and the intent lock on the table space or segmented table is promoted to S or X mode.

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 or row 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
PAGE, TABLESPACE, TABLE, or ROW	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.

YES

Eligible for closing before CLOSE NO data sets. This is the default.

NO

Eligible for closing after all eligible CLOSE YES data sets are closed.

DSETPASS *password*

Specifies a master level password sent to access method services when the data sets of the table space are used by DB2. *password* is a short identifier. If delimited, *password* can contain any characters acceptable to access method services. If DSETPASS is omitted, a password is not passed.

If you use a storage group, *password* is the password that protects the data sets as well as the password that is passed to VSAM when the data sets are used by DB2. If you do not use a storage group, you define the password that protects the data sets through access method services.

If the table space occupies more than one data set, all its data sets that are password protected must have the same password.

The password does not apply to the data sets managed by Storage Management Subsystem. To protect data sets defined to SMS, use RACF or a similar external security system.

COMPRESS

Specifies whether data compression applies to the rows of the table space or partition.

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 Section 2 (Volume 1) of *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. If the SEGSIZE clause is not provided, the table space is not segmented. *integer* must be a multiple of 4 such that $4 \leq integer \leq 64$.

A segmented table space cannot be partitioned and cannot be created in database DSNDB07 or in any work file database.

CCSID *encoding-scheme*

Specifies the encoding scheme for tables stored in the table space.

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 ASCII CCSIDs specified during installation are used.

|
|

#

CREATE TABLESPACE

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 EBCDIC CCSIDs specified during installation
are used.

Usually, each encoding scheme requires only a single CCSID. Additional
CCSIDs are needed when mixed or graphic data is used.

If you do not specify CCSID, 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.

All data stored within a table space must use the same encoding scheme.

LOCKPART

Indicates whether selective partition locking (SPL) is to be used when locking a
partitioned table space. If you specify LOCKPART YES and all conditions
required for SPL are met, only the partitions accessed will be locked. If you
specify LOCKPART YES and all conditions required for SPL are not met, every
partition of the table space is locked. LOCKPART YES is not allowed with
LOCKSIZE TABLESPACE.

If you specify LOCKPART NO, 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, 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.

You cannot specify the MAXROWS value for a table space in a work file
database.

Notes

If neither NUMPARTS nor SEGSIZE are specified, the table space that is created is
a simple table space. See Section 2 (Volume 1) of *Administration Guide* 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 (a database defined with the WORKFILE
clause in a data-sharing environment):

- They can be created only when the database is explicitly stopped by the STOP
DATABASE command without the SPACENAM option.
- 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.
- The following clauses are not allowed:

NUMPARTS

```

SEGSIZE
LOCKSIZE
FREEPAGE
PCTFREE
GBPCACHE

```

Converting a table space to a large table space: You can redefine an existing partitioned table space as a large table space by taking 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 a large table space. Also redefine the partitioned 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 DSN8S51D in database DSN8D51A. Let DB2 define the data sets, using storage group DSN8G510. 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. The VSAM password for the data sets is OSESAME.

```

CREATE TABLESPACE DSN8S51D
  IN DSN8D51A
  USING STOGROUP DSN8G510
  PRIQTY 52
  SECQTY 20
  ERASE NO
  LOCKSIZE PAGE
  BUFFERPOOL BP1
  CLOSE YES
  DSETPASS OSESAME;

```

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 SALESX in database &DAPP 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 DSN8G510. 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

CREATE TABLESPACE

CLOSE YES data sets to close, the buffer manager may close the CLOSE NO data sets when the DSMAX is reached.

```
CREATE TABLESPACE SALESX
  IN DSN8D51A
  USING STOGROUP DSN8G510
  PRIQTY 4000
  SECQTY 130
  ERASE NO
  NUMPARTS 82
  (PART 80
    COMPRESS YES,
  PART 81
    COMPRESS YES,
  PART 82
    ERASE YES
    COMPRESS YES)
  LOCKSIZE PAGE
  BUFFERPOOL BP1
  CLOSE NO;
```

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. However, if the bind option DYNAMICRULES(BIND) applies, the statement cannot be dynamically prepared.

Authorization

For every table or view identified in the *subselect*, the privilege set 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)

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 342.

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 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 subselect. (It could refer instead, for example, to catalog tables or views thereof.) Otherwise, the owner of the view must be the owner of the plan or package.

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 subselect.

- If the privilege set lacks SYSADM or SYSCTRL authority, 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:

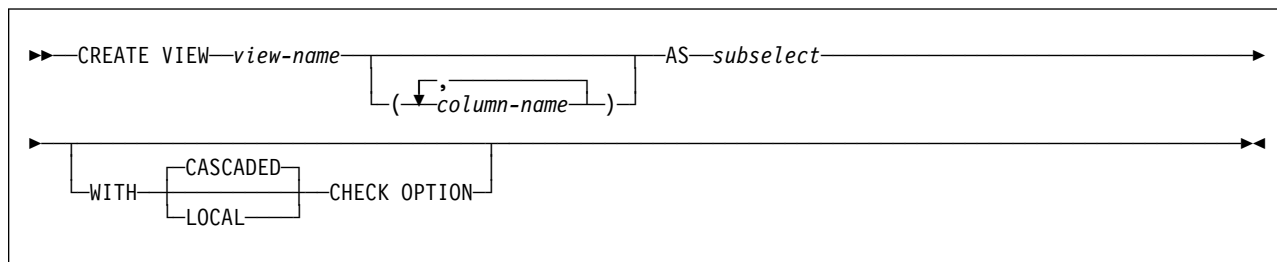
- If the SQL authorization ID of the process has SYSADM authority, the owner of the view can be any authorization ID. If that authorization ID has SYSCTRL but not SYSADM authority, the following is true: the owner of the view can be any authorization ID, provided the view does not refer to user tables or views in the first FROM clause of its defining subselect. (It could refer instead, for example, to catalog tables or views thereof.) Otherwise, the owner of the view must be one of the authorization IDs of the process.

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 subselect.

CREATE VIEW

- If the SQL authorization ID of the process lacks SYSADM or SYSCTRL authority, only the authorization IDs of the process can own the view. In this case, the privilege set is the privileges 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 re-bound. 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 always acquires the SELECT privilege on the view and the authority to drop the view. The SELECT privilege is grantable only if the owner has the grantable SELECT privilege on every table or view identified in the first FROM clause of the SELECT statement of the view. The owner must acquire these grantable privileges before the creation of the view.

The owner can also acquire the INSERT, UPDATE, and DELETE privileges on a view. For this to be possible, the view must not be "read only," in which case a single table or view is identified in the first FROM clause of the subselect. If the owner has one of the above three privileges on this table or view, the owner acquires that privilege on the new view. The privilege is grantable only if the privilege from which it is derived is also grantable. The owner must acquire this privilege before the creation of the view.

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 subselect. 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 subselect.

You must specify a list of column names if the result table of the subselect 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 subselect

Defines the view. At any time, the view consists of the rows that would result if the subselect were executed.

subselect must not refer to host variables or include parameter markers (question marks). For an explanation of *subselect*, see “subselect” on page 170.

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 subselect 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 the view is read-only, its search condition includes a subquery, or the search condition of an underlying view includes a subquery. A check option must not be specified if the *subselect* refers to a 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, *CASCADED* 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*.

CASCADED

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 CASCADED 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. See “Example 2” on page 346 for an example of this situation.

Table 22 illustrates the effect of using the default check option, CASCADED. The information in Table 22 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 CASCADED CHECK OPTION
- CREATE VIEW V3 AS SELECT COL1 FROM V2 WHERE COL1 < 100

Table 22. Examples Using Default Check Option, CASCADED

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 is 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 CASCADED and LOCAL is shown best by example. Consider the following updatable views, where x and y represent either LOCAL or CASCADED:

- 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 23 on page 345 shows the views in which search conditions are checked during an INSERT or UPDATE operation:

Table 23. Views in Which Search Conditions are Checked during INSERT and UPDATE Operations

View used in INSERT or UPDATE Operation	x = LOCAL y = LOCAL	x = CASCADED y = CASCADED	x = LOCAL y = CASCADED	x = CASCADED 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

When a process with appropriate authority creates a view for another user that does not have authorization for the underlying table or view, select authorization 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:

- The first FROM clause identifies more than one table or view
- The first SELECT clause specifies the keyword DISTINCT
- The outer subselect contains a GROUP BY clause
- The outer subselect contains a HAVING clause
- The first SELECT clause contains a column function
- It contains a subquery such that the base object of the outer subselect, 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.

A view cannot map to more than 15 base table instances.

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.

Examples

Example 1: Create the view DSN8510.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 DSN8510.VPROJRE1
  (PROJNO,PROJNAME,PROJDEP,RESPEMP,
   FIRSTNME,MIDINIT,LASTNAME)
AS SELECT ALL
  PROJNO,PROJNAME,DEPTNO,EMPNO,
  FIRSTNME,MIDINIT,LASTNAME
FROM DSN8510.PROJ, DSN8510.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: When a view that is defined WITH LOCAL CHECK OPTION is defined on a view that was defined without a check option. You can update or insert rows that do not conform to the definition of the view. Consider the following views:

```
CREATE VIEW UNDER AS SELECT * FROM DSN8510.EMP
                        WHERE SALARY < 35000;
```

```
CREATE VIEW OVER AS SELECT * FROM UNDER
                        WHERE SALARY > 30000 WITH LOCAL CHECK OPTION;
```

The following UPDATE statement that uses OVER is successful because the updated rows only need to conform to the definition of OVER (SALARY > 30000):

```
UPDATE OVER SET SALARY = SALARY + 5000;
```

However, not all of the rows that you can retrieve through view OVER (over 35,000 rows) are accessible using view UNDER. For example, issuing:

```
SELECT * FROM UNDER
```

returns no rows because no rows conform to the definition of UNDER (SALARY < 35000).

With the CASCADED CHECK OPTION, this situation cannot occur. If OVER had been defined with the WITH CASCADED CHECK OPTION, the UPDATE statement would have failed because the updated rows would not conform to the conjunction of the search conditions OVER and UNDER (SALARY > 3000 and SALARY < 35000).

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:

- If the bind option DYNAMICRULES(RUN) applies, the privilege set is the union of the privilege sets held by each authorization ID of the process. If the bind option DYNAMICRULES(BIND) applies, the privilege set is the privileges held by the authorization ID of the owner of the plan or package.
- 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 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.

DECLARE CURSOR

Syntax

```
DECLARE cursor-name CURSOR (1) FOR select-statement | statement-name  
    WITH HOLD  
    WITH RETURN
```

Note:

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.

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 state.
- 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 Section 3 of *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 188 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 433.

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.

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 more than one table or view
- The first SELECT clause specifies the keyword DISTINCT
- The outer subselect contains a GROUP BY clause
- The outer subselect contains a HAVING clause
- The first SELECT clause contains a column function
- It contains a subquery such that the base object of the outer subselect, and of the subquery, is the same table
- The first FROM clause identifies a read-only view
- The first FROM clause identifies a catalog table with no updateable columns
- The first FROM clause contains a nested table expression
- A UNION or UNION ALL operator is present
- An ORDER BY clause is present
- A FOR FETCH ONLY or a FOR READ ONLY clause is present
- A FOR UPDATE OF clause is not specified and the isolation level at which the statement is executed is UR

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.

DECLARE CURSOR

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 OF 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.
- 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.

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 DSN8510.DEPT. The query itself appears in the DECLARE CURSOR statement.

```
EXEC SQL DECLARE C1 CURSOR FOR
SELECT DEPTNO, DEPTNAME, MGRNO
FROM DSN8510.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 DSN8510.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 DSN8510.EMP
  FOR UPDATE OF WORKDEPT, PHONENO, JOB, EDLEVEL, SALARY;
```

Example 4: In stored procedure SP1, declare C4 as the cursor for a query of the table DSN8510.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 DSN8510.PROJ
  WHERE DEPTNO = 'A01';
```

DECLARE STATEMENT

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.

This statement cannot be included in a REXX application program.

Authorization

None required.

Syntax

```
▶▶ DECLARE 'statement-name' STATEMENT ▶▶
```

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 DSN8510.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 Section 3 of *Application Programming and SQL Guide* and Chapter 2 of *Command Reference*.)

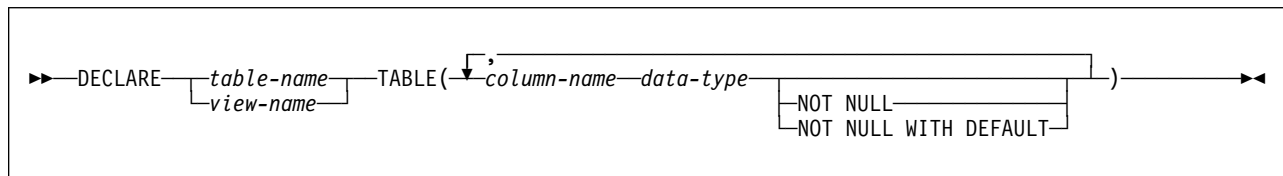
Invocation

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 same name is used in a CREATE TABLE statement in your application program, the description of the table in the CREATE 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.

data-type

Is one of the types in the following list:

INTEGER or **INT**

For a large integer.

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:

REAL	For single precision floating-point
DOUBLE	For double precision floating-point
DOUBLE PRECISION	For double precision floating-point
FLOAT	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:

DECIMAL (<i>integer</i>)	For DECIMAL(<i>integer</i> ,0)
DECIMAL	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*) or **CHAR VARYING**(*integer*) or **CHARACTER VARYING**(*integer*) For a varying-length character string of maximum length *integer*, which can range from 1 to 32704.

LONG VARCHAR For a varying-length character string whose maximum length is determined by DB2.

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 string of double-byte characters, of maximum length *integer*, which can range from 1 to 16352.

LONG VARGRAPHIC For a varying-length string of double-byte characters whose maximum length is determined by DB2.

DATE For a date.

TIME For a time.

TIMESTAMP For a timestamp.

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.

#

DECLARE TABLE

Notes

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.

Example

Declare the sample employee table, DSN8510.EMP.

```
EXEC SQL DECLARE DSN8510.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) );
```

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. 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, 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 containing 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

Searched DELETE and SQL standard rules: 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:

- 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

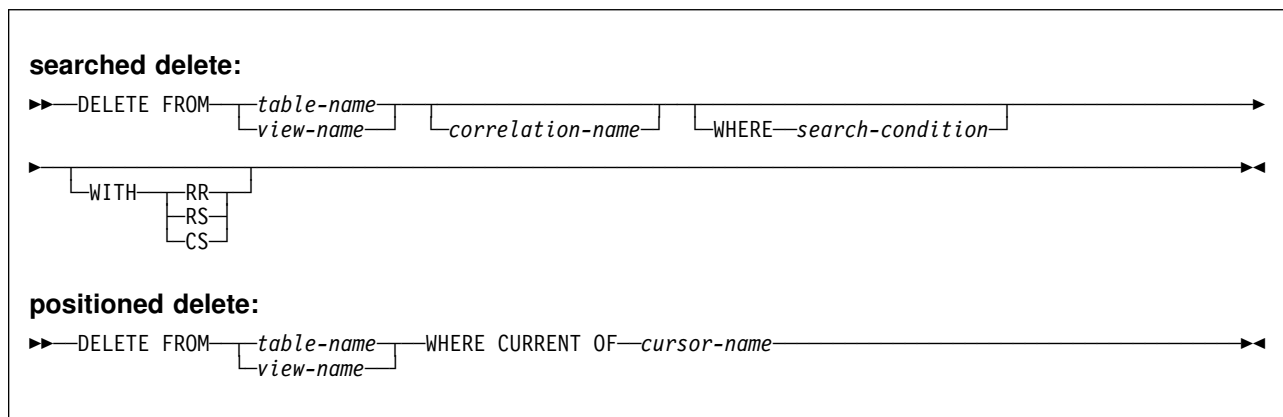
DELETE

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 341.

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 169.

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 and the bind option DYNAMICRULES(RUN) applies, the privilege set is the union of the privilege sets held by each authorization ID of the process. If the statement is dynamically prepared and the bind option DYNAMICRULES(BIND) applies, the privilege set is the privileges held by the authorization ID of the owner of the plan or package.

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:

- 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 341.)

In an IMS or CICS application, the DB2 subsystem containing 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* to qualify references to columns of the table or view. (For an explanation of correlation names, see “Correlation Names” on page 84.)

WHERE

Specifies the rows to be deleted. You can omit the clause, give a search condition or name a cursor. For a temporary table or a view of a 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 3. Language Elements” on page 43. 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 and T2 must not be the same table.
- T1 must not be a dependent of T2 in a relationship with a delete rule of CASCADE or SET NULL.
- 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 “Notes” on page 349. 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 “DECLARE CURSOR” on page 347.) If the cursor is ambiguous and the plan or package was bound with CURRENTDATA(NO), it is possible that DB2 will return an error (SQLCODE -510) 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 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.

DELETE

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.

Notes

Delete rules: 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.

If an error occurs during the execution of any delete operation, no rows are deleted. 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.

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.

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.

DELETE FROM T without a WHERE clause deletes all rows of T. If T is contained in a segmented table space and is not a parent table, this deletion will be performed without accessing T. In this case, database procedures are not invoked and SQLERRD(3) is set to -1.

Except as noted in the above paragraph, 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.

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).

Examples

Assume that the statements in these examples are embedded in PL/I programs.

Example 1: From the table DSN8510.EMP delete the row on which the cursor C1 is currently positioned.

```
EXEC SQL DELETE FROM DSN8510.EMP WHERE CURRENT OF C1;
```

Example 2: From the table DSN8510.EMP, delete all rows for departments E11 and D21.

```
EXEC SQL DELETE FROM DSN8510.EMP
WHERE WORKDEPT = 'E11' OR WORKDEPT = 'D21';
```

DESCRIBE

DESCRIBE

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 433 and “DESCRIBE PROCEDURE” on page 372.

Invocation

This statement can only be embedded in an application program. It is an executable statement that cannot be dynamically prepared.

Authorization

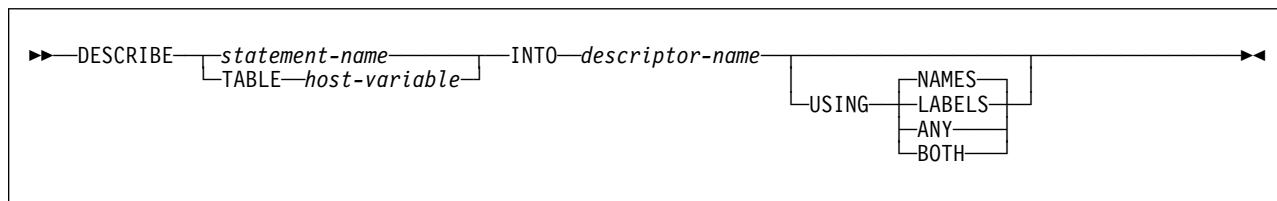
None required if the statement is used for a prepared statement. When it is used instead for a table or view, the privileges held by the authorization ID that owns the plan or package must include at least one of the following (for DESCRIBE TABLE, authorization checking is done only against the plan owner if there is a plan):

- 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 containing the object (tables only)
- SYSADM or SYSCTRL authority

For an RRSF application that does not have a plan and in which the requester and the server are DB2 for OS/390 systems, authorization to execute the package is performed against the primary or secondary authorization ID of the process.

See “PREPARE” on page 433 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. 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 mark is the escape character regardless of the value of the string delimiter option. The reference to the host variable must be preceded by a colon and an indicator variable must not be specified.

INTO *descriptor-name*

Identifies an SQL descriptor area (SQLDA), which is described in Appendix C, “SQLCA and SQLDA” on page 513. See “Identifying an SQLDA in C” on page 526 for how to represent *descriptor-name* in C.

If DESCRIBE is applied to:

- A table or view, the information returned in the SQLDA describes the columns in the table or view
- A prepared statement and that statement is a query, the information returned in the SQLDA describes the columns of the result table
- A prepared statement and that statement is not a query, the information returned in the SQLDA is the fact that the statement is not a query.

#

For languages other than REXX: Before the statement is executed, the SQLDA must be allocated. An SQLDA consists of the fields SQLDAID, SQLDABC, SQLN, and SQLD followed by zero or more occurrences of an SQLVAR structure. For REXX, an SQLDA consists of the field SQLD followed by zero or more occurrences of a set of SQLVAR variables. To obtain column descriptions, the number of occurrences of SQLVAR must be at least as great as the number of columns in the table being described. If the DESCRIBE statement contains USING BOTH, the number of occurrences of SQLVAR must be at least twice the number of columns in the table. For descriptions of methods that determine the required number of occurrences, see “Allocating the SQLDA” on page 365. Before the DESCRIBE statement is executed, the following field in the SQLDA must be set:

Field	Description
SQLN	Indicates the number of SQLVAR occurrences in the SQLDA. The value is not changed by DB2. For performance reasons, you might want to reset this value after the DESCRIBE statement is executed. For details, see “Preparing the SQLDA for data retrieval” on page 366.

#

#

#

#

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*.

Except for SQLN, all the other SQLDA fields are either set by DB2 or ignored, as indicated in the field descriptions below. 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 513.

Field	Description
SQLDAID	Set to 'SQLDA' by DB2.
	A REXX SQLDA does not contain this field.

#

DESCRIBE

	SQLDABC	Length of the SQLDA, in bytes. DB2 sets this value to $SQLN \times 44 + 16$. For performance reasons, you might want to reset this value after the DESCRIBE statement is executed. For details, see “Preparing the SQLDA for data retrieval” on page 366.
#		A REXX SQLDA does not contain this field.
	SQLD	Is set to one of the following values: <ul style="list-style-type: none">• Zero if DESCRIBE is applied to a statement that is not a query• The number of columns in the object being described if USING BOTH is not specified• Twice the number of columns in the object being described if USING BOTH is specified
		If the returned value of SQLD is greater than the value of SQLN, the SQLDA was not large enough for the information requested, and no information is returned in the occurrences of SQLVAR.
	SQLVAR	Assume that the SQLDA is large enough for the requested information, and that N is the number of columns in the object being described. Values are assigned to the first N occurrences of SQLVAR, such that the first occurrence describes the first column in the object, the second occurrence describes the second column, and so on. If USING BOTH appears in the DESCRIBE statement, the second N occurrences of SQLVAR are also assigned values, such that the (N+1)st occurrence describes the first column, the (N+2)nd occurrence describes the second column, and so on. When USING BOTH is used, the first N occurrences of SQLVAR contain column names (where they exist) in the SQLNAME fields, and the second N occurrences contain column labels (where they exist) in the SQLNAME fields. In the second N occurrences, only the SQLNAME fields receive a value. Whether or not USING BOTH occurs, values are assigned as shown below in the first N occurrences of SQLVAR.
#		For REXX, the SQLVAR is a set of stem variables that begin with <i>stem.n</i> , instead of a structure, but the way in which DB2 assigns values to the SQLVAR variables is the same as for other languages. That is, the <i>stem.1</i> variables describe the first column in the result table, the <i>stem.2</i> variables describe the second column in the result table, and so on. If USING BOTH is specified, the <i>stem.n+1</i> variables also describe the first column in the result table, the <i>stem.n+2</i> variables also describe the second column in the result table, and so on.
#		
#		
#		
#		
#		
#		
#		
	SQLTYPE	A code showing the data type of the column and whether or not it can contain null values. For information about the SQLTYPE codes returned following the execution of a DESCRIBE statement, see Table 37 on page 523.
	SQLLEN	A length value depending on the data type of the result columns. For the possible values of SQLLEN, see Table 37 on page 523.
#		In a REXX SQLDA, for DECIMAL or NUMERIC columns, DB2

#		sets the SQLPRECISION and SQLSCALE fields instead of the
#		SQLLEN field.
	SQLDATA	CCSID of a string column, as shown in Table 38 on page 524.
#		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). A string of length 0 if the column does not have a name or label. If the prepared statement includes UNION or UNION ALL, SQLNAME contains the name or label, if any, of the corresponding column of the first operand of UNION. If the described column is the result column of a union, the name in SQLNAME is not necessarily a name that can be used in an ORDER BY clause of the prepared statement.

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. If the prepared statement includes UNION or UNION ALL, SQLNAME contains the name or label, if any, of the corresponding column of the first operand of the union.

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 occurrences of SQLVAR per column are needed to accommodate the additional information. To specify this expansion of the SQLVAR array, set SQLN to $2 \times N$, where N is the number of columns in the object being described. The first N occurrences of SQLVAR for each of the columns contain the column names. The second n occurrences contain the column labels. If the SQLDA is used in a later FETCH statement, set SQLN to N before executing that FETCH statement.

Notes

Information about a prepared statement can also be obtained by using the INTO clause of the PREPARE statement.

Allocating the SQLDA: Among the possible ways to allocate the SQLDA are the three described below. Here, we assume only one SQLVAR for each column in a select list; that is, we assume that USING BOTH does not appear in any DESCRIBE statement:

DESCRIBE

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 the maximum number of columns allowed in a result table. Having done the allocation, the application can use this 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. The value returned for SQLD is equal to the required number of occurrences of SQLVAR.
2. Use the returned value of SQLD to allocate an SQLDA with enough occurrences of SQLVAR. Then 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 DESCRIBE again. For the new SQLDA, use the value of SQLD returned from the first execution of DESCRIBE for the number of occurrences of SQLVAR.

This technique is a compromise between the first two techniques. Its effectiveness depends on a good choice of size for the original SQLDA.

Preparing the SQLDA for data retrieval: 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 meanings of those fields in that context, see Appendix C, "SQLCA and SQLDA" on page 513.

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. You can determine N from the value of SQLD that DB2 returns when the DESCRIBE statement is executed.

Supporting extended dynamic SQL in a distributed environment: In a distributed environment, where DB2 for OS/390 is the server and the requestor supports extended dynamic SQL, a DESCRIBE request that is executed against an SQL statement in the extended dynamic package appears as a DESCRIBE against a static SQL statement in the DB2 package. This request will generate an error unless the DB2 administrator has set the DB2 subsystem parameter DESCSTAT to YES and the package has been rebound. For more information, see Section 3 of *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.

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 365.

```
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

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 be embedded in an application program only. It is an executable statement that cannot be dynamically prepared.

Authorization

None required.

Syntax

```
▶▶DESCRIBE CURSOR { cursor-name | host-variable } INTO descriptor-name ▶▶
```

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 may be up to 19 bytes).
- It must be preceded by a colon and 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 Section 6 of *Application Programming and SQL Guide*.

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

The information that is retrieved by the DESCRIBE CURSOR statement includes
columns names when the statement that generates the result set is dynamic, or the
statement is static and the value of the DB2 subsystem parameter DESCSTAT was
YES when the package or stored procedure was bound.

For the statement to execute successfully, the application program must be connected to the site at which the stored procedure was executed.

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 433 and “DESCRIBE PROCEDURE” on page 372.

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 513. See “Identifying an SQLDA in C” on
page 526 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” on page 362).

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 with these
exceptions:

- # • 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.

For complete information on the contents of the fields, see “SQL Descriptor
Area (SQLDA)” on page 519.

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 519.

Executing DESCRIBE INPUT in a distributed environment: If you use the
 # CONNECT statement to connect to a designated server, check the value of
 # SQLERRP in the SQLCA before executing a DESCRIBE INPUT statement. Ensure
 # that the value is at least DSN05012. If the last byte is less than 2, DESCRIBE
 # INPUT will not execute correctly.

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 366.

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.

```
#      /* 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 365 to allocate the SQLDA.

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 be embedded in an application program only. 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 is a long SQL identifier that must be composed of SBCS characters:

- 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 next two parts identify the stored procedure. A period must separate each of the parts.
- A two-part procedure name is implicitly qualified by the location name of the current server. The name and its implicit qualifier identifies a stored procedure. A period must separate the two parts. The first part identifies the stored procedure at the server. The meaning of the first part depends on the application server.
- 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 identifies the stored procedure at the server. The meaning of the second implicit qualifier depends on the application server (for MVS, the qualifier is SYSPROC). The name and its implicit qualifiers identifies a stored procedure.

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 be preceded by a colon and must not be followed by an indicator variable.

- The value of the host variable is a specification that depends on the application server. Regardless of the application 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 requestor has already invoked using the CALL statement. For the statement to be successful, the application must be connected to the site at which the stored procedure resides.

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.

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 always sets SQLIND to -1.

DESCRIBE PROCEDURE does not return information about the parameters expected by the stored 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.

```
EXEC SQL DESCRIBE PROCEDURE P1 INTO :sqlda1
```

DESCRIBE PROCEDURE

| *Example 2:* Place information about the result sets returned by the stored
| procedure named by :hv1 into the descriptor named by :sqlda2.

| EXEC SQL DESCRIBE PROCEDURE :hv1 INTO :sqlda2

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. However, if the bind option DYNAMICRULES(BIND) applies, the statement cannot be dynamically prepared.

Authorization

To drop a table, table space, or index, the privilege set 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 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 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 defined below must include at least one of the following:

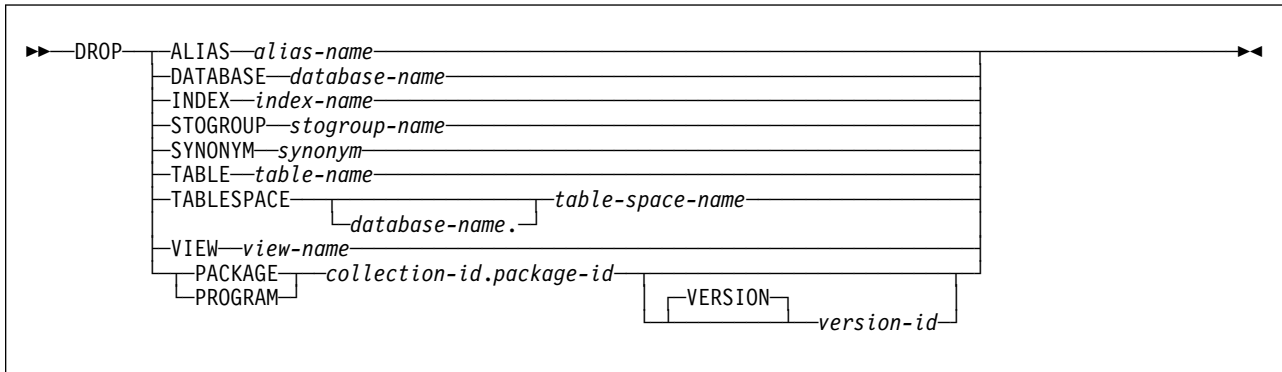
- Ownership of the package
- The BINDAGENT privilege granted from the package owner
- PACKADM authority for the collection or for all collections
- SYSADM or SYSCTRL authority

To drop a synonym, the privilege set defined must include ownership of the synonym.

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 held by each authorization ID of the process.

DROP

Syntax



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.

DATABASE *database-name*

Identifies the database to be dropped. The name must identify a database that exists at the current server. DSNDDB04 or DSNDDB06 must not be specified. If DSNDDB07 or any work file database is specified, it must be in the stopped state and the privilege set must include SYSADM authority.

The database and all of its table spaces, tables, index spaces, and indexes are dropped.

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 partitioned index. (For details on dropping user-defined indexes on catalog tables, see “SQL Statements Allowed on the Catalog” on page 532.) A partitioned index on table T can only be dropped by dropping the table space that contains T.

The index is dropped. 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 a unique index is dropped and that index was used to enforce the uniqueness of a parent key, the definition of the parent table is changed to incomplete. Otherwise, if the index was used to enforce a UNIQUE constraint, the definition of the table is not changed. The table can still be used, but the UNIQUE constraint implied by the index is no longer enforced.

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.

The storage group is dropped. See “Dropping a default storage group” on page 378 for the effect of dropping the default storage group of a database.

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.

The synonym is dropped. This 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 not a catalog table or a table in a partitioned table space. A table in a partitioned table space can only be dropped by dropping the table space.

The table is dropped. 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.

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.

The table space is dropped. Whenever a table space is directly or indirectly dropped, all tables in the table space are also dropped. The name of a dropped table space cannot be reused until a commit operation is performed.

VIEW *view-name*

Identifies the view to be dropped. The name must identify a view that exists at the current server.

The view is dropped. 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.

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 keyword PROGRAM can be used instead of PACKAGE.

The package version is dropped. 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

DROP

- 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 Section 5 of *Application Programming and SQL Guide*.

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 PACKAGE cannot be performed while the package is in use.
- 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.

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: To drop a table space in database DSNDB07 or in any work file database, you must first issue the command STOP DATABASE(*database-name*). Following your DROP, issue -START DATABASE(*database-name*). This process removes the table space you dropped from the pool of table spaces available to DB2.

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 Section 5 (Volume 2) of *Administration Guide* for details.

Dropping a temporary table: To drop a 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 migrated index or table space: Here, “migration” means migrated by the Hierarchical Storage Manager (DFSMSHsm). 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 SYSLGRNX 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 SYSLGRNX records and drop the table space, you cannot reclaim the table space.

Examples

Example 1: Drop table DSN8510.DEPT.

```
DROP TABLE DSN8510.DEPT;
```

Example 2: Drop table space DSN8S51D in database DSN8D51A.

```
DROP TABLESPACE DSN8D51A.DSN8S51D;
```

Example 3: Drop the view DSN8510.VPROJRE1:

```
DROP VIEW DSN8510.VPROJRE1;
```

Example 4: Drop the package DSN8CC0 with the version identifier VERSZZZZ.

The package is in the collection DSN8CC51. Use the version identifier to distinguish the package to be dropped from another package with the same name in the same collection.

```
DROP PACKAGE DSN8CC51.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 DSN8CC51.DSN8CC0 VERSION "1994-07-14-09.56.30.196952";
```

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.

This statement cannot be included in a REXX application program.

Authorization

None required.

Syntax



►►—END DECLARE SECTION—◄◄

The diagram shows the text 'END DECLARE SECTION' centered within a rectangular box. A horizontal line with arrowheads at both ends extends across the width of the box, passing through the text.

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 246.

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.
- Declare sections must not contain statements other than host variable declarations or SQL INCLUDE statements that include host variable declarations.

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

The EXECUTE statement executes a prepared SQL statement.

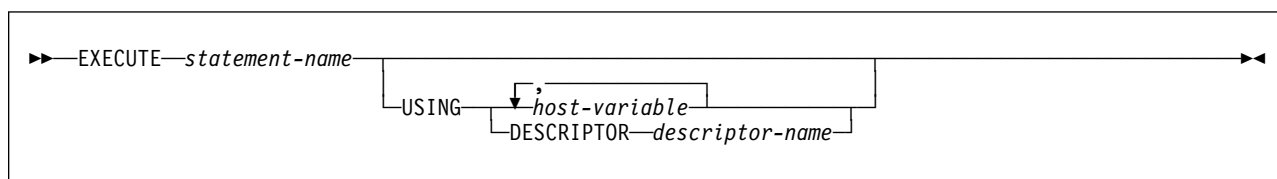
Invocation

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 433 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 433.) 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 383.

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.

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

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. The n th variable described by the SQLDA corresponds to the n th parameter marker in the prepared statement. (For a description of an SQLDA, see Appendix C, “SQLCA and SQLDA” on page 513.)

See “Identifying an SQLDA in C” on page 526 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 65. The attributes of the target variable depend on the role that the parameter marker plays in its SQL statement. The rules for the various roles are shown below. In those rules, P represents the parameter marker in question.

Arithmetic operand: When P is an operand for an infix operator, the other operand cannot also be a parameter marker. The data type, scale, and precision of the target for P are the same as those of the other operand. When P is the operand of a unary minus, the data type of the target is double precision floating-point.

The pattern in a LIKE predicate: With P in this role, the target is a varying-length string. If the first operand in the predicate is a character string column, the target is VARCHAR(n), where n is 10 more than the length attribute of the column, with this exception: If that length attribute is greater than 246, n is 256. If the first operand is a graphic string column, the target is VARGRAPHIC(n), where n is 5 more than the length attribute of the column, with the following exception: If that length attribute is greater than 123, n is 128.

Comparand: In this case, P could be a comparand in a basic predicate, in an IN predicate, or in a BETWEEN predicate. At least one of the comparands in such a predicate must **not** be a parameter marker. One such comparand determines the attributes of the target for P. For a basic predicate, this is simply the other comparand. For a BETWEEN predicate, this is the first (leftmost) comparand that was specified solely as a column name, if one exists. For an IN predicate, and for a

EXECUTE

BETWEEN predicate with no comparand specified solely as a column, this is the first comparand that is not a parameter marker.

If the comparand that determines the attributes has a data type of DATE, TIME, or TIMESTAMP, the target for P is effectively CHAR(255) . Otherwise, the attributes of the target are those of the comparand.

Assignment operand: For this case, P must be the value for a column in an INSERT or UPDATE. The attributes of the target are the same as those of the column, with the following exceptions:

- If the column has the data type DATE, the target is CHAR(*n*), where *n* is the value of field LOCAL DATE LENGTH on installation panel DSNTIP4. If that field is not specified, *n* is 10.
- If the column has the data type TIME, the target is CHAR(*n*), where *n* is the value of field LOCAL TIME LENGTH on installation panel DSNTIP4. If that field is not specified, *n* is 8.
- If the column has the data type TIMESTAMP, the target is CHAR(26).

If the column has the data type DATE, TIME, or TIMESTAMP, trailing blanks are removed from the resulting string before assignment to the target. This is the one exception to the rule that the target is treated like a column.

General rules: 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.

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 DSN8510.DEPT.

```
EXEC SQL PREPARE DEPT_INSERT FROM  
  'INSERT INTO DSN8510.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 character string form of the statement
- Executes the SQL statement
- Destroys the executable form

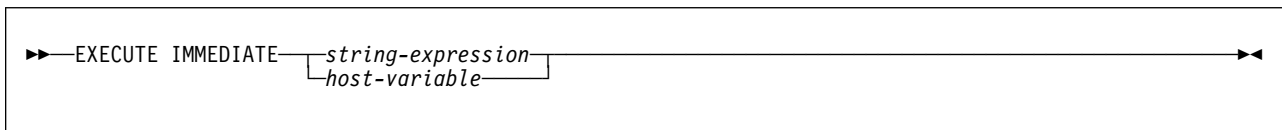
Invocation

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 419 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 character string. 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.

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. 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.

#

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 | LABEL ON |
| COMMENT ON | LOCK TABLE |
| COMMIT | RENAME |
| CREATE | REVOKE |
| DELETE | ROLLBACK |

DROP	SET CURRENT DEGREE
EXPLAIN	SET CURRENT SQLID
GRANT	UPDATE
INSERT	

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 DSN8510.DEPT
WHERE' || PREDS;
```

EXPLAIN

The information about this statement is Product-sensitive Programming Interface and Associated Guidance Information, as defined in “Notices” on page ix.

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*.

Invocation

This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared.

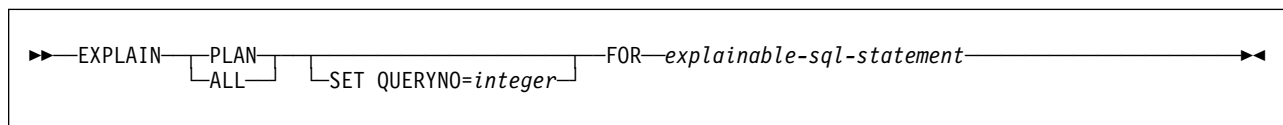
Authorization

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*. Not included are the steps for enforcing referential constraints. The table is described in “Output” on page 389.

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 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 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.

Notes

Output: Output from EXPLAIN is one or more rows of data inserted into the plan table. This table has the name *userid.PLAN_TABLE*, where:

- *userid* is the owner of the plan or package if the EXPLAIN statement is embedded in a plan or package.
- *userid* is the SQL authorization ID of the process if the statement is dynamically prepared.

The table must have been created before the EXPLAIN statement is executed. For information on using the table, see Section 5 (Volume 2) of *Administration Guide*.

EXPLAIN

Creating 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,
   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,
   JOIN_TYPE       CHAR(1)         NOT NULL,
   GROUP_MEMBER    CHAR(8)         NOT NULL,
   IBM_SERVICE_DATA VARCHAR(254)    NOT NULL,
   WHEN_OPTIMIZE   CHAR(1)         NOT NULL,
   QBLOCK_TYPE     CHAR(6)         NOT NULL,
   BIND_TIME       TIMESTAMP       NOT NULL)
  IN database-name.table-space-name;
```

where *database-name.table-space-name* identifies a database and table space you have authorization to use.

Output from BIND or REBIND: DB2 can also add rows to a plan table when a plan or package is bound or rebound. That 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.

The plan table that receives the new rows has the name *userid.PLAN_TABLE*, where *userid* is the owner of the plan or package.

Column descriptions: Table 24 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 24 (Page 1 of 5). Columns in *PLAN_TABLE*. The results of the EXPLAIN statement are stored here.

Column Name	Description
QUERYNO	A number intended to identify the statement being explained. For a row produced by an EXPLAIN statement, you can specify the number in the SET QUERYNO clause; otherwise, DB2 assigns a number based on the line number of the SQL statement in the source program. Values of QUERYNO 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.

²⁸ On each step of a join, DB2 joins the current *composite table* to a *new table*. For the first step of a join, DB2 picks one of the join tables to serve as composite table and another to serve as new table. On later steps, the composite table is the result of all the previous join steps, and the new table is a table whose FROM-table reference has not yet been used. The order in which the tables are used, and the method to be used for each step, are determined by DB2 when the statement containing the join is prepared.

EXPLAIN

Table 24 (Page 2 of 5). Columns in PLAN_TABLE. The results of the EXPLAIN statement are stored here.

Column Name	Description
METHOD	A number (0, 1, 2, 3, or 4) that indicates the join method used for the step ²⁸ : <ul style="list-style-type: none"> 0 First table accessed, continuation of previous table accessed, or not used 1 <i>Nested loop</i> join. For each row of the present composite table, matching rows of a new table are found and joined. 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. 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. 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, temporary table, materialized view, table expression, or an intermediate result table for an outer join that is accessed in this step; blank if METHOD is 3. For an outer join, this column contains the temporary table name of the work file in the form DSNWFQB(<i>qblockno</i>). Merged views show the base table names and correlation names. A materialized view is another query block with its own materialized views, tables, and so on.
TABNO	Values are for IBM use only.
ACCESSTYPE	The method of accessing the new table: <ul style="list-style-type: none"> I By an index (identified in ACCESSCREATOR and ACCESSNAME) I1 By a one-fetch index scan N By an index scan when matching predicate contains IN keyword R By a table space scan M By a multiple index scan; followed by MX, MI, or MU MX By an index scan on the index named in ACCESSNAME MI By an intersection of multiple indexes MU By a union of multiple indexes blank Not applicable to the current row.
MATCHCOLS	For ACCESSTYPE I, I1, N, or MX, the number of index keys used in an index scan; otherwise, 0.
ACCESSCREATOR	For ACCESSTYPE I, I1, N, or MX, the creator of the index; otherwise, blank.
ACCESSNAME	For ACCESSTYPE 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 Section 6 of <i>Application Programming and SQL Guide</i> .
SORTN_UNIQ	Whether a sort is performed on the new table to remove duplicate rows. Y=Yes; N=No.
SORTN_JOIN	Whether a sort is performed on the new table if METHOD is 2 or 4. Y=Yes; N=No.
SORTN_ORDERBY	Whether an ORDER BY clause results in a sort on the new table. Y=Yes; N=No.
SORTN_GROUPBY	Whether a GROUP BY clause results in a sort on the new table. Y=Yes; N=No.
SORTC_UNIQ	Whether a sort is performed on the composite table to remove duplicate rows. Y=Yes; N=No.
SORTC_JOIN	Whether a sort is performed on the composite table if METHOD is 1, 2 or 4. Y=Yes; N=No.

Table 24 (Page 3 of 5). Columns in PLAN_TABLE. The results of the EXPLAIN statement are stored here.

Column Name	Description
SORTC_ORDERBY	Whether an ORDER BY clause or a quantified predicate results in a sort on the composite table. Y=Yes; N=No.
SORTC_GROUPBY	Whether a GROUP BY clause results in a sort on the composite table. Y=Yes; N=No.
TSLOCKMODE	<p>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:</p> <p>IS Intent share lock IX Intent exclusive lock S Share lock U Update lock X Exclusive lock SIX Share with intent exclusive lock N UR isolation; no lock</p> <p>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.0</p> <p>NS For UR isolation, no lock; for CS, RS, or RR, an S lock. NIS For UR isolation, no lock; for CS, RS, or RR, an IS lock. NSS For UR isolation, no lock; for CS or RS, an IS lock; for RR, an S lock. SS For UR, CS, or RS isolation, an IS lock; for RR, an S lock.</p> <p>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.</p>
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 = when data is retrieved; S = when data is sorted; blank = to be decided at execution.
MIXOPSEQ	<p>The sequence number of a step in a multiple index operation.</p> <p>1, 2, ... n For the steps of the multiple index procedure (ACCESSTYPE is MX, MI, or MU.)</p> <p>0 For any other rows (ACCESSTYPE is I, I1, M, N, R, or blank.)</p>
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.
Note: All of the following 9 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, and can be 0 if there is a host variable. The actual number of parallel operations used at execution time could be different. This column contains 0 if there is a host variable.

EXPLAIN

Table 24 (Page 4 of 5). Columns in PLAN_TABLE. The results of the EXPLAIN statement are stored here.

Column Name	Description
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.
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; <ul style="list-style-type: none"> I Query I/O parallelism C Query CP parallelism X 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 an outer join. <ul style="list-style-type: none"> F FULL OUTER JOIN L LEFT OUTER JOIN blank INNER JOIN or no join 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: <ul style="list-style-type: none"> blank At bind time, using a default filter factor for any host variables, parameter markers, or special registers. B At bind time, using a default filter factor for any host variables, parameter markers, or special registers; however the statement will be 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. R 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.

Table 24 (Page 5 of 5). Columns in PLAN_TABLE. The results of the EXPLAIN statement are stored here.

Column Name	Description
QBLOCK_TYPE	For each query block, the type of SQL operation performed. For the outermost query, it identifies the statement type. Possible values: SELECT SELECT INSERT INSERT UPDATE UPDATE DELETE DELETE SELUPD SELECT with FOR UPDATE OF DELCUR DELETE WHERE CURRENT OF CURSOR UPDCUR UPDATE WHERE CURRENT OF CURSOR CORSUB Correlated subquery NCOSUB Non-correlated subquery
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.

Table creation options: A plan table can function with fewer columns than those shown in the foregoing CREATE statement. The options are:

- All the columns up to and including REMARKS
- All the columns up to and including MIXOPSEQ
- All the columns up to and including COLLID
- All the columns shown in the CREATE statement

Only these options can be used. Whichever option you choose, the columns defined must appear in the indicated order. With an ALTER TABLE statement you can add columns to an existing plan table, as long as the modified table satisfies one of the options. For example, you could add the columns PREFETCH, COLUMN_FN_EVAL, and MIXOPSEQ, but not just the column PREFETCH. When adding NOT NULL columns, give them the NOT NULL WITH DEFAULT attribute.

Missing columns are ignored when rows are added to a plan table.

Table migration: You can migrate existing plan tables to later releases or fall back to earlier releases. If you fall back to an earlier release, the extra columns are simply ignored when EXPLAIN is executed. If you migrate to a later release, the missing columns are likewise ignored.

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 DSN8510.EMPPROJACT X, DSN8510.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 with the following query:

EXPLAIN

```
SELECT * FROM PLAN_TABLE WHERE QUERYNO = 13  
ORDER BY QBLOCKNO, PLANNO, 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.

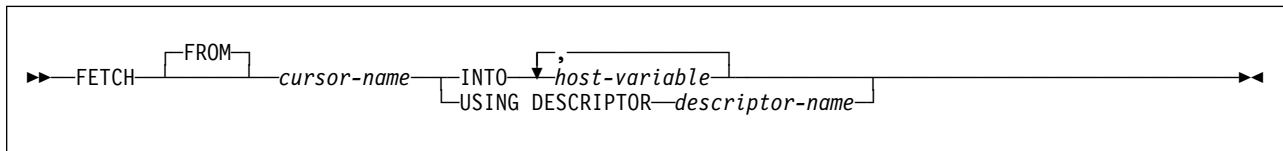
Invocation

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 347 for an explanation of the authorization required to use a cursor.

Syntax



Description

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 349, or an allocated cursor, as explained in “ALLOCATE CURSOR” on page 200 . When the FETCH statement is executed, the cursor must be in the open state.

If the cursor is currently positioned on or after the last row of its result table, the SQLCODE field of the SQLCA is set to +100, SQLSTATE is set to '02000', the cursor is positioned after the last row, and values are not assigned to host variables.

If the cursor is currently positioned before a row, the cursor is positioned on that row and values are assigned to host variables as specified by INTO or USING.

If the cursor is currently positioned on a row other than the last row, the cursor is positioned on the next row and values of that row are assigned to host variables as specified by INTO or USING.

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.

FETCH

USING 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

SQLD must be set to a value greater than or equal to zero and less than or equal to SQLN. Each SQLVAR occurrence describes a host variable or buffer into which a value in the result set is to be assigned. The first value in the result row is assigned to the first host variable or buffer described in the SQLDA; the second value is assigned to the second host variable or buffer described in the SQLDA; and so on. (For a description of an SQLDA, see Appendix C, “SQLCA and SQLDA” on page 513.)

See “Identifying an SQLDA in C” on page 526 for how to represent *descriptor-name* in C.

Notes

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 63. 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 3. Language Elements” on page 43. 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 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.

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.

Cursor positioning: An open cursor has three possible positions:

- Before a row
- On a row
- After the last row

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.

The current row of a cursor cannot be updated or deleted by another application process if it is locked or a temporary copy of a result table was created when the cursor was opened. 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

Example

The FETCH statement fetches the results of the SELECT statement into the application program variables DNUM, DNAME, and MNUM.

```
EXEC SQL DECLARE C1 CURSOR FOR
  SELECT DEPTNO, DEPTNAME, MGRNO FROM DSN8510.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;
```

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
- Package
- Plan
- 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. However, if the bind option DYNAMICRULES(BIND) applies, the statement cannot be dynamically prepared.

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

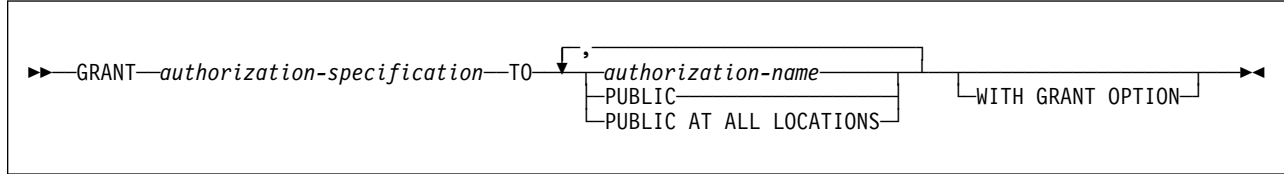
The presence of SYSCTRL authority in the privilege set allows the granting of all authorities except:

- DBADM on databases
- DELETE, INSERT, SELECT, and UPDATE on user tables or views
- EXECUTE on plans or packages
- PACKADM on collections
- SYSADM

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 341 for a description of rules that apply to views.

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 privileges 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 application 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, and REFERENCES.

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 Section 3 (Volume 1) of *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

GRANT

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*.

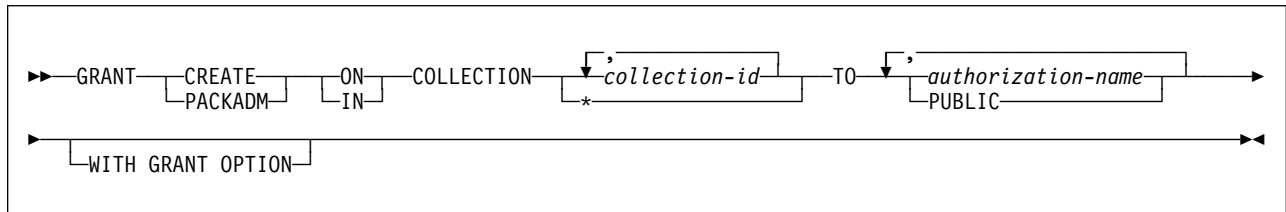
Different grantors can grant the same privilege to a single grantee. The grantee retains that privilege as long as one or more of those grants are recorded in the catalog. Privileges that imply other privileges are also termed *authorities*. Grants are removed from the catalog by executing SQL REVOKE statements.

Whenever a grant is made for a database, package, plan, table or view, or USE privilege for an object that does not exist, an SQL return code is issued and the grant is not made.

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 400 for a description of the TO clause.

Example

Grant the privilege to create new packages in collections QAACLONE and DSN8CC51 to CLARK.

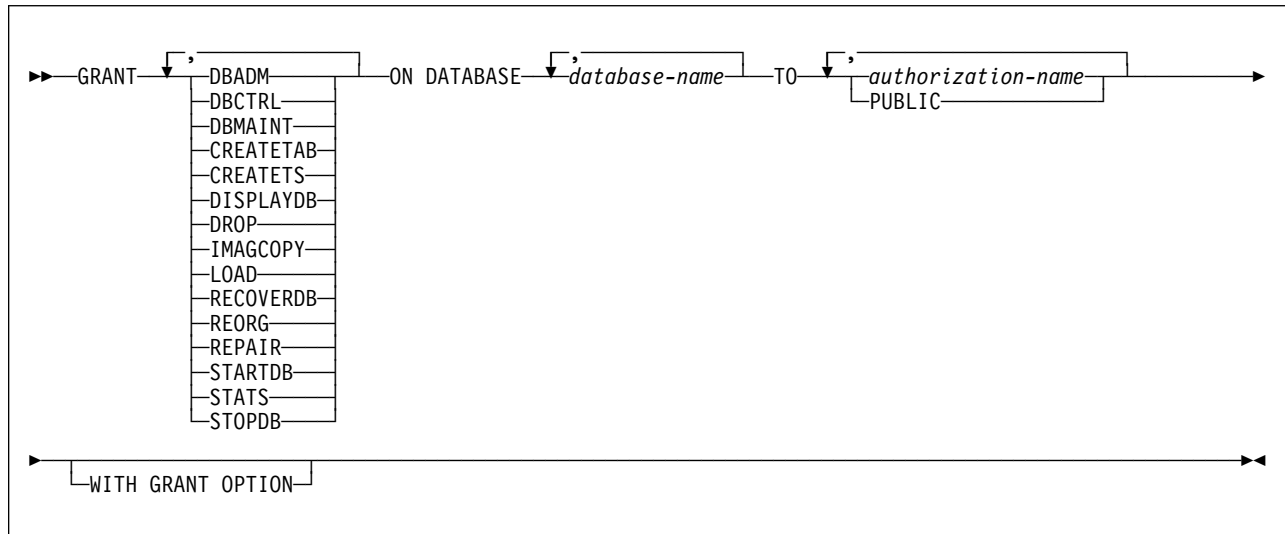
```
GRANT CREATE IN COLLECTION QAACLONE, DSN8CC51 TO CLARK;
```

GRANT (Database Privileges)

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.

CREATETS

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.

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 400 for a description of the TO clause.

Examples

Example 1: Grant drop privileges on database DSN8D51A to user PEREZ.

```
GRANT DROP
  ON DATABASE DSN8D51A
  TO PEREZ;
```

Example 2: Grant repair privileges on database DSN8D51A to all local users.

```
GRANT REPAIR
  ON DATABASE DSN8D51A
  TO PUBLIC;
```

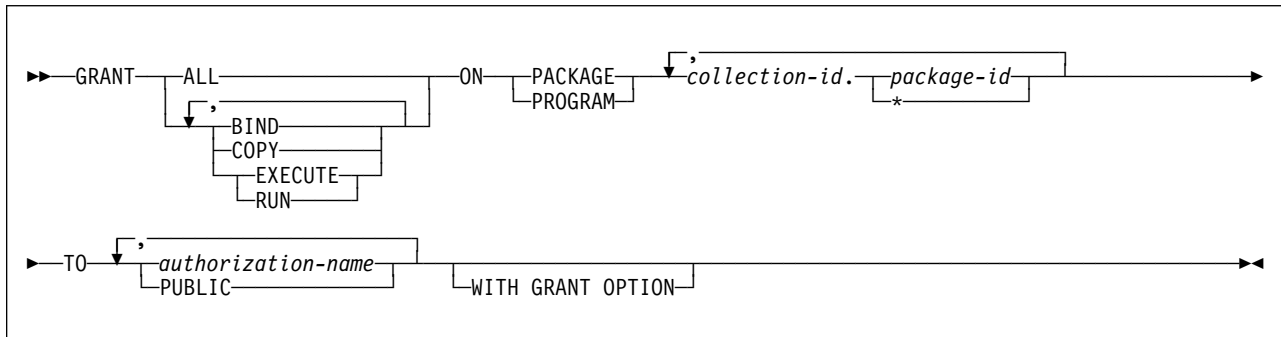
Example 3: Grant authority to create new tables and load tables in database DSN8D51A to users WALKER, PIANKA, and FUJIMOTO, and give them grant privileges.

```
GRANT CREATETAB,LOAD
  ON DATABASE DSN8D51A
  TO WALKER,PIANKA,FUJIMOTO
  WITH GRANT OPTION;
```

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 can also be used to allow a user 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 407.

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 future packages as well as those that currently exist. 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 400 for a description of the TO 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
- Users with the BIND package privilege can also add a new version of an existing package

Examples

Example 1: Grant the privilege to copy all packages in collection DSN8CC51 to LEWIS.

```
GRANT COPY ON PACKAGE DSN8CC51.* 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.

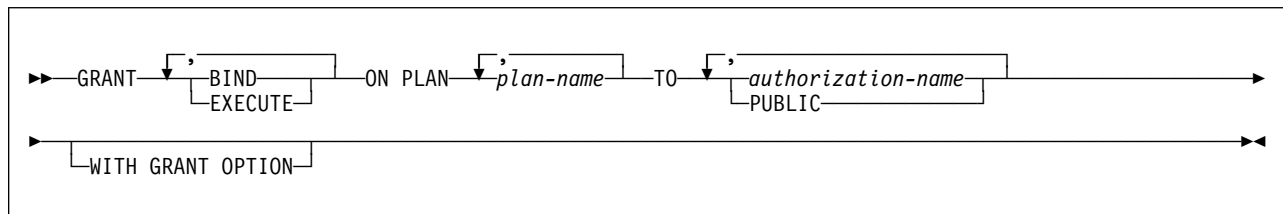
```
GRANT ALL ON PACKAGE CLCT1.PKG1, CLCT2.PKG2 TO JONES;
```

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 400 for a description of the **TO** clause.

Examples

Example 1: Grant the privilege to bind plan **DSN8IP51** to user **JONES**.

```
GRANT BIND ON PLAN DSN8IP51 TO JONES;
```

Example 2: Grant privileges to bind and execute plan **DSN8CP51** to all users at the current server.

```
GRANT BIND,EXECUTE ON PLAN DSN8CP51 TO PUBLIC;
```

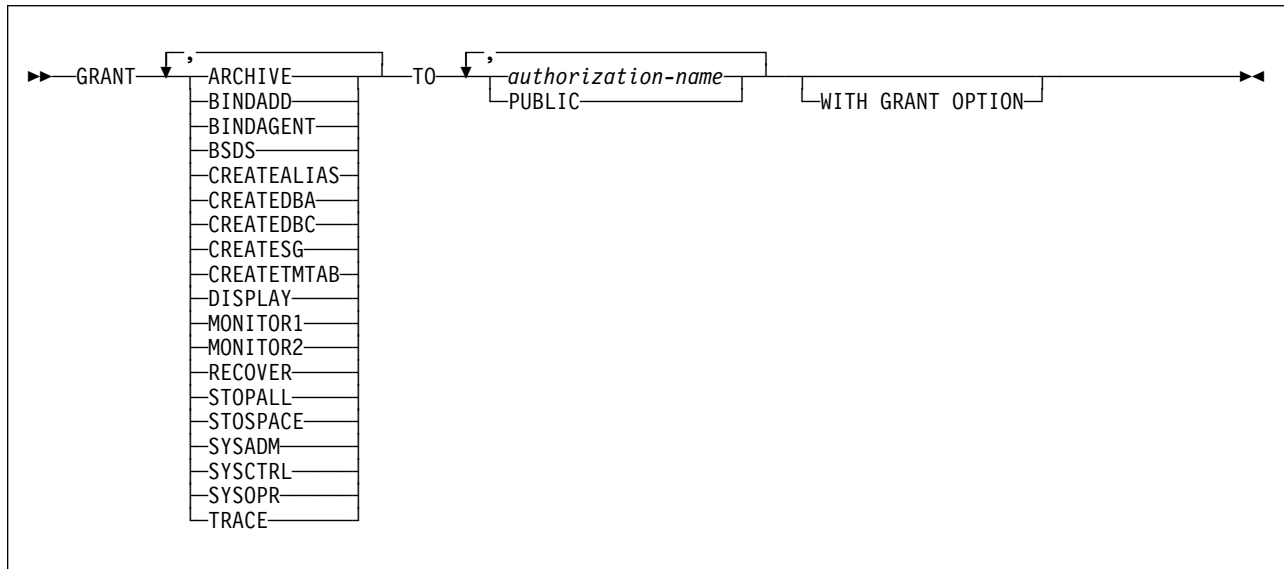
Example 3: Grant the privilege to execute plan **DSN8CP51** to users **ADAMSON** and **BROWN** with grant option.

```
GRANT EXECUTE ON PLAN DSN8CP51 TO ADAMSON,BROWN WITH GRANT OPTION;
```

GRANT (System Privileges)

This form of the GRANT statement grants system privileges.

Syntax



Description

ARCHIVE

Grants the privilege to use the ARCHIVE LOG command.

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.

CREATEDBC

Grants the privilege to issue the CREATE DATABASE statement and acquire DBCTRL authority over those databases.

GRANT (System Privileges)

CREATESG

Grants the privilege to create new storage groups.

CREATETMTAB

Grants the privilege to use the CREATE GLOBAL TEMPORARY TABLE statement.

DISPLAY

Grants the privilege to do the following:

- Use the DISPLAY ARCHIVE command for archive log information
- Use the DISPLAY BUFFERPOOL command for the status of buffer pools
- Use the DISPLAY DATABASE command for the status of all databases
- Use the DISPLAY LOCATION command for statistics about threads with a distributed relationship
- Use the DISPLAY THREAD command for information on active threads within DB2
- Use 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. (Having MONITOR2 privilege also includes having 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 Section 3 (Volume 1) of *Administration Guide*.

SYSCTRL

Grants the system control authority, allowing the holder most of the privileges of a system administrator but excluding privileges to read or change user data. Using WITH GRANT OPTION when granting SYSCTRL is redundant but valid. For more on SYSCTRL authority, see Section 3 (Volume 1) of *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 400 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.

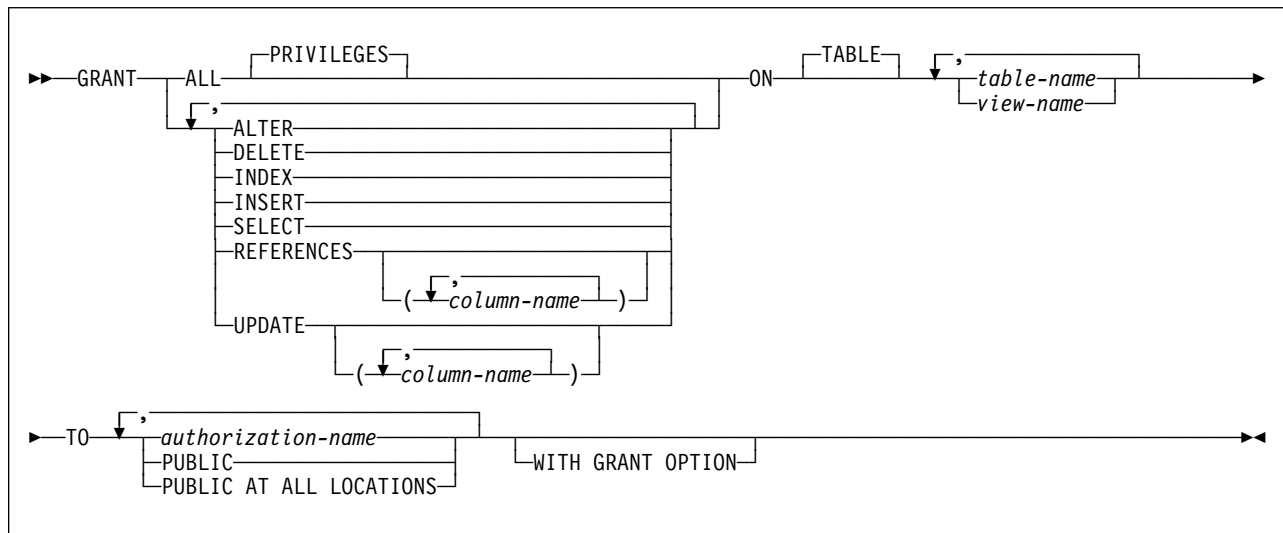
```
GRANT TRACE
  TO PUBLIC;
```

GRANT (Table or View Privileges)

GRANT (Table or View Privileges)

This form of the GRANT statement grants privileges on table 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, or REFERENCES 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 a view.

DELETE

Grants the privilege to use the DELETE statement.

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.

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, and it cannot be granted on 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.

UPDATE

Grants the privilege to use the UPDATE statement.

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.

ON or ON TABLE

Names 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.

If you use GRANT ALL, then for each named table or view, the privilege set (described in “Authorization” in “GRANT” on page 400) must include at least one privilege with the GRANT option.

TO

Refer to “GRANT” on page 400 for a description of the TO 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 temporary table or a view of a 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 temporary table.

Examples

Example 1: Grant SELECT privileges on table DSN8510.EMP to user PULASKI.

```
GRANT SELECT ON DSN8510.EMP TO PULASKI;
```

Example 2: Grant UPDATE privileges on columns EMPNO and WORKDEPT in table DSN8510.EMP to all users at the current server.

```
GRANT UPDATE (EMPNO,WORKDEPT) ON TABLE DSN8510.EMP TO PUBLIC;
```

Example 3: Grant all privileges on table DSN8510.EMP to users KWAN and THOMPSON, with the WITH GRANT OPTION.

```
GRANT ALL ON TABLE DSN8510.EMP TO KWAN,THOMPSON WITH GRANT OPTION;
```

Example 4: Grant the SELECT and UPDATE privileges on the table DSN8510.DEPT to every user in the network.

```
GRANT SELECT, UPDATE ON TABLE DSN8510.DEPT  
TO PUBLIC AT ALL LOCATIONS;
```

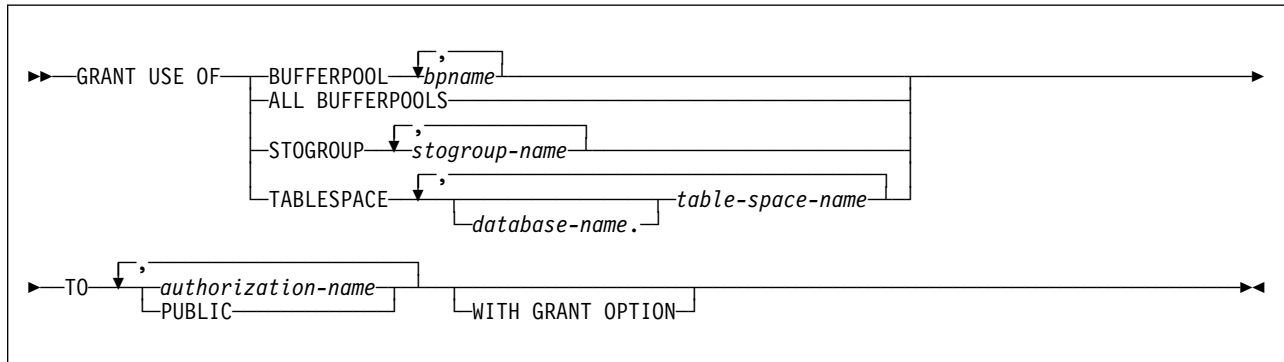
GRANT (Table or View Privileges)

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 529. Controlling access is described in Section 3 (Volume 1) of *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 48 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.

TO

Refer to “GRANT” on page 400 for a description of the TO clause.

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.

GRANT (Use Privileges)

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 DSN8S51D in database DSN8D51A.

```
GRANT USE OF TABLESPACE  
DSN8D51A.DSN8S51D  
TO PUBLIC;
```

INCLUDE

The INCLUDE statement inserts declarations or code into a source program.

Invocation

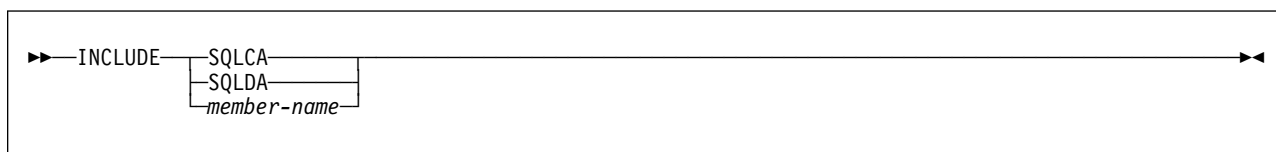
This statement can only be embedded in an application program. It is not an executable statement.

This statement cannot be included in a REXX application program.

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.

For a description of the SQLCA, see “SQL Communication Area (SQLCA)” on page 513.

SQLDA

Indicates that the description of an SQL descriptor area (SQLDA) is to be included. It must not be specified in a FORTRAN or COBOL program. For a description of the SQLDA, see “SQL Descriptor Area (SQLDA)” on page 519.

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.

INCLUDE

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 and/or views.

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 containing 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 341.

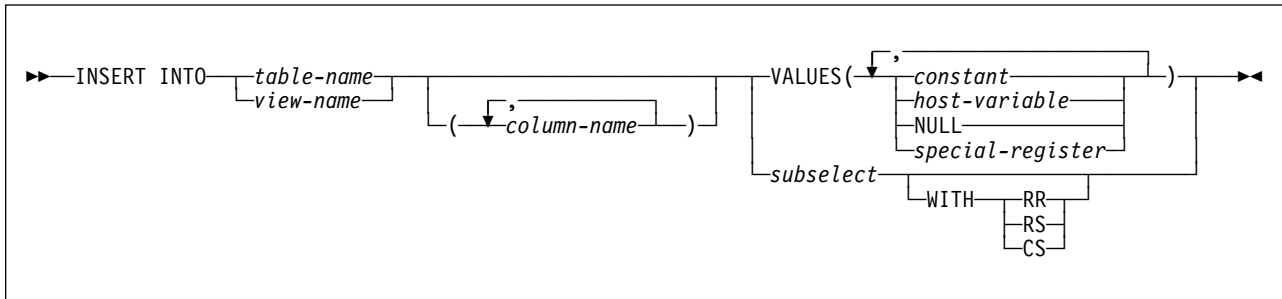
If a subselect is specified, the privilege set must include authority to execute the subselect. For more information about the subselect authorization rules, see “Authorization” on page 169.

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 and the bind option DYNAMICRULES(RUN)

INSERT

applies, the privilege set is the union of the privilege sets held by each authorization ID of the process. If the statement is dynamically prepared and the bind option DYNAMICRULES(BIND) applies, the privilege set is the privileges held by the authorization ID of the owner of the plan or package.

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:

- 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 341.)

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 containing 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 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.

VALUES

Specifies one new row in the form of a list of values. Each host variable in the list 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 the statement, a reference to a host structure is replaced by a reference to each of its variables.

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.

For an explanation of *constant* and *host-variable*, see “Chapter 3. Language Elements” on page 43. For a description of *special-register*, see “Special Registers” on page 78. NULL specifies the null value.

subselect

Specifies a set of new rows in the form of the result table of a subselect. If the result table is empty, SQLCODE is set to +100, and SQLSTATE is set to '02000'.

(For an explanation of subselect, see “Chapter 5. Queries” on page 167.)

The base object of the INSERT, and the base object of the subselect, or any subquery of the subselect, must not be the same table.

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.

If the object table is self-referencing, the subselect must not return more than one row.

WITH

Specifies the isolation level at which the subselect 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.

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.
- *Assignment.* Insert values are assigned to columns in accordance with the assignment rules described in “Chapter 3. Language Elements” on page 43.

INSERT

- *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 341.

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.

Number of rows inserted: After an INSERT statement completes execution, the value of SQLERRD(3) in SQLCA is the number of rows inserted. (For a description of the SQLCA, see “SQL Communication Area (SQLCA)” on page 513.)

Locking: Unless appropriate locks already exist, one or more exclusive locks are acquired at the execution of a successful INSERT statement. Until the locks are released by a commit or rollback operation, an inserted row can only be accessed by the application process that performed the insert and the locks can prevent other application processes from performing operations on the table.

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 *Administration Guide*.

Datetime representation when using datetime registers: As explained under “Datetime special registers” on page 78, 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 table DSN8510.EMP.

```
INSERT INTO DSN8510.EMP
VALUES ('000205', 'MARY', 'T', 'SMITH', 'D11', '2866',
       '1981-08-10', 'ANALYST', 16, 'F', '1956-05-22',
       16345, 500, 2300);
```


Example 2: Populate the temporary table SMITH.TEMPEMPL with data from table DSN8510.EMP.

```
INSERT INTO SMITH.TEMPEMPL
  SELECT *
  FROM DSN8510.EMP;
```

Example 3: Populate the temporary table SMITH.TEMPEMPL with data from department D11 from DSN8510.EMP.

```
INSERT INTO SMITH.TEMPEMPL
  SELECT *
  FROM DSN8510.EMP
  WHERE WORKDEPT='D11';
```

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.

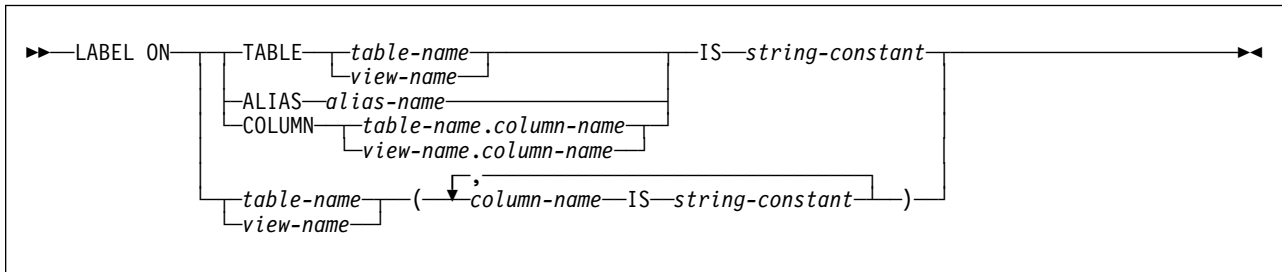
Authorization

The privilege set 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

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 and the bind option DYNAMICRULES(RUN) applies, the privilege set is the union of the privilege sets held by each authorization ID of the process. If the statement is dynamically prepared and the bind option DYNAMICRULES(BIND) applies, the privilege set is the privileges held by the authorization ID of the owner of the plan or package.

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. 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 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,...
```

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 DSN8510.DEPT.

```
LABEL ON COLUMN DSN8510.DEPT.DEPTNO
IS 'DEPARTMENT NUMBER';
```

Example 2: Enter labels on two columns in table DSN8510.DEPT.

```
LABEL ON DSN8510.DEPT
(MGRNO IS 'MANAGER'S EMPLOYEE NUMBER',
ADMNDEPT IS 'ADMINISTERING DEPARTMENT');
```

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.

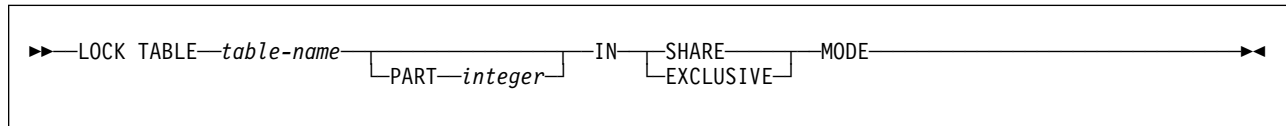
Authorization

The privilege set 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

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 and the bind option DYNAMICRULES(RUN) applies, the privilege set is the union of the privilege sets held by each authorization ID of the process. If the statement is dynamically prepared and the bind option DYNAMICRULES(BIND) applies, the privilege set is the privileges held by the authorization ID of the owner of the plan or package.

Syntax



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, or a catalog table. The lock might or might not apply exclusively to the 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

Requests the acquisition of a lock that prevents other processes from executing anything but read-only operations on the table. 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.

Notes

For more information on using LOCK TABLE (such as the size and duration of locks), and on locking in general, see Section 4 of *Application Programming and SQL Guide* or Section 5 (Volume 2) of *Administration Guide*.

Example

Obtain a lock on the sample table named DSN8510.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 DSN8510.EMP IN EXCLUSIVE MODE;
```

OPEN

The OPEN statement opens a cursor.

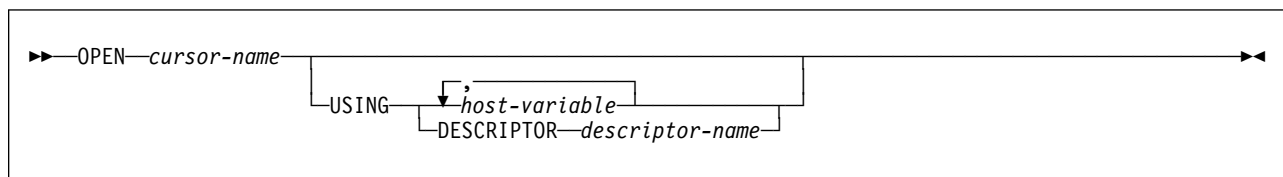
Invocation

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 347 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 349 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 433.) 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 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.

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

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. The *n*th variable described by the SQLDA corresponds to the *n*th parameter marker in the prepared statement. (For a description of an SQLDA, see Appendix C, “SQLCA and SQLDA” on page 513.)

See “Identifying an SQLDA in C” on page 526 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 430.

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. 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 65. The attributes of the target variable depend on the role that the parameter marker plays in its query. The rules for the various roles are shown below. In those rules, P represents the parameter marker in question.

Arithmetic operand: When P is an operand for an infix operator, the other operand cannot also be a parameter marker. The data type, scale, and precision of the target for P are the same as those of the other operand. When P is the operand of a unary minus, the data type of the target is double precision floating-point.

The pattern in a LIKE predicate: With P in this role, the target is a varying-length string. If the first operand in the predicate is a character string column, the target is VARCHAR(*n*), where *n* is 10 more than the length attribute of the column, with this exception: If that length attribute is greater than 246, *n* is 256. If the first operand is a graphic string column, the target is VARGRAPHIC(*n*), where *n* is 5 more than the length attribute of the column, with the following exception: if that length attribute is greater than 123, *n* is 128.

Comparand: In this case, P could be a comparand in a basic predicate, in an IN predicate, or in a BETWEEN predicate. At least one of the comparands in such a predicate must **not** be a parameter marker. One such comparand determines the attributes of the target for P. For a basic predicate, this is simply the other comparand. For a BETWEEN predicate, this is the first (leftmost) comparand that was specified solely as a column name, if one exists. For an IN predicate, and for a BETWEEN predicate with no comparand specified solely as a column, this is the first comparand that is not a parameter marker.

#

If the comparand that determines the attributes has a data type of DATE, TIME, or TIMESTAMP, the target for P is effectively CHAR(255) . Otherwise, the attributes of the target are those of the comparand.

General rules: 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.

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.

OPEN

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 DSN8510.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

The PREPARE statement creates an executable SQL statement from a character string form of the statement. The executable form is called a prepared statement. The character string form is called a statement string.

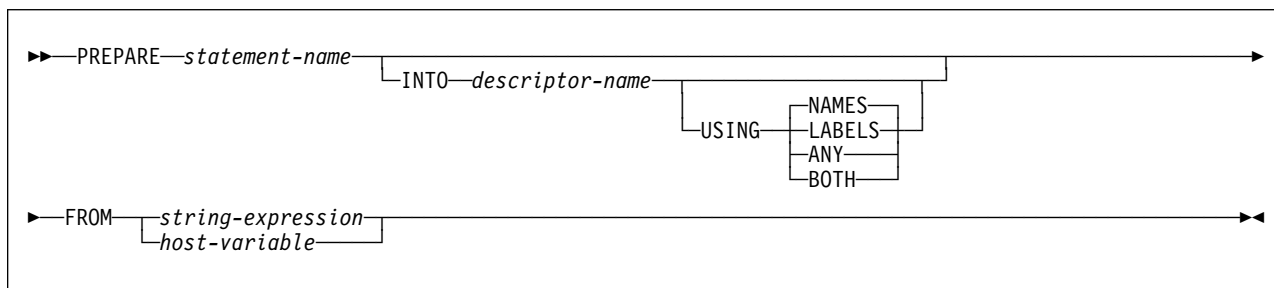
Invocation

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 5. Queries” on page 167 for the authorization rules that apply when a SELECT statement is prepared.

Syntax



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;
```

See “DESCRIBE” on page 362 for an explanation of the information that is placed in the SQLDA.

descriptor-name

Identifies the SQLDA. For languages other than REXX, SQLN must be set to indicate the number of SQLVAR occurrences.

See “Identifying an SQLDA in C” on page 526 for how to represent *descriptor-name* in C.

#

PREPARE

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.

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 that case, two occurrences of SQLVAR per column are needed to accommodate the additional information. To specify this form of the SQLVAR array, set SQLN to $2 \times n$, where n is the number of columns in the result table. The first n occurrences of SQLVAR for each of the columns in the result table contain the column names. The second n occurrences contain the column labels. If the SQLDA is used in a later FETCH statement, set SQLN to n before executing that FETCH statement.

A REXX SQLDA does not include the SQLN field, so you do not need to set SQLN for REXX programs.

#

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 character string. 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.

host-variable

Must identify a host variable that is described in the application program in accordance with the rules for declaring character string variables. 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.

Notes

Rules for statement strings: The statement string must be one of the following SQL statements:

ALTER	LABEL ON
COMMENT ON	LOCK TABLE
COMMIT	RENAME
CREATE	REVOKE
DELETE	ROLLBACK
DROP	SET CURRENT DEGREE
EXPLAIN	SET CURRENT SQLID

GRANT	UPDATE
INSERT	<i>select-statement</i>

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*. Those can be 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 382, “OPEN” on page 428, and Section 6 of *Application Programming and SQL Guide* .

Rules for parameter markers:

- Parameter markers must not be used:
 - In a select list (SELECT ? is invalid)
 - As an operand of the concatenation operator
 - As both operands of a single arithmetic or comparison operator (WHERE ? = ? is invalid)
 - As an operand in a datetime arithmetic expression
 - In a SET statement
 - In the first operand of a LIKE predicate
 - In the first operand of a NULL predicate
 - As the factored expression following the CASE keyword in a *simple-when* clause of a CASE expression
- At least one of the operands of the BETWEEN or IN predicates must not be a parameter marker.
- At least one *result-expression* in a CASE expression must not be a parameter marker.
- An argument of a scalar function cannot be specified solely as a parameter marker. However, if a scalar function is used in other than a SELECT list, and it has an argument that can be specified as an arithmetic expression, a parameter marker can be included in that expression, provided that it is the operand of an arithmetic operator and that the other operand is a number.
- In other than a SELECT list, a parameter marker can be the operand of a unary minus. For example, WHERE C = -?.

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,

PREPARE

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 <i>not</i> 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 KEEP DYNAMIC(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 349 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.

Example

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 DSN8510.DEPT VALUES(?,?,?,?)';
```

(Check for successful execution and read values into host variables)

```
EXEC SQL EXECUTE DEPT_INSERT USING :S1, :S2, :S3, :S4, :S5;
```

RELEASE

The RELEASE statement places one or more connections in the release pending state.

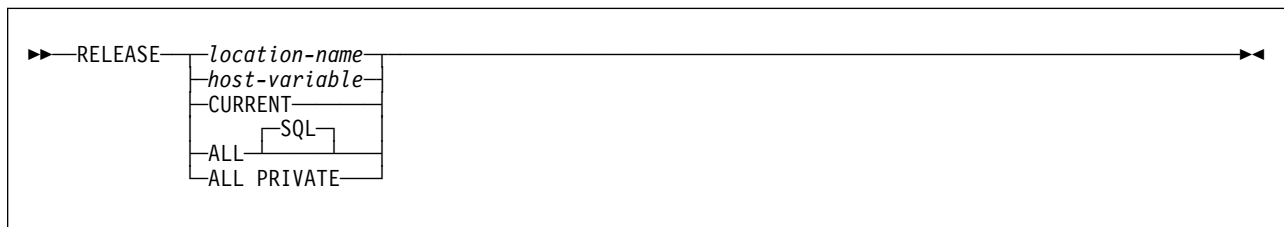
Invocation

This statement can only be embedded in an application program. It is an executable statement that cannot be dynamically prepared. RELEASE cannot be used if the program is executing as a stored procedure.

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 may be up to 17 bytes.)
- It must be preceded by a colon and 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 statement is successful, the identified connection is placed in the release pending state and will therefore be ended during the next commit operation. If the RELEASE 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.

RELEASE

If the RELEASE statement is successful, the identified connection is placed in the release pending state and will therefore be ended during the next commit operation. If the RELEASE 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 state. 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 state. 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.

RELEASE 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 state and one that is going to be reused should not be in the release pending state. 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 state 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.

An application 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 36.

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;
```


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 application 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 application 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;
```

RENAME

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. However, if the bind option DYNAMICRULES(BIND) applies, the statement cannot be dynamically prepared.

Authorization

The privileges set 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 held by each authorization ID of the process.

If the name of the existing table includes a qualifier that is different from the authorization ID of the statement, the privileges held by the authorization ID of the statement must include administrative authority.

Syntax

```
RENAME [TABLE] source-table-name TO target-identifier
```

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 already exists at the current server. The name cannot be a catalog table, a view, a synonym, or an active RLST table. 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.

An error is issued if the source table is currently referenced in any view definitions.

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:

- SYSCHECKS
- SYSCHECKDEP
- SYSCOLAUTH
- SYSCOLDIST
- SYSCOLDISTSTATS
- SYSCOLSTATS
- SYSCOLUMNS
- SYSFIELDS
- SYSFORIGNKEYS
- SYSINDEXES
- SYSPLANDEP
- SYSPACKDEP
- SYSRELS
- SYSSYNONYMS
- SYSTABAUTH
- SYSTABLES
- SYSTABSTATS

Entries in SYSTMTS and SYSPACKSTMTS are not updated.

Invalidation of plans, packages , and dynamic statements: When the RENAME TABLE statement is executed, any plans or packages that refer to that table are invalidated. If any dynamic statements in the statement cache refer to this table, they are invalidated; DB2 must refresh those statements in the cache the next time they are executed.

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.

RENAME

| **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
- Package
- Plan
- 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. However, if the bind option DYNAMICRULES(BIND) applies, the statement cannot be dynamically prepared.

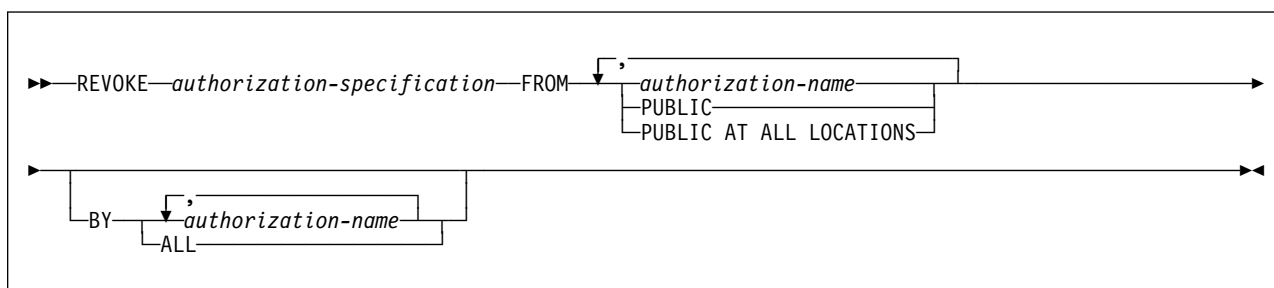
If the authorization mechanism was not activated when the DB2 subsystem was installed, an error condition occurs.

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.

Syntax



REVOKE

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.

Notes

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:

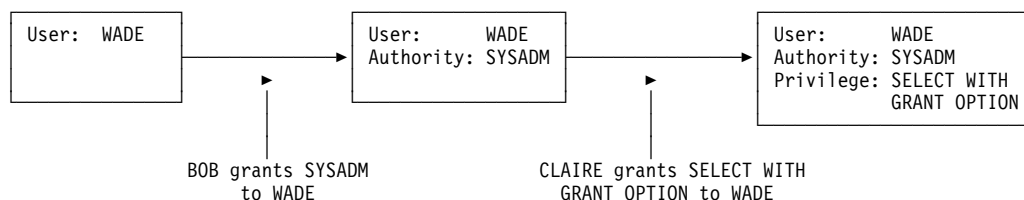
- P and P' are the same privilege.
- U2 granted privilege P' to U3.
- No one granted privilege P to U2 prior to the grant by U1.
- U2 does not have installation SYSADM authority.

The rules also apply to the implicit grants that are made as a result of a CREATE VIEW statement.

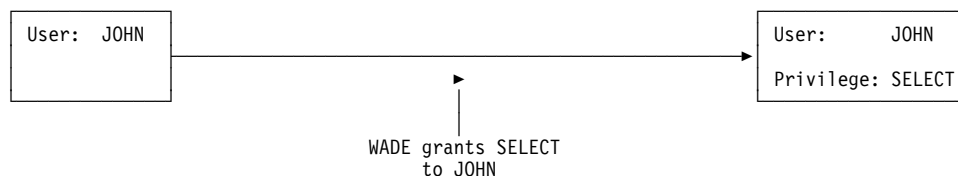
Cascade revoke does not occur if the privilege was granted by a current install SYSADM.

Refer to the diagrams for the following example:

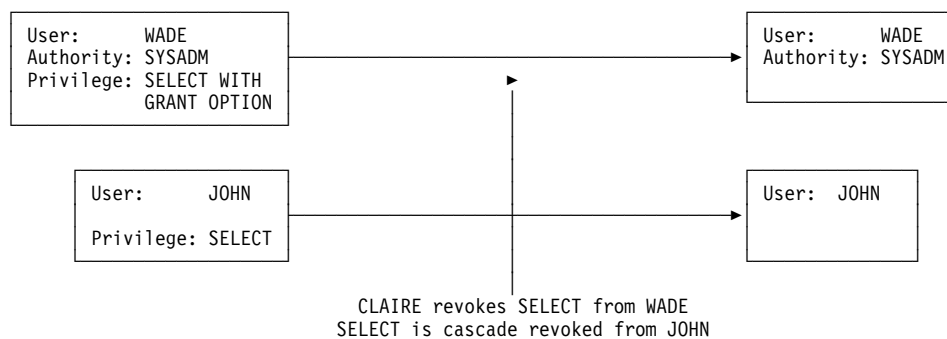
1. Suppose BOB grants SYSADM authority to WADE. Later, CLAIRE grants the SELECT privilege on a table with the WITH GRANT OPTION to WADE.



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 Section 3 (Volume 1) of *Administration Guide*. For another example of cascading revokes, see Section 3 (Volume 1) of *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 341.

REVOKE

Invalidation of plans and packages: A revoke or cascade revoke of any privilege that was exercised to create a plan or package invalidates the plan or package when the revokee no longer holds the privilege from any other source.

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:

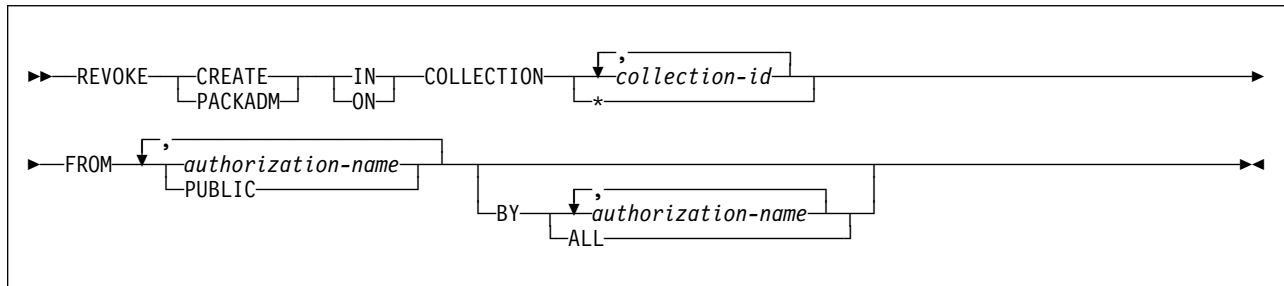
- If X is revoked, the user still has privilege P.
- If P is revoked, the user still has the privilege because it is inherent in X.

Ownership privileges: The privileges inherent in the ownership of an object cannot be revoked.

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 443 for a description of the FROM clause.

BY

Refer to “REVOKE” on page 443 for a description of the BY clause.

Example

Revoke the privilege to create new packages in collections QAACLONE and DSN8CC51 from CLARK.

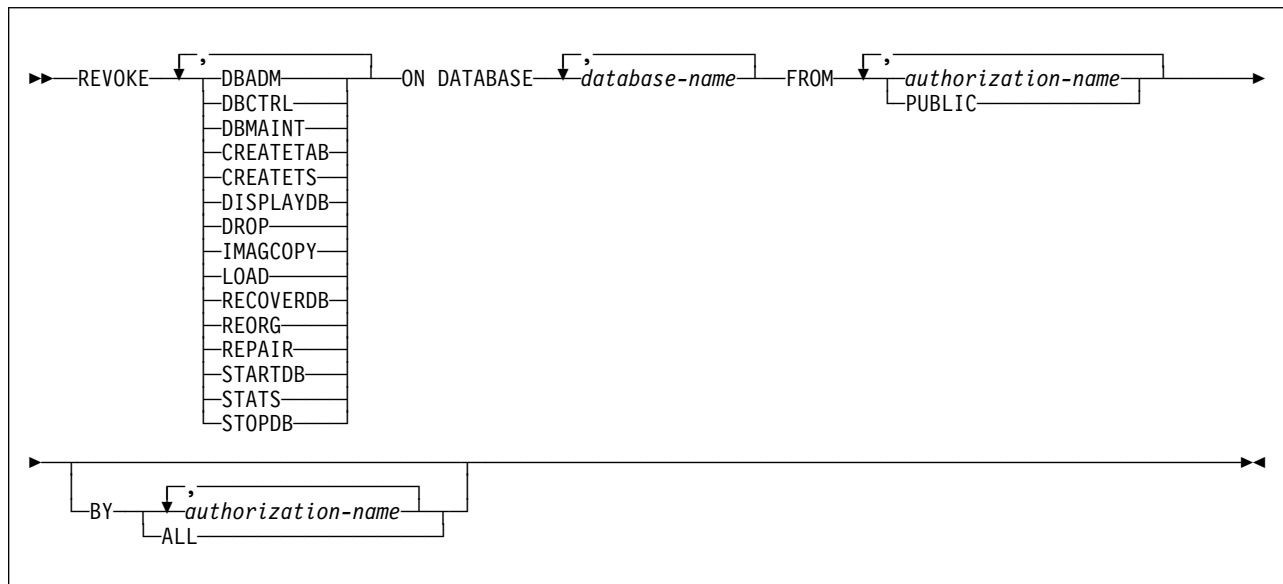
```
REVOKE CREATE IN COLLECTION QAACLONE, DSN8CC51 FROM CLARK;
```

REVOKE (Database Privileges)

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.

CREATETS

Revokes the privilege to create new table spaces.

DISPLAYDB

Revokes the privilege to issue the DISPLAY DATABASE command.

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 443 for a description of the FROM clause.

BY

Refer to “REVOKE” on page 443 for a description of the BY clause.

Examples

Example 1: Revoke drop privileges on database DSN8D51A from user PEREZ.

```
REVOKE DROP
  ON DATABASE DSN8D51A
  FROM PEREZ;
```

Example 2: Revoke repair privileges on database DSN8D51A from all local users. (Grants to specific users will not be affected.)

```
REVOKE REPAIR
  ON DATABASE DSN8D51A
  FROM PUBLIC;
```

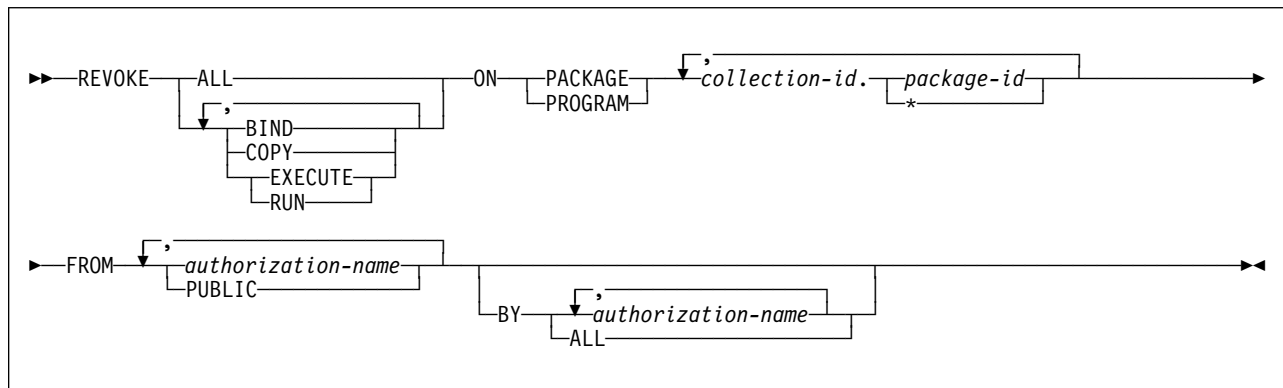
Example 3: Revoke authority to create new tables and load tables in database DSN8D51A from users WALKER, PIANKA, and FUJIMOTO.

```
REVOKE CREATETAB,LOAD
  ON DATABASE DSN8D51A
  FROM WALKER,PIANKA,FUJIMOTO;
```

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 407 for “GRANT (Package Privileges)” on page 406.)

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.

The word PROGRAM can be used in place of PACKAGE.

FROM

Refer to “REVOKE” on page 443 for a description of the FROM clause.

BY

Refer to “REVOKE” on page 443 for a description of the BY clause.

Example

Revoke the privilege to copy all packages in collection DSN8CC51 from LEWIS.

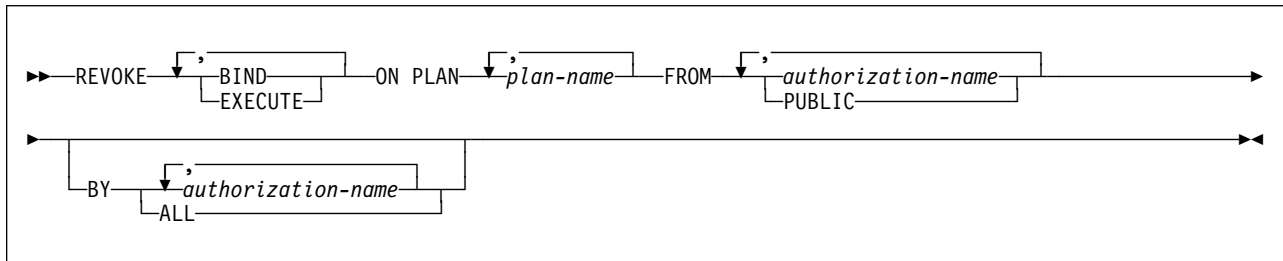
```
REVOKE COPY ON PACKAGE DSN8CC51.* FROM LEWIS;
```

REVOKE (Plan Privileges)

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 443 for a description of the FROM clause.

BY

Refer to “REVOKE” on page 443 for a description of the BY clause.

Examples

Example 1: Revoke authority to bind plan DSN8IP51 from user JONES.

```
REVOKE BIND ON PLAN DSN8IP51 FROM JONES;
```

Example 2: Revoke authority previously granted to all users at the current server to bind and execute plan DSN8CP51. (Grants to specific users will not be affected.)

```
REVOKE BIND,EXECUTE ON PLAN DSN8CP51 FROM PUBLIC;
```

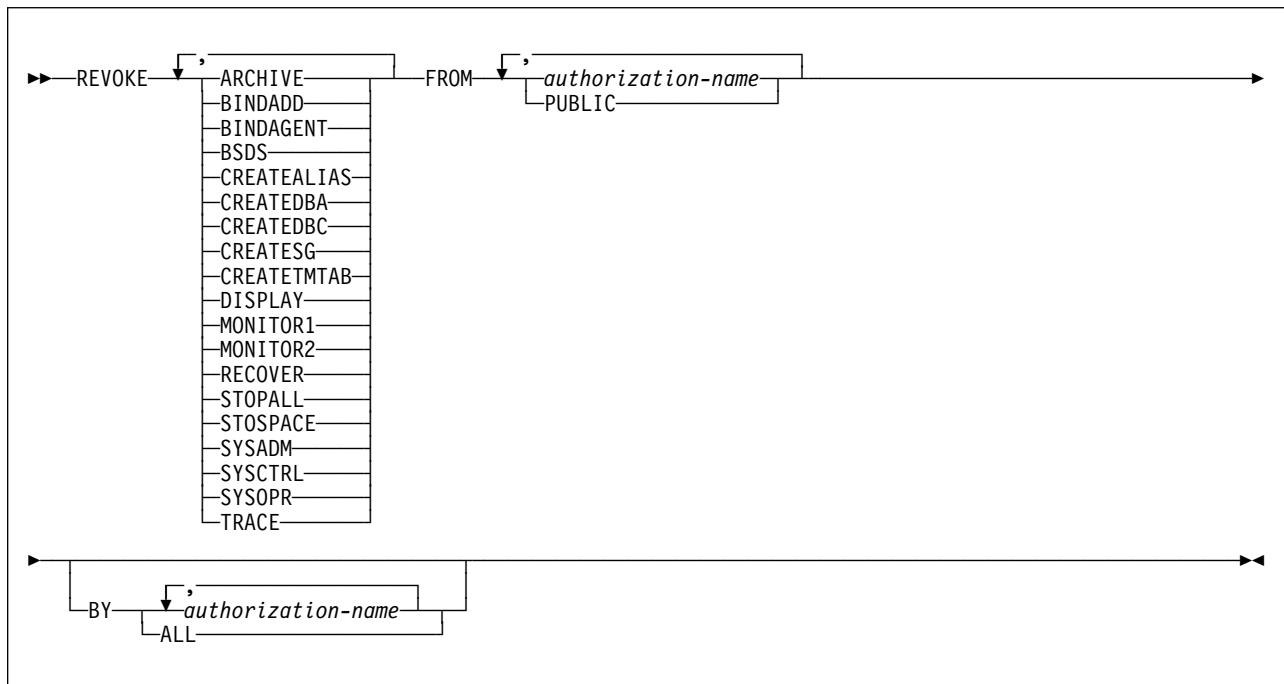
Example 3: Revoke authority to execute plan DSN8CP51 from users ADAMSON and BROWN.

```
REVOKE EXECUTE ON PLAN DSN8CP51 FROM ADAMSON,BROWN;
```

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.

REVOKE (System Privileges)

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 do the following:

- Use the DISPLAY ARCHIVE command for archive log information
- Use the DISPLAY BUFFERPOOL command for the status of buffer pools
- Use the DISPLAY DATABASE command for the status of all databases
- Use the DISPLAY LOCATION command for statistics about threads with a distributed relationship
- Use the DISPLAY THREAD command for information on active threads within DB2
- Use 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.

TRACE

Revokes the privilege to use the MODIFY TRACE, START TRACE, and STOP TRACE commands.

FROM

Refer to "REVOKE" on page 443 for a description of the FROM clause.

BY

Refer to “REVOKE” on page 443 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;
```

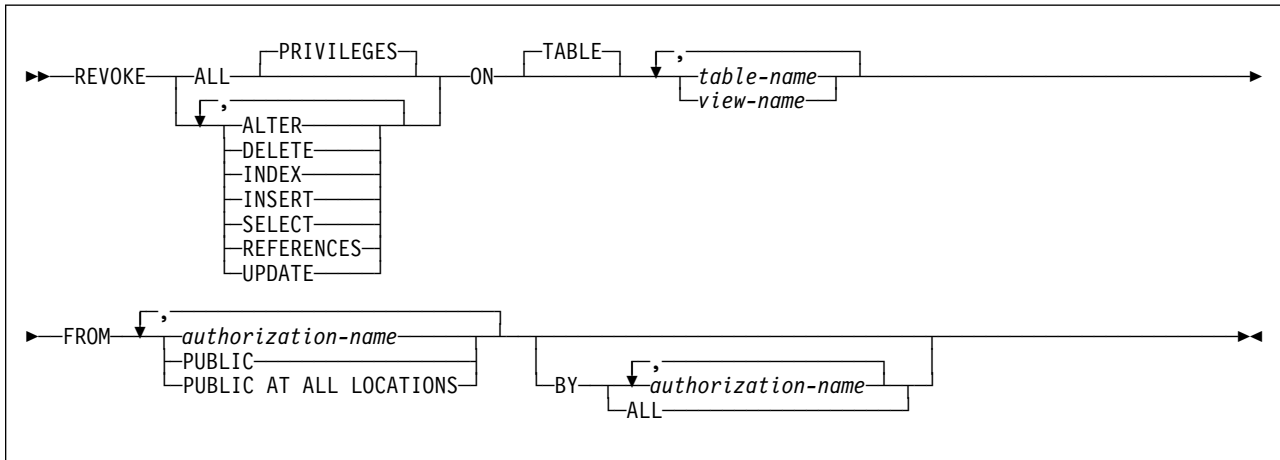
Example 3: Revoke TRACE privileges previously granted to all local users. (Grants to specific users will not be affected.)

```
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 a 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.

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.

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.

FROM

Refer to “REVOKE” on page 443 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 443 for a description of the BY clause.

Notes

For a temporary table or a view of a temporary table, only ALL or ALL PRIVILEGES can be revoked. Specific table or view privileges cannot be revoked.

Examples

Example 1: Revoke SELECT privileges on table DSN8510.EMP from user PULASKI.

```
REVOKE SELECT ON TABLE DSN8510.EMP FROM PULASKI;
```

Example 2: Revoke update privileges on table DSN8510.EMP previously granted to all local DB2 users. (Grants to specific users are not affected.)

```
REVOKE UPDATE ON TABLE DSN8510.EMP FROM PUBLIC;
```

Example 3: Revoke all privileges on table DSN8510.EMP from users KWAN and THOMPSON.

```
REVOKE ALL ON TABLE DSN8510.EMP FROM KWAN, THOMPSON;
```

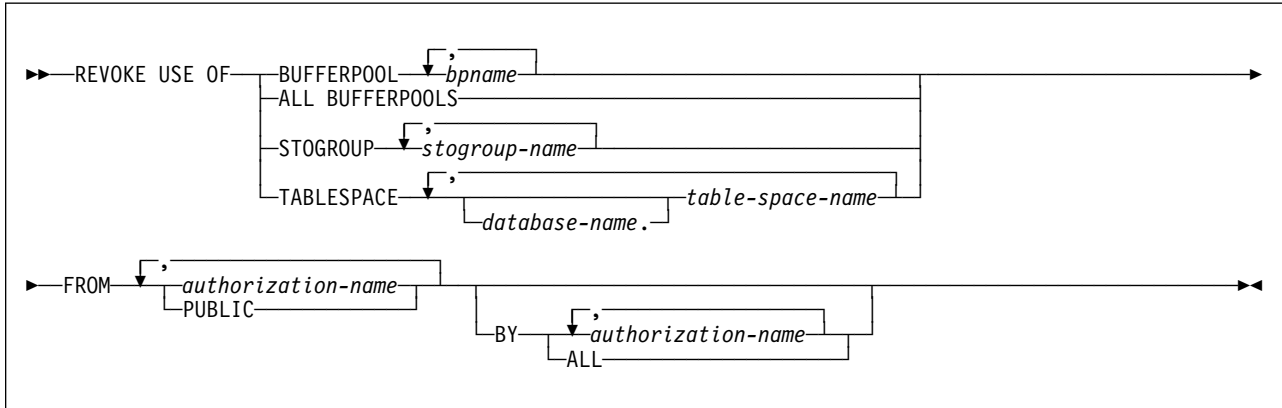
Example 4: Revoke the grant of SELECT and UPDATE privileges on the table DSN8510.DEPT to every user in the network. Doing so does not affect users who obtained these privileges from some other grant.

```
REVOKE SELECT, UPDATE ON TABLE DSN8510.DEPT  
FROM PUBLIC AT ALL LOCATIONS;
```

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 48 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.

FROM

Refer to “REVOKE” on page 443 for a description of the FROM clause.

BY

Refer to “REVOKE” on page 443 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 DSN8S51D in the database DSN8D51A. 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 DSN8D51A.DSN8S51D
FROM PUBLIC;
```

ROLLBACK

ROLLBACK

The ROLLBACK statement ends a unit of recovery and backs out 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.

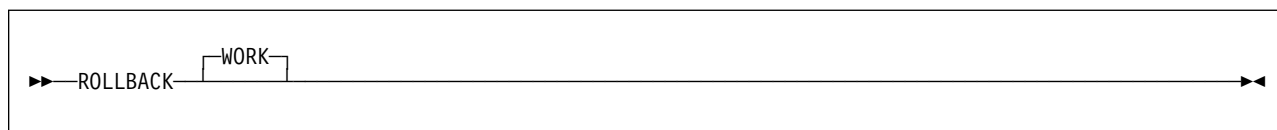
Invocation

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 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.

All locks implicitly acquired during the unit of recovery are released. See “LOCK TABLE” on page 426 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 temporary table of the application process are deleted.

ROLLBACK has no effect on connections.

Notes

When the ROLLBACK statement is used in a program executing as a stored procedure, the rollback does not take place immediately. Instead, the caller receives a negative SQLCODE for the statement, and the unit of work is placed in a “must rollback” state.

The SQL ROLLBACK statement cannot be used in the IMS or CICS environment. To do a rollback operation 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 the CICS or IMS environment can behave differently than the SQL ROLLBACK statement with regards to the closing of cursors that were

declared using the WITH HOLD option. 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.

In all DB2 environments, the abend of a process is an implicit rollback operation.

Example

Roll back all DB2 database changes made since the unit of recovery was started.

```
ROLLBACK WORK;
```

SELECT INTO

The SELECT INTO statement produces a result table containing 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.

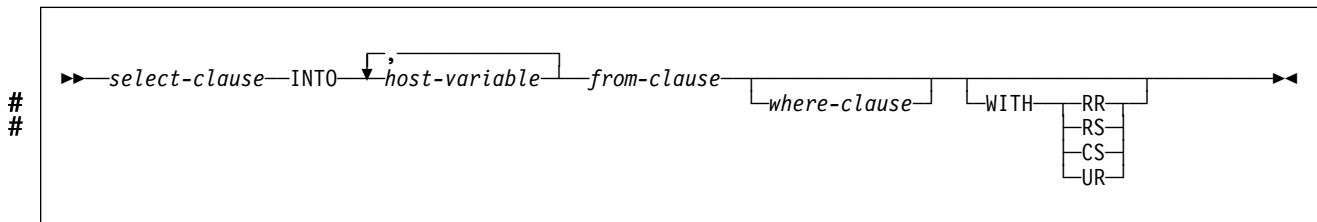
This statement cannot be included in a REXX application program.

Authorization

The privileges 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
- 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*, and *select-clause*, in this order. The *from-clause* must not identify a view that includes a *group-by-clause* or a *having-clause*. See “Chapter 5. Queries” on page 167 for a description of these clauses.

INTO *host-variable*,...

Each *host-variable* must identify a structure or variable that is described 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 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 513.)

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 3. Language Elements” on page 43. 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 3. Language Elements” on page 43. 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.

#	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

SELECT INTO

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
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

Notes

When you set a host variable with an expression, you must specify a table in the FROM clause even though you may not be interested in the contents of the table. Instead of creating your own dummy table to reference, you can use DB2 catalog table SYSIBM.SYSDUMMY1. For an example, see “Example 3”.

Examples

Example 1: Put the maximum salary in DSN8510.EMP into the host variable MAXSALRY.

```
EXEC SQL SELECT MAX(SALARY)
        INTO :MAXSALRY
        FROM DSN8510.EMP;
```

Example 2: Put the row for employee 528671, from DSN8510.EMP, into the host structure EMPREC.

```
EXEC SQL SELECT * INTO :EMPREC
        FROM DSN8510.EMP
        WHERE EMPNO = '528671'
END-EXEC.
```

Example 3: Put a date that is twenty days from the current date into host variable DUEDATE. Use DB2 catalog table SYSIBM.SYSDUMMY1 as the table referenced.

```
SELECT CURRENT DATE + 20 DAYS INTO :DUEDATE FROM SYSIBM.SYSDUMMY1
```

SET CONNECTION

The SET CONNECTION statement establishes the application server of the process by identifying one of its existing connections.

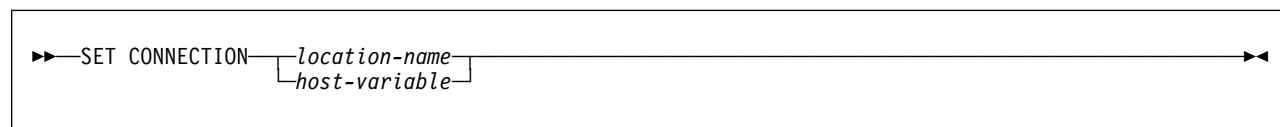
Invocation

This statement can only be embedded in an application program. It is an executable statement that cannot be dynamically prepared. SET CONNECTION cannot be used if the program is executing as a stored procedure.

Authorization

None required.

Syntax



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 may be up to 17 bytes.)
- It must be preceded by a colon and 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.

SET 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 application server S is placed in the SQLERRP field of the SQLCA. If the application server is an IBM product, the information has the form *pppvrrm*, where:
 - *ppp* is:
 - ARI for SQL/DS
 - DSN for DB2
 - QSQ for OS/400
 - *vv* is a two-digit version identifier such as '05'.
 - *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 5 of DB2 for OS/390, the value of SQLERRP is 'DSN05010'.
- 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.

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 DEGREE

The SET CURRENT DEGREE statement assigns a value to the CURRENT DEGREE special register.

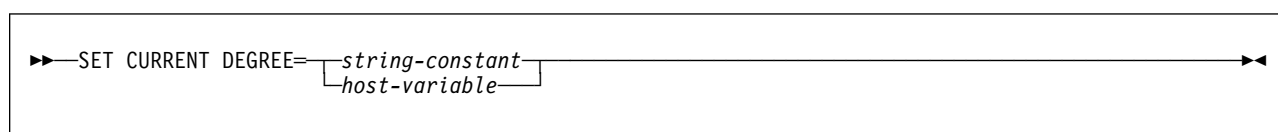
Invocation

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



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 Section 6 of *Application Programming and SQL Guide*.

Examples

Example 1: The following statement inhibits parallel operations:

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

Example 2: The following statement allows parallel operations:

```
SET CURRENT DEGREE = 'ANY';
```

SET CURRENT PACKAGESET

The SET CURRENT PACKAGESET statement assigns a value to the CURRENT PACKAGESET special register.

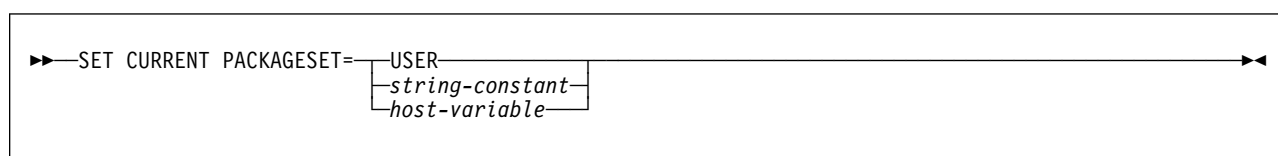
Invocation

This statement can only be embedded in an application program. It is an executable statement that cannot be dynamically prepared.

Authorization

None required.

Syntax



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 no 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 Section 4 of *Application Programming and SQL Guide*.

SET CURRENT PACKAGESET is executed by the application requester and is therefore classified as a local SET statement in DRDA.

CURRENT PACKAGESET special register and stored procedures: When a stored procedure receives control, the value of the special register CURRENT PACKAGESET is set to the value of the COLLID column of the SYSIBM.SYSPROCEDURES row that is associated with the stored procedure. The stored procedure can use the command SET CURRENT PACKAGESET to change the value of the special register CURRENT PACKAGESET. This allows the stored procedure to select the version of the DB2 package used to process the SQL statements in the stored procedure and in any external subroutines it calls.

When control returns from the stored procedure to the calling program, the special register CURRENT PACKAGESET is restored to the value it was 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, and the value must be 'DEC15' or 'DEC31'. 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

<pre> ▶▶ SET CURRENT RULES = <i>string-constant</i> <i>host-variable</i> ▶▶ </pre>
--

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 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 81.

Example

Set the SQL rules to be followed to DB2.

```
EXEC SQL SET CURRENT RULES = 'DB2';
```

SET CURRENT SQLID

The SET CURRENT SQLID statement assigns a value to the CURRENT SQLID special register.

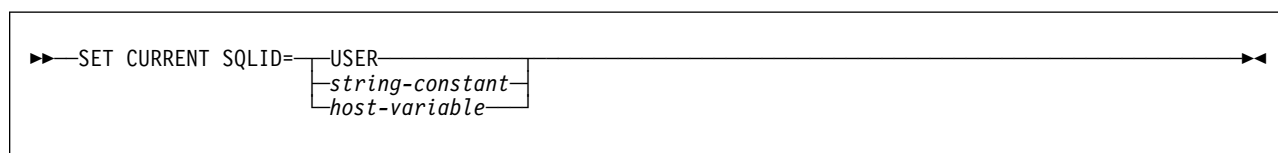
Invocation

This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared. SET CURRENT SQLID cannot be used if the program is executing as a stored procedure or the bind option DYNAMICRULES(BIND) applies.

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



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.

The initial value of the SQL authorization ID is established during connection or signon 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.

SET CURRENT SQLID is executed by the application server and is therefore classified as a non-local SET statement in DRDA.

Example

Set the CURRENT SQLID to the primary authorization ID.

```
SET CURRENT SQLID=USER;
```

SET host-variable

The SET *host-variable* statement assigns the value of a special register to a host variable.

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 host-variable = special-register ▶▶
```

Description

host-variable

Identifies the host variable to which the value of the special register is assigned. The reference must not include an indicator variable.

special-register

Identifies the special register whose value is placed in the host variable.

Notes

The assignment rules are essentially the same as the assignment rules for FETCH and SELECT INTO:

- If CURRENT TIMEZONE is specified, the host variable must be a numeric variable with the capacity to hold the value.
- If CURRENT DATE, CURRENT TIME, or CURRENT TIMESTAMP is specified, the host variable must be a character string variable of a minimum length as defined in “Datetime Assignments” on page 71.
- If USER, CURRENT SQLID, CURRENT SERVER, CURRENT DEGREE, or CURRENT PACKAGESET is specified, the host variable must be a character string variable with a length attribute that is not less than the register length.

For a description of each special register, see “Special Registers” on page 78, beginning on page 78.

Examples

Example 1: Set the host variable SERVER to the name of the current server.

```
EXEC SQL SET :SERVER = CURRENT SERVER;
```

Example 2: Set the host variable XTIME to the local time at the current server.

```
EXEC SQL SET :XTIME = CURRENT TIME;
```

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. 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 updates are allowed, or a view, and whether the statement is a searched UPDATE 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 UPDATE privilege on the table
- The UPDATE privilege on each column to be updated
- Ownership of the table
- DBADM authority on the database containing 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

Searched UPDATE and SQL standard rules: In a searched UPDATE, the SELECT privilege is required in addition to the UPDATE privilege when the option for the SQL standard is set 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'.

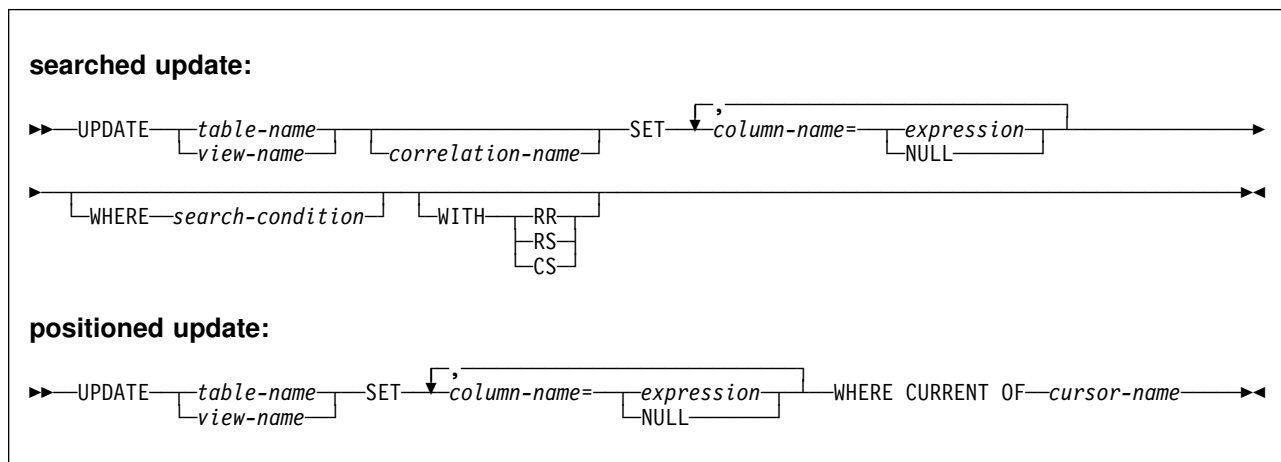
UPDATE

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 341.

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 169.

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 and the bind option DYNAMICRULES(RUN) applies, the privilege set is the union of the privilege sets held by each authorization ID of the process. If the statement is dynamically prepared and the bind option DYNAMICRULES(BIND) applies, the privilege set is the privileges held by the authorization ID of the owner of the plan or package.

Syntax



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:

- A temporary table
- A view of a temporary table
- A catalog table with no updateable columns
- A view of a catalog table with no updateable columns
- A read-only view. (For a description of a read-only view, see “CREATE VIEW” on page 341.)

In the IMS or CICS environments, the DB2 subsystem containing 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 updateable column. If a column of a catalog table is updateable, then its description in Appendix D, “DB2 Catalog Tables” on page 529 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 *Administration Guide* .

correlation-name

Can be used within *search-condition* to designate the table or view. (For an explanation of *correlation-name*, see “Correlation Names” on page 84.)

SET

Introduces a list of column names and values. The column names must not be qualified, and a column must not be specified more than once.

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 view column derived
from a scalar function, constant, or expression.

For a positioned update, allowable column names can be further restricted to those in a certain list. This list appears in the FOR UPDATE OF clause of the SELECT statement for the associated cursor. If the select statement is dynamically prepared, the FOR UPDATE OF clause must always be present. Otherwise, the clause can be omitted using the conditions described in “Positioned Updates of Columns” on page 126.

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 or **NULL**

Indicates the new value of the column. The *expression* is any expression of the type described in “Expressions” on page 92. It must not include a column function. NULL specifies the null value.

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.

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 3. Language Elements” on page 43. Each *column-name* in the search condition, other than in a subquery, must identify a column of the table or view. The search condition must not include a subquery where the base object of both the UPDATE and the subquery is the same table.

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.

UPDATE

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 347.

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, and the columns to be updated must be identified in the FOR UPDATE OF clause of that SELECT statement. The result table of the cursor must not be read-only. (For an explanation of read-only result tables, see “DECLARE CURSOR” on page 347.)

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 cursor 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.

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 3. Language Elements” on page 43.

If the update value is... Then the column must...

the null value allow null values.

a number	be a numeric column with the capacity to represent the integral part of the number.
a character string	be a character string column with a length attribute that is not less than the length of the string. The column can also be a DATE, TIME, or TIMESTAMP column, in which case the update value must be a valid character string representation of a date, time, or timestamp, respectively.
a graphic string	be a graphic string column with a length attribute that is not less than the length of the string.
a datetime value	be a DATE, TIME, or TIMESTAMP column with the same data type or a character string column of an appropriate length as specified in “Chapter 3. Language Elements” on page 43. For more details on datetime special registers, see “Datetime special registers” on page 78.

- *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 341.

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.

Number of rows updated: After an UPDATE statement completes execution, the value of SQLERRD(3) in the SQLCA is the number of rows updated. (For a description of the SQLCA, see “SQL Communication Area (SQLCA)” on page 513.)

Locking: Unless appropriate locks already exist, one or more exclusive locks are acquired by the execution of a successful UPDATE statement. Until the locks are released by a commit or rollback operation, the updated row can only be accessed by the application process that performed the update and the locks can prevent other application processes from performing operations on the table.

UPDATE

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 78, 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.

Examples

The following examples refer to the sample table DSN8510.EMP.

Example 1: Change employee 000190's telephone number to 3565 in DSN8510.EMP.

```
UPDATE DSN8510.EMP
  SET PHONENO='3565'
  WHERE EMPNO='000190';
```

Example 2: Give each member of department D11 a 100-dollar raise.

```
UPDATE DSN8510.EMP
  SET SALARY = SALARY + 100
  WHERE WORKDEPT = 'D11';
```

Example 3: Employee 000250 is going on a leave of absence. Set the salary to null.

```
UPDATE DSN8510.EMP
  SET SALARY = NULL
  WHERE EMPNO='000250';
```

Example 4: Double the salary of the employee represented by the row on which the cursor C1 is positioned.

```
EXEC SQL UPDATE DSN8510.EMP
  SET SALARY = 2 * SALARY
  WHERE CURRENT OF C1;
```

WHENEVER

The WHENEVER statement specifies the host language statement to be executed when a specified exception condition occurs.

Invocation

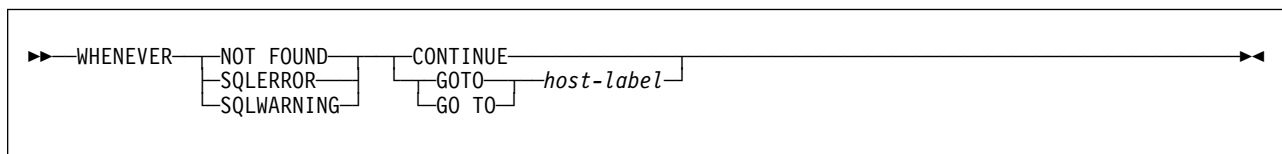
This statement can only be embedded in an application program. It is not an executable statement.

This statement cannot be included in a REXX application program.

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*.

WHENEVER

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 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 7. 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*.

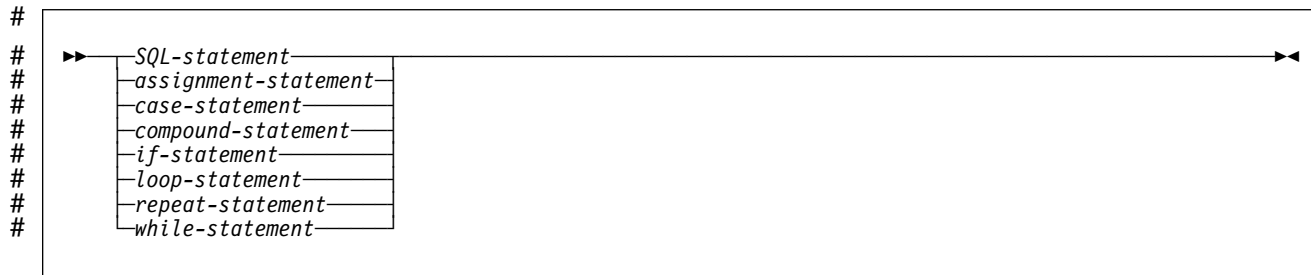
This chapter contains syntax diagrams, semantic descriptions, rules, and examples
of the use of the statements that constitute the procedure body.

SQL Procedure Body

Procedure body

The procedure body contains the source code for an SQL stored procedure.

Syntax



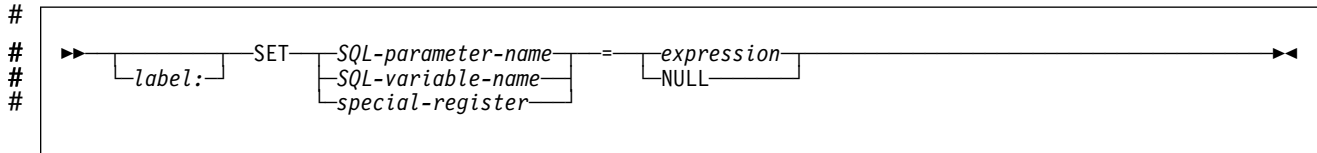
Notes

See Table 32 on page 511 for a list of valid values for *SQL-statement*.

Assignment statement

The assignment statement assigns a value to an output parameter or to an SQL
 # variable.

Syntax



Description

- # **label**
 # Defines the label for the assignment statement. 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**
 # Identifies the SQL variable that is the assignment target. An SQL variable must
 # be declared before it is used. For information on declaring SQL variables, see
 # “Compound statement” on page 491.
- # **special-register**
 # Identifies the special register that is the assignment target. If the special
 # register accepts a schema name, DB2 determines whether the value on the
 # right side of the SET statement is an SQL variable. If the value is an SQL
 # variable, DB2 assigns the value in the SQL variable to the special register. If
 # the value is not an SQL variable, DB2 determines that the value is a schema
 # name and assigns that name to the special register.
- # **expression or NULL**
 # Specifies the expression or value that is the assignment source. See
 # “Expressions” on page 92 for information on expressions.

Notes

- # Assignments statements in SQL procedures must conform to the SQL assignment
 # rules. See “Assignment and Comparison” on page 65 for assignment rules.
- # The data type of the target and source must be compatible.
- # 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 negative SQLCODE is set.

Assignment Statement (SQL procedure)

If truncation of the whole part of the number occurs on assignment to a numeric variable, a negative SQLCODE is set.

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.

Examples

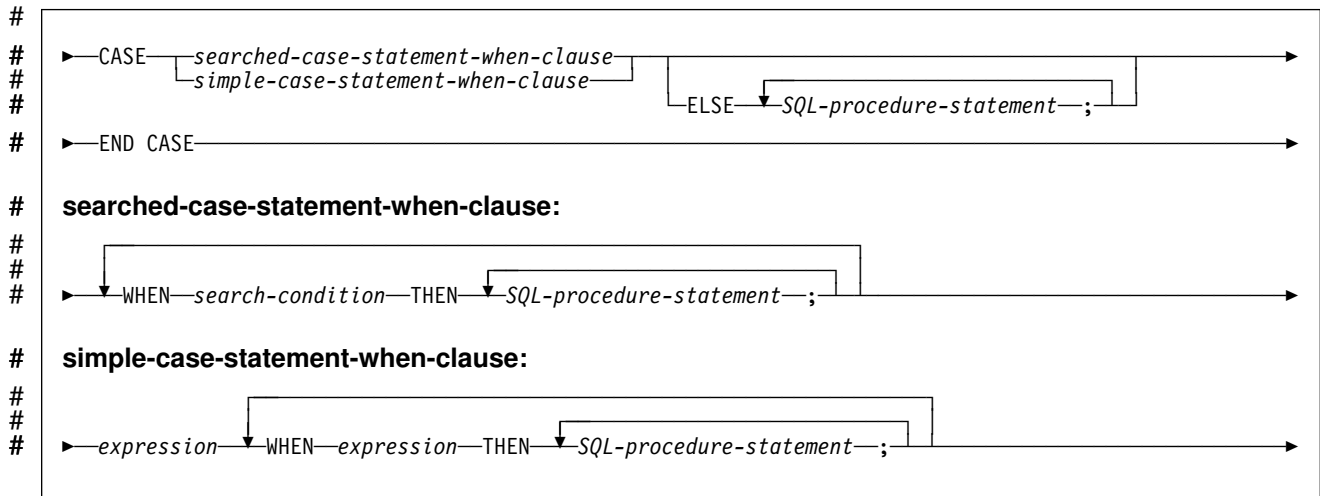
Increase the SQL variable p_salary by 10 percent.
SET p_salary = p_salary + (p_salary * .10)

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)

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 103

Syntax**# Description**

CASE
 # Begins a *case-expression*.

searched-case-statement-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-case-statement-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. Specifies the result for each WHEN keyword when the expressions
 # are equal.

The *expression* prior to the first WHEN keyword is tested for equality with the
 # value of the *expression* that follows the WHEN keyword. 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.

SQL-procedure-statement
 # Specifies a statement that follows the THEN and ELSE keyword. The statement
 # must be one of the statements listed under “SQL procedure statement” on
 # page 504. It 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.

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.

CASE (SQL procedure)

```
#          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.
```

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
```

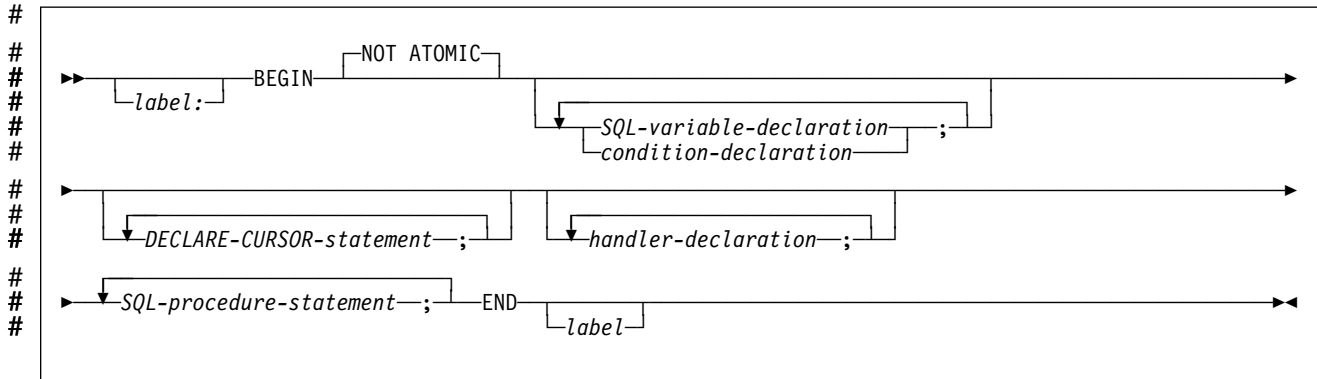
```
#          Use a searched case statement when clause to update column DEPTNAME in  
#          table DEPT, depending on the value of SQL variable v_workdept.
```

```
#          CASE  
#          WHEN v_workdept = 'A00'  
#             THEN UPDATE department SET  
#                deptname = 'DATA ACCESS 1';  
#          WHEN v_workdept = 'B01'  
#             THEN UPDATE department SET  
#                deptname = 'DATA ACCESS 2';  
#          ELSE UPDATE department SET  
#                deptname = 'DATA ACCESS 3';  
#          END CASE
```

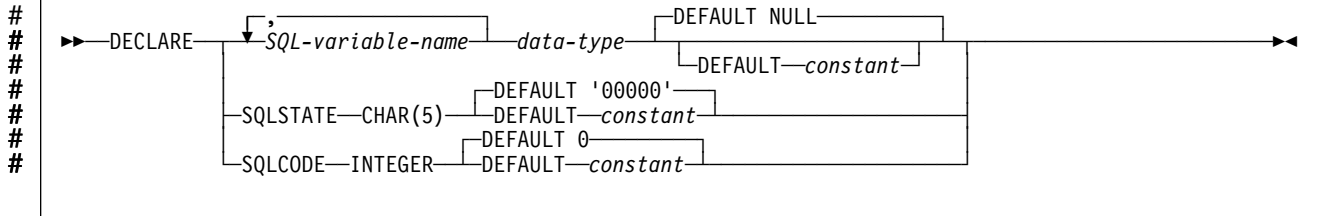
Compound statement

A compound statement contains a group of statements and declarations for SQL
 # variables, cursors, and condition handlers.

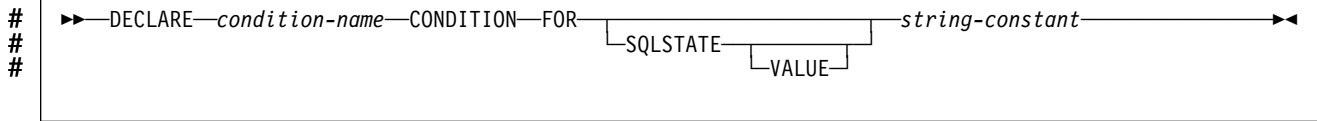
Syntax



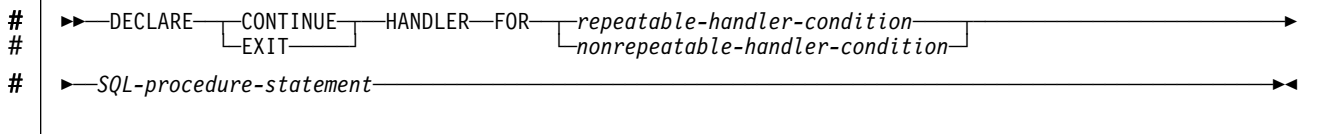
SQL-variable-declaration:



condition-declaration:



handler-declaration:

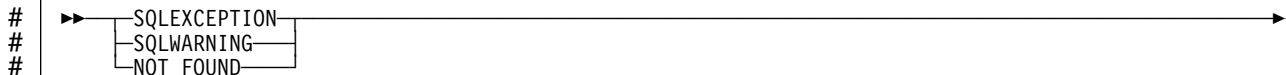


Compound Statement (SQL procedure)

repeatable-handler-condition:



nonrepeatable-handler-condition:



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.

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
18 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.

SQLSTATE
A special variable named SQLSTATE that DB2 sets to the SQLSTATE value
that it returns after it executes an SQL statement. You can assign a value to
this variable. However, exception handlers ignore the value that you set, and
the next SQL statement replaces the value that you set.

```

#           SQLCODE
#           A special variable named SQLCODE that DB2 sets to the SQLCODE value that
#           it returns after it executes an SQL statement. You can assign a value to this
#           variable. However, exception handlers ignore the value that you set, and the
#           next SQL statement replaces the value that you set.

#           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 procedure)” on page 295 for a
#           description of SQL data types and lengths.

#           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.

#           If the SQL variable name is SQLCODE and the data type is INT, the variable is
#           used as a stand-alone SQLCODE in the procedure and can be checked to
#           determine whether SQL statements are successful. Similarly, if the SQL
#           variable name is SQLSTATE and the data type is CHAR(5), the variable is
#           used as a stand-alone SQLSTATE in the procedure. After the SQLCODE and
#           SQLSTATE variables are declared, they can be referenced anywhere in the
#           procedure. The SQLCODE and SQLSTATE variables cannot be set to NULL.
#           An SQLCODE or SQLSTATE variable should not be used on the left side of an
#           assignment statement.

#           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.

#           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'.

#           declare-cursor-statement
#           Declares a cursor. Each cursor in the procedure body must have a unique
#           name. The cursor can be referenced only from within the compound statement.

#           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 504 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

```

Compound Statement (SQL procedure)

```
#          that raised the exception is an IF, CASE, or WHILE 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.

#          The conditions under which the handler is invoked are:

#          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 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 and condition declarations
#          2. Cursor declarations
#          3. Handler declarations
#          4. Procedure body statements (CASE, IF, LOOP, REPEAT, WHILE, SQL)

#          Compound statements cannot be nested.

#          Unlike host variables, SQL variables are not preceded by colons when they are
#          used in SQL statements.

#          Datetime arithmetic operations cannot be performed on SQL variables.

#          The following rules apply to handlers:
#          • 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.
#
#           • If an error occurs for which there is no handler, execution of the compound
#           statement is terminated.
#
#           • 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:
#
#           • SQLSTATE must be declared as CHAR(5), and SQLCODE must be declared
#           as INTEGER.. Any other data type results in an error.
#
#           • A null value cannot be assigned to SQLSTATE or SQLCODE.
#
#           • To avoid losing the contents of SQLCODE and SQLSTATE, assign their values
#           to other variables immediately after SQLCODE and SQLSTATE are set. If a
#           handler is declared for SQLSTATE, this assignment needs to be the first
#           statement in the SQL procedure statement of 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_firstnme 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 FIRSTNME, MIDINIT, LASTNAME,
#         EDLEVEL, SALARY
#     FROM EMP;
#     DECLARE CONTINUE HANDLER FOR not_found SET NOROWS=1;
#     OPEN c1;
#     FETCH c1 INTO v_firstnme, 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 491.

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
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 statement

The GOTO statement transfers program control to a labelled statement.

Syntax

```
#  
# ─▶ GOTO label ─▶
```

Description

label
Specifies a labelled statement at which processing is to continue.
A label name cannot be the same as the name of the SQL procedure in which
the label is used.

Notes

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.

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 EXIT;
# END IF;
# IF rating = 1
# THEN SET new_salary =
# new_salary + (new_salary * .10);
# ELSEIF rating = 2
# THEN SET new_salary =
# new_salary + (new_salary * .05);
# END IF;
# UPDATE EMP
# SET SALARY = new_salary
# WHERE EMPNO = v_empno;
# EXIT: SET return_parm = service;
# END
```

LEAVE statement

The LEAVE statement transfers program control out of a loop or a block of code.

Syntax

```
#  
#   ▶—LEAVE—label—▶
```

Description

label
Specifies the label of the block 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

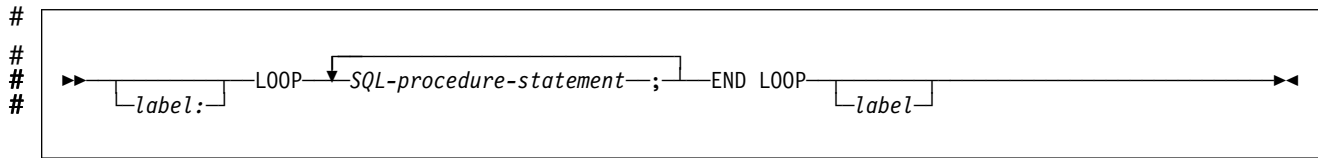
Use a LEAVE statement to transfer control out of a LOOP statement when a
negative SQLCODE occurs.

```
#       ftch_loop: LOOP  
#        FETCH c1 INTO  
#         v_firstnme, v_midinit,  
#         v_lastname, v_edlevel, v_salary;  
#        IF SQLCODE=100 THEN LEAVE ftch_loop;  
#        END IF;  
#        END LOOP
```

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.

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.

```

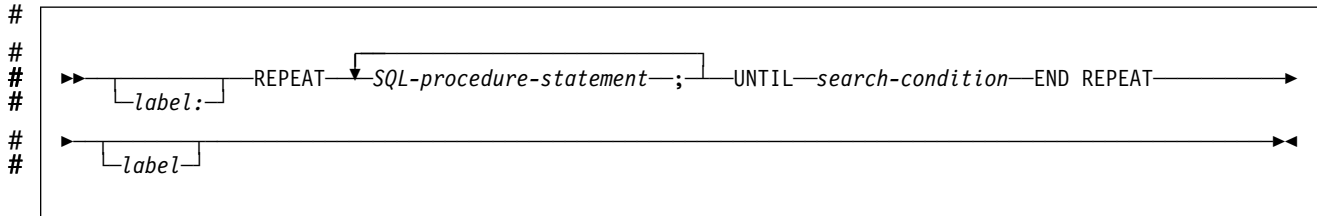
ftch_loop: LOOP
  FETCH c1 INTO
    v_firstname, v_midinit,
    v_lastname, v_edlevel, v_salary;
  IF SQLCODE<>0 THEN SET badsql=1;
  END IF;
END LOOP
  
```

REPEAT (SQL procedure)

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.

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.

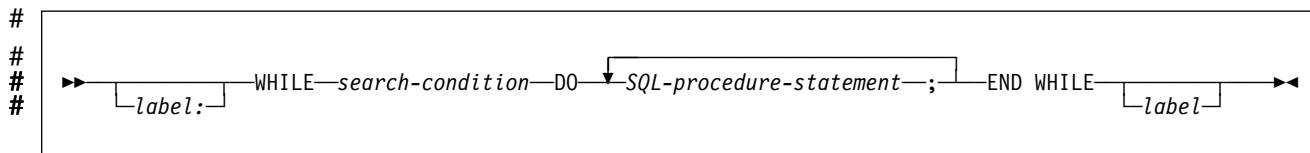
Examples

Use a REPEAT statement to fetch rows from a table.

```
# fetch_loop:  
# REPEAT  
#   FETCH c1 INTO  
#     v_firstnme, v_midinit, v_lastname;  
# UNTIL  
#   SQLCODE <> 0  
# END REPEAT fetch_loop
```


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.

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
```

SQL procedure statement

Syntax



Notes

See Table 32 on page 511 for a list of valid values for *nested-SQL-statement*.

Appendix A. Limits in DB2 for OS/390

System storage limits might preclude the limits specified here. The limit for items not specified below is system storage.

Table 25. Identifier Length Limits

Item	Limit
Longest alias, synonym, collection ID, correlation name, statement name, or name of a column, cursor, index, table, view, or table check constraint	18 bytes
Longest authorization name, package name, or name of a plan, database, table space, storage group, or referential constraint	8 bytes
Longest host identifier	64 bytes
Longest server name or location identifier	16 bytes

Table 26. Numeric Limits

Item	Limit
Smallest INTEGER value	-2147483648
Largest INTEGER value	2147483647
Smallest SMALLINT value	-32768
Largest SMALLINT value	32767
Largest decimal precision	31
Smallest FLOAT value	About -7.2×10^{75}
Largest FLOAT value	About 7.2×10^{75}
Smallest positive FLOAT value	About 5.4×10^{-79}
Largest negative FLOAT value	About -5.4×10^{-79}
Smallest DECIMAL value	$1 - 10^{31}$
Largest DECIMAL value	$10^{31} - 1$

Table 27 (Page 1 of 2). String Length Limits

Item	Limit
# Maximum length of CHAR	255 bytes
Maximum length of GRAPHIC	127 characters
Maximum length of VARCHAR ²⁹	4046 bytes, for 4KB pages 32704 bytes, for 32KB pages
Maximum length of VARGRAPHIC ²⁹	4046 bytes (2023 DBCS characters), for 4KB pages 32704 bytes (16352 DBCS characters), for 32KB pages
# Maximum length of a character constant	255 bytes

²⁹ 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.

Limits in DB2 for OS/390

Table 27 (Page 2 of 2). String Length Limits

Item	Limit
Maximum length of a hexadecimal constant	254 digits
Maximum length of a graphic string constant	124 characters
Maximum length of a concatenated character string	32764 bytes
Maximum length of a concatenated graphic string	16382 DBCS characters

Table 28. 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 29 (Page 1 of 2). 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)	750 or fewer 749 if the table is a dependent
Maximum number of base tables in a view	15
Maximum row and record sizes for a table	See "Maximum record size" on page 324 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 large 254 for table spaces that are large
# Maximum size of a partition (table space or index)	For table spaces that are not large: 4 gigabytes, for 1 to 16 partitions 2 gigabytes, for 17 to 32 partitions 1 gigabyte, for 33 to 64 partitions For table spaces that are large: 4 gigabytes, for 1 to 254 partitions
Maximum size of a DBRM entry	131072 bytes
Longest index key	254 bytes less the number of key columns that allow nulls. See Table 18 on page 284.
Maximum number of bytes used in the partitioning of a partitioned index ³⁰	40
Maximum number of columns in an index key	64

³⁰ If the key of a partitioned index is longer than 40 bytes, only the first 40 bytes are used to determine the high value for each partition.

Table 29 (Page 2 of 2). DB2 Limits on SQL Statements

Item	Limit
Maximum number of tables and views that can be identified in a subselect	15 or fewer, depending on the complexity of the subselect
Maximum total length of host and indicator variables pointed to in an SQLDA	32767 bytes
Longest host variable used for insert or update	32704 bytes
Longest SQL statement	32765 bytes
Maximum number of elements in a select list	750
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
Maximum number of parameters of a stored procedure and any CALL statement referencing the procedure	As many as can be defined by the parameter list stored in SYSPROCEDURES.PARMLIST. The maximum length of the column is 3000 bytes.
Maximum number of bytes that can be passed in a single parameter of an SQL CALL statement	32765 bytes

Table 30. 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	1016 gigabytes
Largest log space	2 ⁴⁸
Largest active log data set	2 gigabytes
Largest archive log data set	2 gigabytes
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	32511
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 29 on page 506

Appendix B. Characteristics of SQL Statements in DB2 for OS/390

This appendix provides a summary of the actions that are allowed on SQL statements in DB2 for OS/390. It also contains a list of the SQL statements that can be executed in SQL procedures.

#

Actions Allowed on SQL Statements

The following table shows whether a specific DB2 statement can be executed, prepared interactively or dynamically, or processed by the application requester, the application server, or the precompiler. The letter **Y** means yes.

Table 31 (Page 1 of 2). Actions Allowed on SQL Statements in DB2 for OS/390

SQL Statement	Executable	Interactively or Dynamically Prepared	Processed by		
			Requesting System	Server	Precompiler
ALLOCATE CURSOR	Y	Y	Y		
ALTER ¹	Y	Y		Y	
ASSOCIATE LOCATORS	Y	Y	Y		
BEGIN DECLARE SECTION					Y
CALL	Y			Y	
CLOSE	Y			Y	
COMMENT	Y	Y		Y	
COMMIT	Y	Y		Y	
CONNECT (Type 1 and Type 2)	Y		Y		
CREATE ¹	Y	Y		Y	
DECLARE CURSOR					Y
DECLARE STATEMENT					Y
DECLARE TABLE					Y
DELETE	Y	Y		Y	
DESCRIBE	Y			Y	
DESCRIBE CURSOR	Y		Y		
DESCRIBE PROCEDURE	Y		Y		
DROP ¹	Y	Y		Y	
END DECLARE SECTION					Y
EXECUTE	Y			Y	
EXECUTE IMMEDIATE	Y			Y	
EXPLAIN	Y	Y		Y	
FETCH	Y			Y	
GRANT ¹	Y	Y		Y	
INCLUDE					Y

Characteristics of SQL Statements in DB2 for OS/390

Table 31 (Page 2 of 2). Actions Allowed on SQL Statements in DB2 for OS/390

SQL Statement	Executable	Interactively or Dynamically Prepared	Processed by		
			Requesting System	Server	Precompiler
INSERT	Y	Y		Y	
LABEL	Y	Y		Y	
LOCK TABLE	Y	Y		Y	
OPEN	Y			Y	
PREPARE	Y			Y ³	
RELEASE	Y		Y		
RENAME	Y	Y		Y	
REVOKE ¹	Y	Y		Y	
ROLLBACK	Y	Y		Y	
SELECT INTO	Y			Y	
SET CONNECTION	Y		Y		
SET CURRENT DEGREE	Y	Y		Y	
SET CURRENT PACKAGESET	Y		Y		
SET CURRENT PACKAGESET	Y		Y		
# SET CURRENT PRECISION	Y	Y		Y	
SET CURRENT SQLID ²	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 PACKAGESET	Y		Y		
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	
UPDATE	Y	Y		Y	
WHENEVER					

Note:

1. If the bind option DYNAMICRULES(BIND) applies, the statement cannot be dynamically prepared.
2. If the bind option DYNAMICRULES(BIND) applies, neither a static nor a dynamic SET CURRENT SQLID statement can be used.
3. The requesting system processes the PREPARE statement when the statement being prepared is ALLOCATE CURSOR or ASSOCIATE LOCATORS.

SQL statements allowed in SQL procedures

Table 32 lists the SQL statements that are valid in an SQL procedure body. The
 # table lists the SQL statements that can be used as the only statement in the SQL
 # procedure and the statements that can be nested in a compound statement.

Table 32 (Page 1 of 2). 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		
ALTER DATABASE	Y	Y
ALTER INDEX	Y	Y
ALTER STOGROUP	Y	Y
ALTER TABLE	Y	Y
ALTER TABLESPACE	Y	Y
ASSOCIATE LOCATORS		
BEGIN DECLARE SECTION		
CALL		
CLOSE		Y
COMMENT ON	Y	Y
COMMIT		
CONNECT (Type 1 and Type 2)		
CREATE ALIAS	Y	Y
CREATE DATABASE	Y	Y
CREATE GLOBAL TEMPORARY TABLE	Y	Y
CREATE INDEX	Y	Y
CREATE PROCEDURE (SQL)		
CREATE STOGROUP	Y	Y
CREATE SYNONYM	Y	Y
CREATE TABLE	Y	Y
CREATE TABLESPACE	Y	Y
CREATE VIEW	Y	Y
DECLARE CURSOR		Y
DECLARE STATEMENT		
DECLARE TABLE		
DELETE	Y	Y
DESCRIBE		
DESCRIBE CURSOR		
DESCRIBE INPUT		
DESCRIBE PROCEDURE		
DROP	Y	Y

Table 32 (Page 2 of 2). Valid SQL statements in an SQL procedure body

SQL statement	SQL statement is...	
	The only statement in the procedure	Nested in a compound statement
END DECLARE SECTION		
EXECUTE		Y
EXECUTE IMMEDIATE	Y	Y
EXPLAIN		
FETCH		Y
GRANT	Y	Y
INCLUDE		
INSERT	Y	Y
LABEL ON	Y	Y
LOCK TABLE	Y	Y
OPEN		Y
PREPARE FROM		Y
RELEASE		
RENAME	Y	Y
REVOKE	Y	Y
ROLLBACK		
SELECT		
SELECT INTO	Y	Y
SET CONNECTION		
SET special register	Y	Y
UPDATE	Y	Y
WHENEVER		

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 containing 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 518 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 33 (Page 1 of 3). Fields of SQLCA

Assembler, COBOL, or PL/I Name	C Name	FORTRAN Name	Data Type	Purpose
SQLCAID	sqlcaid	Not used.	CHAR(8)	An “eye catcher” for storage dumps, containing the text 'SQLCA'.
SQLCABC	sqlcab	Not used.	INTEGER	Contains the length of the SQLCA: 136.

Table 33 (Page 2 of 3). Fields of SQLCA

Assembler, COBOL, or PL/I Name	C Name	FORTTRAN Name	Data Type	Purpose								
SQLCODE (See note 1)	SQLCODE	SQLCOD	INTEGER	Contains the SQL return code. (See note 2) <table border="0"> <tr> <td>Code</td> <td>Means</td> </tr> <tr> <td>0</td> <td>Successful execution (though there might have been warning messages).</td> </tr> <tr> <td>positive</td> <td>Successful execution, but with an exception condition.</td> </tr> <tr> <td>negative</td> <td>Error condition.</td> </tr> </table>	Code	Means	0	Successful execution (though there might have been warning messages).	positive	Successful execution, but with an exception condition.	negative	Error condition.
Code	Means											
0	Successful execution (though there might have been warning messages).											
positive	Successful execution, but with an exception condition.											
negative	Error condition.											
SQLERRML (See note 3)	sqlerrml (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 (See note 3)	sqlerrmc (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.								
SQLERRD(1)	sqlerrd[0]	SQLERR(1)	INTEGER	Contains an internal error code.								
SQLERRD(2)	sqlerrd[1]	SQLERR(2)	INTEGER	Contains 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. SQLERRD(3) can also contain the reason code of a timeout or deadlock for SQLCODES -911 and -913.								
# # # 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.								
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 W if at least one other indicator also contains a W; otherwise, contains a blank.								
SQLWARN1	SQLWARN1	SQLWRN(1)	CHAR(1)	Contains W if the value of a string column was truncated when assigned to a host variable.								
SQLWARN2	SQLWARN2	SQLWRN(2)	CHAR(1)	Contains 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 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 W if a prepared UPDATE or DELETE statement does not include a WHERE clause.								

Table 33 (Page 3 of 3). Fields of SQLCA

Assembler, COBOL, or PL/I Name	C Name	FORTTRAN Name	Data Type	Purpose
SQLWARN5	SQLWARN5	SQLWRN(5)	CHAR(1)	Contains W if the SQL statement was not executed because it is not a valid SQL statement in DB2 for OS/390.
SQLWARN6	SQLWARN6	SQLWRN(6)	CHAR(1)	Contains 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 DISTINCT 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 Section 2 of *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" on page 515.
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 *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;

```

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 34 lists the
variables in a REXX SQLCA.

Table 34 (Page 1 of 2). Variables in a REXX SQLCA

# Variable	Contents
# SQLCODE	The SQL return code.
# SQLERRMC	One or more tokens, separated by X'FF', that are substituted for variables in the descriptions of error conditions.
# SQLERRP	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, the product signature is 'DSN'.
# SQLERRD.1	An internal error code.
# SQLERRD.2	An internal error code.
# SQLERRD.3	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.
#	For SQLCODE -911 or -913, SQLERRD.3 can also contain the reason code for a timeout or deadlock.

Table 34 (Page 2 of 2). Variables in a REXX SQLCA

# 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.
# SQLERRD.5	The position or column of a syntax error for a PREPARE or EXECUTE IMMEDIATE statement.
# SQLERRD.6	An internal error code.
# SQLWARN.0	Blank if all other indicators are blank; W if at least one other indicator also contains a W.
# SQLWARN.1	W if the value of a string column was truncated when assigned to a host variable.
# SQLWARN.2	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	W if the number of result columns is larger than the number of host variables. Z if the ASSOCIATE LOCATORS statement contains fewer locators than the stored procedure returned.
# SQLWARN.4	W if a prepared UPDATE or DELETE statement does not include a WHERE clause.
# SQLWARN.5	W if the SQL statement was not executed because it is not a valid SQL statement in DB2 for OS/390.
# SQLWARN.6	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	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	W if a character that could not be converted was replaced with a substitute character.
# SQLWARN.9	W if arithmetic exceptions were ignored during COUNT DISTINCT processing. Z if the stored procedure returned multiple result sets.
# SQLWARN.10	W if at least one character field of the SQLCA is invalid due to a character conversion error.
# SQLSTATE	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 some SQL
| statements, and is optional for some other SQL statements. 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. *Application Programming and SQL Guide* discusses ways to use the
| SQLDA.

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 followed by an arbitrary number of occurrences of a sequence of five variables collectively named SQLVAR. The meaning of the information in an SQLDA depends on its use:

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. An SQLDA can be used in a DESCRIBE *statement-name* or PREPARE INTO statement, modified with the addresses of host variables, and then reused in a FETCH statement.

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. SQLD contains the number of result sets that the stored procedure returned, and 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.

Table 35. Fields of SQLDA

C Name Assembler or PL/I Name	Data Type	Usage in DESCRIBE ¹ and PREPARE INTO	Usage in FETCH, OPEN, EXECUTE, and CALL
sqldaid SQLDAID	CHAR(8)	An “eye catcher” for storage dumps, containing the text 'SQLDA '. For DESCRIBE CURSOR, it 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. For DESCRIBE PROCEDURE, it is set to 'SQLPR'.	A plus sign (+) in the 6th byte indicates that SQLNAME contains an overriding CCSID. Otherwise, SQLDAID is not used.
sqldabc SQLDABC	INTEGER	Length of the SQLDA, equal to SQLN×44+16.	Not used.
sqln SQLN	SMALLINT	Total number of occurrences of SQLVAR.	Same.
sqld SQLD	SMALLINT	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. For DESCRIBE PROCEDURE, the number of result sets returned by the stored procedure. Contains a 0 if no result sets are returned.	The number of host variables described by occurrences of SQLVAR.

Notes:

- 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 in an Occurrence of SQLVAR

Table 36 (Page 1 of 2). Fields in an Occurrence of SQLVAR

C Name Assembler or PL/I Name	Data Type	Usage in DESCRIBE ¹ and PREPARE INTO	Usage in FETCH, OPEN, EXECUTE, and CALL
sqltype SQLTYPE	SMALLINT	Tells the data type of the column and whether or not it allows null values. For a description of the type codes, see Table 37 on page 523.	Tells the data type of the host variable and whether an indicator variable is provided. For a description of the type codes, see “SQLTYPE and SQLLEN” on page 523.

Table 36 (Page 2 of 2). Fields in an Occurrence of SQLVAR

C Name Assembler or PL/I Name	Data Type	Usage in DESCRIBE ¹ and PREPARE INTO	Usage in FETCH, OPEN, EXECUTE, and CALL
sqlen SQLLEN	SMALLINT	The length attribute of the column. For datetime columns, the length of the string representation of the value. See "SQLTYPE and SQLLEN" on page 523 for a description of allowable values.	The length attribute of the host variable. See "SQLTYPE and SQLLEN" on page 523 for a description of allowable values.
sqldata SQLDATA	pointer	For character or graphic columns, this variable contains X'0000zzzz', where zzzz is the associated CCSID. (For BIT data, zzzz is X'FFFF'). Not used for other types of data. For DESCRIBE PROCEDURE, the result set locator value associated with the result set.	Contains the address of the host variable.
sqlind SQLIND	pointer	Reserved For DESCRIBE PROCEDURE, it is set to -1.	Contains the address of an associated indicator variable, if SQLTYPE is odd.
sqlname SQLNAME	VARCHAR(30)	Contains the 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. For DESCRIBE INPUT, SQLNAME is not used. 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.	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: <ul style="list-style-type: none"> • The 6th byte of SQLDAID is '+' • SQLTYPE indicates the host variable is a string variable • The length of SQLNAME is 8 • The first two bytes of SQLNAME are X'0000'.

#

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.

Table 37 (Page 1 of 2). 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 DATA TYPE	SQLLEN	HOST VARIABLE DATA TYPE	SQLLEN
384/385	date	10 ³¹	fixed-length character string representation of a date	length attribute of the host variable
388/389	time	8 ³²	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
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, 8 for double precision	floating point	4 for single precision, 8 for double precision
484/485	packed decimal	precision in byte 1; scale in byte 2	packed decimal	precision in byte 1; scale in byte 2
496/497	large integer	4	large integer	4
500/501	small integer	2	small integer	2

³¹ Might be different if a date installation exit is specified.

³² Might be different if a time installation exit is specified.

SQLDA

Table 37 (Page 2 of 2). *SQLTYPE* and *SQLLEN* Values for *DESCRIBE*, *PREPARE INTO*, *FETCH*, *OPEN*, *EXECUTE*, and *CALL*

SQLTYPE	For <i>DESCRIBE</i> and <i>PREPARE INTO</i>		For <i>FETCH</i> , <i>OPEN</i> , <i>EXECUTE</i> , and <i>CALL</i>	
	COLUMN DATA TYPE	SQLLEN	HOST VARIABLE DATA TYPE	SQLLEN
504/505	N/A	N/A	DISPLAY SIGN LEADING SEPARATE	precision in byte 1; scale in byte 2
972/973	result set locator	4	result set locator	4

SQLDATA

The following table identifies the CCSID values that appear in the *SQLDATA* field when the *SQLVAR* element describes a string column.

Table 38. *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

The Included SQLDA

The description of the *SQLDA* that is given by *INCLUDE SQLDA* is shown below. Only Assembler, PL/I, and C, are supported. Though you can use an *SQLDA* in VS COBOL II, the *INCLUDE* statement does not provide the code. You must provide it as shown in the chapter on dynamic SQL in Section 6 of *Application Programming and SQL Guide*.

Assembler:

```

SQLDA    DSECT
SQLDAID  DS      CL8
SQLDABC  DS      F
SQLN     DS      H
SQLD     DS      H
SQLVAR   DS      0F
SQLVARN  DSECT  ,
SQLTYPE  DS      H
SQLLEN   DS      0H
SQLPRCSN DS      X
SQLSCALE DS      X
SQLDATA  DS      A
SQLIND   DS      A
SQLNAME  DS      H,CL30
SQLVSIZ  DS      *-SQLDATA
SQLSIZV  DS      *-SQLVARN

```

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 SQLSIZE FIXED(15) BINARY;
DECLARE SQLDAPTR POINTER;

```

C:

```

#ifndef SQLDASIZE
struct sqlda
{
    unsigned char  sqldaid[8];
    long          sqldabc;
    short         sqln;
    short         sqld;
    struct sqlvar
    {
        short      sqltype;
        short      sqllen;
        unsigned char *sqldata;
        short      *sqlind;
        struct sqlname
        {
            short      length;
            unsigned char data[30];
        } sqlname;
    } sqlvar[1];
};
#define SQLDASIZE(n) (sizeof(struct sqlda)+(n-1)*sizeof(struct sqlvar))
#endif

```

```

C++:
#ifndef SQLDASIZE
struct sqlvar
    { short  sqltype;
      short  sqlllen;
    unsigned char  *sqldata;
      short  *sqlind;
      struct sqlname
          { short  length;
            unsigned char  data[30];
          } sqlname;
    };
struct sqlda
    { unsigned char  sqldaid[8];
      long  sqldabc;
      short  sqln;
      short  sqld;
      struct sqlvar  sqlvar[1];
    };

#define SQLDASIZE(n) \
    ( sizeof(struct sqlda) + ((n)-1) * sizeof(struct sqlvar) )
#endif

```

Identifying an SQLDA in C

A *descriptor-name* can be specified in the CALL, DESCRIBE, EXECUTE, FETCH, and OPEN statements. When the host application is written in C, *descriptor-name* can be a pointer variable with pointer notation, for example,

```
*sqlptr
```

descriptor-name could be declared as

```
sqlda *outsqlda;
```

and afterwards 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 39 on page 527 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 39 (Page 1 of 2). Fields of a REXX SQLDA

# Variable name	Usage in DESCRIBE and PREPARE INTO	Usage in FETCH, OPEN, EXECUTE, and CALL
# <i>stem</i> .SQLD	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. For DESCRIBE PROCEDURE, the number of result sets returned by the stored procedure. Contains a 0 if no result sets are returned.	The number of host variables that are used by the SQL statement.
# Each SQLDA contains <i>stem</i> .SQLD of the following variables. Therefore, $1 \leq n \leq \textit{stem}.\textit{SQLD}$. 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</i> .SQLTYPE	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 523.	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 523.
# <i>stem.n</i> .SQLLEN	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 523 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 523 for a description of allowable values.
# <i>stem.n</i> .SQLLEN.SQLPRECISION	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</i> .SQLLEN.SQLSCALE	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</i> .SQLCCSID	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</i> .SQLLOCATOR	For DESCRIBE PROCEDURE, the value of a result set locator.	Not used.
# <i>stem.n</i> .SQLDATA	Not used.	Before EXECUTE or OPEN, contains the value of an input host variable. The application must supply this value. After FETCH, contains the values of an output host variable.

SQLDA

Table 39 (Page 2 of 2). Fields of a REXX SQLDA

# Variable name	Usage in DESCRIBE and PREPARE INTO	Usage in FETCH, OPEN, EXECUTE, and CALL
# <i>stem.n</i> .SQLIND	Not used.	Before EXECUTE or OPEN, contains a negative number to indicate that the input host variable in <i>stem.n</i> .SQLDATA is null. The application must supply this value.
#		After FETCH, contains a negative number if the value of the output host variable in <i>stem.n</i> .SQLDATA is null.
#		
#		
#		
#		
# <i>stem.n</i> .SQLNAME	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 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.

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 DSNDB06. ...	TABLE SYSIBM. ...	Page	INDEX SYSIBM. ...	INDEX FIELDS	
SYSCOPY	SYSCOPY	554	DSNUCH01	DBNAME.TSNAME.START_RBA.1 TIMESTAMP1	
			DSNUCX01	DSNAME	
SYSDBASE	SYSCOLAUTH	546			
	SYSCOLUMNS	550	DSNDCX01	TBCREATOR.TBNAME.NAME	
	SYSFIELDS	565			
	SYSFOREIGNKEYS	566			
	SYSINDEXES		567	DSNDXX01	CREATOR.NAME
				DSNDXX02	DBNAME.INDEXSPACE
				DSNDXX03	TBCREATOR.TBNAME.CREATOR. NAME
	SYSINDEXPART	570	DSNDRX01	IXCREATOR.IXNAME.PARTITION	
	SYSKEYS	573	DSNDKX01	IXCREATOR.IXNAME.COLNAME	
	SYSRELS	593	DSNDLX01	REFTBCREATOR.REFTBNAME	
	SYSSYNONYMS	599	DSNDYX01	CREATOR.NAME	
	SYSTABAUTH		600	DSNATX01	GRANTOR
				DSNATX02	GRANTEE.TCREATOR.TTNAME. GRANTEETYPE.UPDATECOLS. ALTERAUTH.DELETEAUTH. INDEXAUTH.INSERTAUTH. SELECTAUTH.UPDATEAUTH. CAPTUREAUTH.REFERENCESAUTH. REFCOLS
DSNATX03				GRANTEE.GRANTEETYPE.COLLID CONOKEN	
SYSTABLEPART	603	DSNDPX01	DBNAME.TSNAME.PARTITION		
SYSTABLES		606	DSNDTX01	CREATOR.NAME	
			DSNDTX02	DBID.OBID.CREATOR.NAME	
SYSTABLESPACE	610	DSNDSX01	DBNAME.NAME		
SYSDBAUT	SYSDATABASE	557	DSNDDH01	NAME	
			DSNDDX02	GROUP_MEMBER	
	SYSDBAUTH	559	DSNADH01	GRANTEE.NAME	
			DSNADX01	GRANTOR.NAME	
SYSDDF	IPNAMES	537	DSNFPX01	LINKNAME	
	LOCATIONS	538	DSNFCX01	LOCATION	
	LULIST	539	DSNFLX01	LINKNAME.LUNAME	
			DSNFLX02	LUNAME	
	LUMODES	540	DSNFMX01	LUNAME.MODENAME	
	LUNAMES	541	DSNFX01	LUNAME	
	MODESELECT	543	DSNFDX01	LUNAME.AUTHID1.PLANNAME1	
	USERNAMES	620	DSNFEX01	TYPE.AUTHID1.LINKNAME1	

TABLE SPACE DSNDB06. ...	TABLE SYSIBM. ...	Page	INDEX SYSIBM. ...	INDEX FIELDS
SYSGPAUT	SYSRESAUTH	594	DSNAGH01	GRANTEE.QUALIFIER. NAME.OBTYPE
			DSNAGX01	GRANTOR.QUALIFIER. NAME.OBTYPE
SYSGROUP	SYSSTOGROUP	597	DSNSSH01	NAME
	SYSVOLUMES	619		
SYSPKAGE	SYSPACKAGE	574	DSNKX01	LOCATION.COLLID.NAME. VERSION
			DSNKX02	LOCATION.COLLID.NAME. CONTOKEN
			DSNKAX01	GRANTOR.LOCATION.COLLID.NAME
	SYSPACKAUTH	578	DSNKAX02	GRANTEE.LOCATION.COLLID. NAME.BINDAUTH.COPYAUTH. EXECUTEAUTH
			DSNKAX03	LOCATION.COLLID.NAME
			DSNKDX01	DLOCATION.DCOLLID.DNAME. DCONTOKEN
	SYSPACKDEP	579	DSNKDX02	BQUALIFIER.BNAME.BTYPE
			DSNKCX01	PROCEDURE.AUTHID ¹ .LUNAME ¹
	SYSPACKLIST	580	DSNKLX01	LOCATION.COLLID.NAME
			DSNKLX02	PLANNAME.SEQNO.LOCATION. COLLID.NAME
SYSPACKSTMT	581	DSNKSX01	LOCATION.COLLID.NAME. CONTOKEN.SEQNO	
SYSPKSYSTEM	583	DSNKYX01	LOCATION.COLLID.NAME. CONTOKEN.SYSTEM.ENABLE	
SYSPLSYSTEM	589	DSNKPX01	NAME.SYSTEM.ENABLE	
SYSPLAN	SYSDBRM	562		
	SYSPLAN	584	DSNPPH01	NAME
	SYSPLANAUTH	587	DSNAPH01	GRANTEE.NAME.EXECUTEAUTH
			DSNAPX01	GRANTOR
	SYSPLANDEP	588	DSNGGX01	BCREATOR.BNAME.BTYPE
SYSSTMT	595			
SYSSTATS	SYSCOLDIST	547	DSNTNX01	TBOWNER.TBNAME.NAME
	SYSCOLDISTSTATS	548	DSNTPX01	TBOWNER.TBNAME.NAME PARTITION
	SYSCOLSTATS	549	DSNTCX01	TBOWNER.TBNAME.NAME PARTITION
	SYSINDEXSTATS	572	DSNTXX01	OWNER.NAME.PARTITION
	SYSTABSTATS	613	DSNTTX01	OWNER.NAME.PARTITION
SYSSTR	SYSSTRINGS	598	DSNSSX01	OUTCCSID.INCCSID.IBMREQD
	SYSCHECKS	545	DSNSCX01	TBOWNER.TBNAME.CHECKNAME

DB2 Catalog Tables

TABLE SPACE DSNDB06. ...	TABLE SYSIBM. ...	Page	INDEX SYSIBM. ...	INDEX FIELDS
	SYSCHECKDEP	544	DSNSDX01	TBOWNER.TBNAME.CHECKNAME COLNAME
SYSUSER	SYSUSERAUTH	614	DSNAUH01	GRANTEE
			DSNAUX02	GRANTOR
SYSVIEWS	SYSVIEWDEP	617	DSNGGX02	BCREATOR.BNAME.BTYPE
	SYSVIEWS	618	DSNVVX01	CREATOR.NAME.SEQNO

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	Only these clauses are allowed: CONVERT TO ³³ CLOSE FREEPAGE GBPCACHE PCTFREE PIECESIZE
ALTER TABLE		The only clause allowed is DATA CAPTURE CHANGES.
ALTER TABLESPACE		Only these clauses are allowed: CLOSE FREEPAGE GBPCACHE LOCKMAX PCTFREE
CREATE INDEX	User-created	All clauses are allowed, except for: CLOSE YES CLUSTER UNIQUE DEFER YES (only on tables SYSINDEXES, SYSINDEXPART, and SYSKEYS) The only value allowed for BUFFERPOOL is BP0. You can create up to 100 indexes on the catalog.

#

³³ In a data sharing environment, the only allowable value to specify for SUBPAGES on the COVERT TO clause is 1.

SQL Statement	Index	Allowable Clauses and Usage Notes
ALTER INDEX	User-created	All clauses are allowed, except for: BUFFERPOOL
DROP INDEX	User-created	All clauses are allowed.

When using the clause CONVERT TO TYPE 1 to change the type of IBM-defined indexes, regardless of the value you implicitly or explicitly specify for SUBPAGES, the number of subpages is 1 for the following indexes:

DSNAPH01	DSNDTX02	DSNFPX01	DSNKSX01
DSNATX01	DSNDXX01	DSNGGX01	DSNSCX01
DSNATX02	DSNDXX02	DSNGGX02	DSNSDX01
DSNATX03	DSNDYX01	DSNKAX01	DSNTCX01
DSNDCX01	DSNFCX01	DSNKAX02	DSNTNX01
DSNDKX01	DSNFDX01	DSNKAX03	DSNTPX01
DSNDLX01	DSNFEX01	DSNKDX01	DSNTTX01
DSNDPX01	DSNFLX01	DSNKDX02	DSNTXX01
DSNDRX01	DSNFLX02	DSNKKX01	DSNVTH01
DSNDSX01	DSNFMX01	DSNKKX02	DSNVVX01
DSNDTX01	DSNFX01	DSNKLX02	

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.SYSPKAGE
- DSNDB06.SYSSTR
- DSNDB06.SYSSTATS
- DSNDB06.SYSUSER
- DSNDB01.SCT02
- DSNDB01.SPT01

For details on running REORG TABLESPACE, see *Utility Guide and Reference*.

New and Changed Catalog Tables

Descriptions of the following catalog tables have been added:

- SYSIBM.IPNAMES
- SYSIBM.LOCATIONS
- SYSIBM.LULIST
- SYSIBM.LUNAMES
- SYSIBM.LUMODES
- SYSIBM.MODESELECT
- SYSIBM.SYSDUMMY1
- SYSIBM.USERNAMES

The following tables have new or revised columns, column values, or column descriptions to support the new function in DB2 Version 5:

Table Name	New Column	Revised Column
SYSCOLAUTH	PRIVILEGE GRANTEDTS	
SYSCOLDIST	TYPE CARDF COLGROUPCOLNO NUMCOLUMNS FREQUENCYF	FREQUENCY NAME
SYSCOLDISTSTATS	TYPE CARDF COLGROUPCOLNO NUMCOLUMNS FREQUENCYF	FREQUENCY NAME
SYSCOLUMNS	COLCARDF	COLCARD DEFAULT DEFAULTVALUE
SYSCOPY		STYPE
SYSDATABASE	CREATEDTS ALTEREDTS ENCODING_SCHEME SBCS_CCSID DBCS_CCSID MIXED_CCSID	
SYSDBAUTH	GRANTEDTS	
SYSDBRM	PRECOMPTS	HOSTLANG
SYSINDEXES	FIRSTKEYCARDF FULLKEYCARDF CREATEDTS ALTEREDTS PIECESIZE	UNIQUERULE FIRSTKEYCARD FULLKEYCARD
SYSINDEXPART	FAROFFPOSF NEAROFFPOSF CARDF	CARD LIMITKEY FAROFFPOS NEAROFFPOS
SYSPACKAGE	REOPTVAR DEFERPREPARE KEEPDYNAMIC	VALID HOSTLANG REMOTE IBMREQD
SYSPACKDEP		BTYPE
SYSPACKSTMT	STATUS	

Table Name	New Column	Revised Column
SYSPLAN	BOUNDTS REOPTVAR KEEPDYNAMIC	VALID IBMREQD
SYSPLANAUTH	GRANTEDS	
SYSPLANDEP		BTYPE
SYS PROCEDURES	RESULT_SETS WLM_ENV PGM_TYPE EXTERNAL_SECURITY COMMIT_ON_RETURN	LUNAME
SYSRELS		DELETERULE IXNAME IXOWNER
SYSRESAUTH	GRANTEDTS	
SYSSTMT	STATUS	
SYSSTOGROUP	STATSTIME CREATEDTS ALTEREDTS	
SYSSTRINGS		TRANSTYPE
SYSSYNONYMS	CREATEDTS	
SYSTABAUTH	REFCOLS GRANTEDTS	UPDATECOLS
SYSTABLEPART	CHECKRID5B	LIMITKEY CHECKRID
SYSTABLES	CARDF CHECKRID5B ENCODING_SCHEME	TYPE DBNAME TSNAME DBID OBID CARD NPAGES PCTPAGES IBMREQD PARENTS CHILDREN KEYCOLUMNS STATUS CHECKRID AUDITING DATACAPTURE RBA1 RBA2 PCTROWCOMP STATSTIME CHECKS
SYSTABLESPACE	TYPE CREATEDTS ALTEREDTS ENCODING_SCHEME SBCS_CCSID DBCS_CCSID MIXED_CCSID MAXROWS LOCKPART	IBMREQD
SYSUSERAUTH	GRANTEDTS CREATETMTABAUTH	
SYSVIEWS		CHECK IBMREQD

DB2 Catalog Tables

	Table Name	New Column	Revised Column
	SYSVTREE		IBMREQD

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) 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) NOT NULL WITH DEFAULT 'A'	<p>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:</p> <p>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.</p> <p>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> <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.</p> <p>P The option is "password." Outbound connection requests contain an authorization ID and a password. The password is obtained from the SYSIBM.USERNAMES table.</p> <p> The USERNAMES column must specify "O."</p>	G
USERNAMES	CHAR(1) NOT NULL WITH DEFAULT	<p>This column controls outbound authorization ID translation. Outbound translation is performed when an authorization ID is sent by DB2 to a remote server.</p> <p>O An outbound ID is subject to translation. Rows in the SYSIBM.USERNAMES table are used to perform ID translation.</p> <p> No translation or "come from" checking is performed on inbound IDs.</p> <p>blank No translation occurs.</p>	G
IBMREQD	CHAR(1) NOT NULL WITH DEFAULT 'N'	<p>Whether the row came from the basic machine-readable material (MRM) tape:</p> <p>N No Y Yes</p>	G
IPADDR	VARCHAR(254) NOT NULL WITH DEFAULT	<p>This column contains the IP address or domain name of a remote TCP/IP host. The IPADDR column must be specified as follows:</p> <ul style="list-style-type: none"> 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. 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. 	G

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) 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) 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"> 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. 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. 	G
IBMREQD	CHAR(1) NOT NULL WITH DEFAULT 'N'	Whether the row came from the basic machine-readable material (MRM) tape: <p>N No</p> <p>Y Yes</p>	G
PORT	CHAR(32) NOT NULL WITH DEFAULT	TCP/IP is used for outbound DRDA connections when the following statement is true: <ul style="list-style-type: none"> A row exists in SYSIBM.IPNAMES, where the LINKNAME column matches the value specified in the SYSIBM.LOCATIONS LINKNAME column. <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"> If PORT is blank, the default DRDA port (446) is used. If PORT is nonblank, the value specified for PORT can take one of two forms: <ul style="list-style-type: none"> 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. 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. 	G
TPN	VARCHAR(64) NOT NULL WITH 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. <p>For an SQL/DS server, TPN should contain the resource ID of the SQL/DS machine.</p>	G

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) 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) 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) NOT NULL WITH DEFAULT 'N'	Whether the row came from the basic machine-readable material (MRM) tape: N No Y Yes	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) NOT NULL	LU name of the other system involved in the CNOS processing.	G
MODENAME	CHAR(8) NOT NULL	Name of a logon mode description in the VTAM logon mode table.	G
# CONVLIMIT #	SMALLINT 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) NOT NULL WITH DEFAULT 'N'	Whether the row came from the basic machine-readable material (MRM) tape: N No Y Yes	G

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) 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. 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	CHAR(8) NOT NULL WITH DEFAULT	Mode used to establish inter-system conversations. A blank indicates the default mode IBMDB2LM (DB2 private protocol access).	G
SECURITY_IN	CHAR(1) NOT NULL WITH DEFAULT 'A'	This column defines the security options that are accepted by this DB2 when an SNA client connects to DB2: V 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 DCE security ticket. A The option is "already verified." A request does not need a password, although a password is checked if it is sent. 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 DCE security ticket. 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) NOT NULL WITH 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: 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 server's LU name is used as the RACF PassTicket application name. 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. P 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. The USERNAMES column must specify 'B' or 'O'.	G

SYSIBM.LUNAMES

Column Name	Data Type	Description	Use
ENCRYPTPSWDS	CHAR(1) NOT NULL WITH DEFAULT 'N'	<p>This column only applies to DB2 for OS/390 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 ENCRYPTPSWDS='Y' option.</p> <p>N No, passwords are not in internal RACF encrypted format. This is the default.</p> <p>Y 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) NOT NULL WITH DEFAULT 'N'	<p>Whether to use the SYSIBM.MODESELECT table:</p> <p>N Use default modes: IBMDB2LM (for DB2 private protocol) and IBMRDB (for DRDA).</p> <p>Y Searches SYSIBM.MODESELECT for appropriate mode name.</p>	G
USERNAMES	CHAR(1) NOT NULL WITH DEFAULT	<p>This column controls inbound and outbound authorization ID translation, as well as "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>I An inbound ID is subject to translation and "come from" checking. No translation is performed on outbound IDs.</p> <p>O No translation or "come from" checking is performed on inbound IDs. An outbound ID is subject to translation.</p> <p>B An inbound ID is subject to translation and "come from" checking. An outbound ID is subject to translation.</p> <p>blank No translation occurs.</p>	G
GENERIC	CHAR(1) NOT NULL WITH 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>N The real VTAM LU name of this DB2 subsystem</p> <p>Y The VTAM generic LU name of this DB2 subsystem</p>	G
IBMREQD	CHAR(1) NOT NULL WITH DEFAULT 'N'	<p>Whether the row came from the basic machine-readable material (MRM) tape:</p> <p>N No</p> <p>Y Yes</p>	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) NOT NULL 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) NOT NULL 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) NOT NULL	LU name associated with the SQL request.	G
MODENAME	CHAR(8) 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
IBMREQD	CHAR(1) NOT NULL WITH DEFAULT 'N'	Whether the row came from the basic machine-readable material (MRM) tape: N No Y Yes	G

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) NOT NULL	The authorization ID of the owner of the table on which the table check constraint is defined.	G
TBNAME	VARCHAR(18) NOT NULL	The name of the table on which the check constraint is defined.	G
CHECKNAME	VARCHAR(128) NOT NULL	The name of the check constraint.	G
COLNAME	VARCHAR(18) NOT NULL	The name of the column referenced by the table check constraint.	G
IBMREQD	CHAR(1) NOT NULL	Whether the row came from the basic machine readable material (MRM) tape: N No Y Yes	G

SYSIBM.SYSCHECKS Table

Contains one row for each table check constraint.

Column Name	Data Type	Description	Use
TBOWNER	CHAR(8) NOT NULL	The authorization ID of the owner of the table on which the table check constraint is defined.	G
CREATOR	CHAR(8) NOT NULL	The authorization ID of the creator of the table check constraint.	G
DBID	SMALLINT NOT NULL	The internal identifier of the database for the table check constraint.	S
OBID	SMALLINT NOT NULL	The internal identifier of the table check constraint.	S
TIMESTAMP	TIMESTAMP NOT NULL	The time when the table check constraint was created.	G
RBA	CHAR(6) FOR BIT DATA NOT NULL	The log RBA when the table check constraint was created.	G
IBMREQD	CHAR(1) NOT NULL	Whether the row came from the basic machine readable material (MRM) tape: N No Y Yes	G
TBNAME	VARCHAR(18) NOT NULL	The name of the table on which the check constraint is defined	G
CHECKNAME	VARCHAR(128) NOT NULL	The table check constraint name.	G
CHECKCONDITION	VARCHAR(3800) NOT NULL	The text of the table check constraint.	G

SYSIBM.SYSCOLAUTH Table

Records the UPDATE or REFERENCES privileges held by users on individual columns of a table or view.

Column Name	Data Type	Description	Use
GRANTOR	CHAR(8) NOT NULL	Authorization ID of the user who granted the privileges. Could also be PUBLIC or PUBLIC followed by an asterisk ³⁴ .	G
GRANTEE	CHAR(8) 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) NOT NULL	Type of grantee: blank GRANTEE is an authorization ID P GRANTEE is an application plan or a package. It is a package if COLLID is not blank.	G
CREATOR	CHAR(8) NOT NULL	The authorization ID of the owner of the table or view on which the update privilege is held.	G
TNAME	VARCHAR(18) NOT NULL	The name of the table or view.	G
	CHAR(12) NOT NULL	Internal use only	I
DATEGRANTED	CHAR(6) NOT NULL	Date the privilege was granted, in the form <i>yymmdd</i> .	G
TIMEGRANTED	CHAR(8) NOT NULL	Time the privilege was granted, in the form <i>hhmmssst</i> .	G
COLNAME	VARCHAR(18) NOT NULL	Name of the column to which the UPDATE privilege applies.	G
IBMREQD	CHAR(1) NOT NULL	Whether the row came from the basic machine readable material (MRM) tape: N No Y Yes	G
	CHAR(16) NOT NULL WITH DEFAULT	Not used	N
COLLID	CHAR(18) NOT NULL WITH DEFAULT	If GRANTEE is a package, its collection name. Otherwise, blank.	G
CONTOKEN	CHAR(8) NOT NULL WITH DEFAULT	If GRANTEE is a package, the consistency token of the DBRM from which the package was derived. Otherwise, blank.	S
PRIVILEGE	CHAR(1) NOT NULL WITH DEFAULT	Indicates which privilege this row describes: R Row pertains to the REFERENCES privilege. blank Row pertains to the UPDATE privilege.	G
GRANTEDTS	TIMESTAMP NOT NULL WITH DEFAULT	Time when the GRANT statement was executed.	G

³⁴ PUBLIC followed by an asterisk (PUBLIC*) denotes PUBLIC AT ALL LOCATIONS. For the conditions where GRANTOR can be PUBLIC or PUBLIC*, see Section 3 (Volume 1) of *Administration Guide*.

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 NOT NULL	Not used	N
STATSTIME	TIMESTAMP NOT NULL WITH DEFAULT	If RUNSTATS updated the statistics, the date and time when the last invocation of RUNSTATS updated the statistics.	G
IBMREQD	CHAR(1) NOT NULL	Whether the row came from the basic machine-readable material (MRM) tape. N No Y Yes	G
TBOWNER	CHAR(8) NOT NULL	Authorization ID of the owner of the table containing the column.	G
TBNAME	VARCHAR(18) NOT NULL	Name of the table that contains the column.	G
NAME	VARCHAR(18) 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) NOT NULL FOR BIT DATA	Contains the data of a frequently occurring value. If the value has a non-character data type, the data might not be printable.	S
TYPE	CHAR(1) NOT NULL WITH DEFAULT 'F'	The type of statistics gathered: C Cardinality F Frequent Value	G
CARDF	FLOAT NOT NULL WITH DEFAULT -1	The number of distinct values for the column group. This number is valid only for TYPE C statistics.	S
COLGROUPCOLNO	VARCHAR(254) NOT NULL WITH 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 NOT NULL WITH DEFAULT 1	Identifies the number of columns associated with the statistics.	G
FREQUENCYF	FLOAT NOT NULL WITH 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%.	G

SYSIBM.SYSCOLDISTSTATS Table

Contains zero or more rows per partition for the first key column of a partitioned index. Rows are inserted when RUNSTATS scans index partitions of the partitioned index. No row is inserted if the index is a nonpartitioned index. Rows in this table can be inserted, updated, and deleted.

Column Name	Data Type	Description	Use
	SMALLINT NOT NULL	Not used	N
STATSTIME	TIMESTAMP NOT NULL WITH DEFAULT	If RUNSTATS updated the statistics, the date and time when the last invocation of RUNSTATS updated the statistics.	G
IBMREQD	CHAR(1) NOT NULL	Whether the row came from the basic machine-readable material (MRM) tape. N No Y Yes	G
PARTITION	SMALLINT NOT NULL	Partition number for the table space containing the table in which the column is defined.	G
TBOWNER	CHAR(8) NOT NULL	Authorization ID of the owner of the table that contains the column.	G
TBNAME	VARCHAR(18) NOT NULL	Name of the table that contains the column.	G
NAME	VARCHAR(18) 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) NOT NULL FOR BIT DATA	Contains the data of a frequently occurring value. If the value has a non-character data type, the data may not be printable.	S
TYPE	CHAR(1) NOT NULL WITH DEFAULT 'F'	The type of statistics gathered: C Cardinality F Frequent Value	G
CARDF	FLOAT NOT NULL WITH DEFAULT -1	The number of distinct values for the column group. This number is valid only for TYPE C statistics.	S
COLGROUPCOLNO	VARCHAR(254) NOT NULL WITH 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 NOT NULL WITH DEFAULT 1	Identifies the number of columns associated with the statistics.	G
FREQUENCYF	FLOAT NOT NULL WITH 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%.	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.

Column Name	Data Type	Description	Use
HIGHKEY	CHAR(8) NOT NULL FOR BIT DATA	Highest value of the column within the partition. Blank if statistics have not been gathered. If the column has a non-character data type, the data might not be printable. This is an updatable column.	S
HIGH2KEY	CHAR(8) NOT NULL FOR BIT DATA	Second highest value of the column within the partition. Blank if statistics have not been gathered. If the column has a non-character data type, the data might not be printable.	S
LOWKEY	CHAR(8) NOT NULL FOR BIT DATA	Lowest value of the column within the partition. Blank if statistics have not been gathered. If the column has a non-character data type, the data might not be printable. This is an updatable column.	S
LOW2KEY	CHAR(8) NOT NULL FOR BIT DATA	Second lowest value of the column within the partition. Blank if statistics have not been gathered. If the column has a non-character data type, the data might not be printable. This is an updatable column.	S
	INTEGER NOT NULL	Number of distinct column values in the partition. This is an updatable column.	S
STATSTIME	TIMESTAMP 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', RUNSTATS should be run to update the statistics before they are used. This is an updatable column.	G
#			
#			
#			
IBMREQD	CHAR(1) NOT NULL	Whether the row came from the basic machine-readable material (MRM) tape. N No Y Yes	G
PARTITION	SMALLINT NOT NULL	Partition number for the table space containing the table in which the column is defined.	G
TBOWNER	CHAR(8) NOT NULL	Authorization ID of the owner of the table that contains the column.	G
TBNAME	VARCHAR(18) NOT NULL	Name of the table that contains the column.	G
NAME	VARCHAR(18) NOT NULL	Name of the column.	G
COLCARDATA	VARCHAR(1000) NOT NULL FOR BIT DATA	Internal use only	I

SYSIBM.SYSCOLUMNS Table

Contains one row for every column of each table and view.

Column Name	Data Type	Description	Use
NAME	VARCHAR(18) NOT NULL	Name of the column.	G
TBNAME	VARCHAR(18) NOT NULL	Name of the table or view which contains the column.	G
TBCREATOR	CHAR(8) NOT NULL	Authorization ID of the owner of the table or view that contains the column.	G
COLNO	SMALLINT NOT NULL	Numeric place of the column in the table or view; for example 4 (out of 10). 0 in an additional row if the definition of the table is incomplete (all required unique indexes have not been created).	G
COLTYPE	CHAR(8) NOT NULL	The type of the column specified in the definition of the column: INTEGER Large integer SMALLINT Small integer FLOAT Floating-point CHAR Fixed-length character string VARCHAR Varying-length character string LONGVAR Varying-length character string DECIMAL Decimal GRAPHIC Fixed-length graphic string VARG Varying-length graphic string LONGVARG Varying-length graphic string DATE Date TIME Time TIMESTMP Timestamp Whether a column described as VARCHAR, LONGVAR, VARG, or LONGVARG is a long string column or not depends on its length attribute.	G
LENGTH	SMALLINT NOT NULL	The length attribute of the column or, in the case of a decimal column, its precision. The number does not include the internal prefixes used to record actual length and null state, where applicable. INTEGER 4 SMALLINT 2 FLOAT 4 or 8 CHAR Length of string VARCHAR Maximum length of string LONGVAR Maximum length of string DECIMAL Precision of number GRAPHIC Number of DBCS characters VARG Maximum number of DBCS characters LONGVARG Maximum number of DBCS characters DATE 4 TIME 3 TIMESTMP 10	G
SCALE	SMALLINT NOT NULL	Scale of decimal data. Zero if not a decimal column.	G
NULLS	CHAR(1) NOT NULL	Whether the column can contain null values: N No Y Yes 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 it is referenced in an outer select list.	G
	INTEGER NOT NULL	Not used	N

Column Name	Data Type	Description	Use
HIGH2KEY	CHAR(8) NOT NULL FOR BIT DATA	Second highest value of the column. Blank if statistics have not been gathered. If the column has a non-character data type, the data might not be printable. This is an updatable column.	S
LOW2KEY	CHAR(8) NOT NULL FOR BIT DATA	Second lowest value of the column. Blank if statistics have not been gathered. If the column has a non-character data type, the data might not be printable. This is an updatable column.	S
UPDATES	CHAR(1) NOT NULL	Whether the column can be updated: N No Y Yes The value is N if the column is: <ul style="list-style-type: none"> • Part of the key of a partitioned index • Derived from a function or expression. The value can be Y for columns of a read-only view.	G
IBMREQD	CHAR(1) NOT NULL	Whether the row came from the basic machine-readable material (MRM) tape: N No Y Yes	G
REMARKS	VARCHAR(254) NOT NULL	A character string provided by the user with the COMMENT ON statement.	G

SYSIBM.SYSCOLUMNS

Column Name	Data Type	Description	Use																												
DEFAULT	CHAR(1) NOT NULL	<p>Default indicator:</p> <p>N The column has no default value.</p> <p>Y If the NULLS column is Y, the column has a default value of null.</p> <p>If the NULLS column is N, the default value depends on the data type of the column.</p> <table border="1"> <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> <p>B The default value depends on the data type of the column.</p> <table border="1"> <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> <p>1 The column has a default value that is the string constant found in the DEFAULTVALUE column of this table row.</p> <p>2 The column has a default value that is the floating-point constant found in the DEFAULTVALUE column of this table row.</p> <p>3 The column has a default value that is the decimal constant found in the DEFAULTVALUE column of this table row.</p> <p>4 The column has a default value that is the integer constant found in the DEFAULTVALUE column of this table row.</p> <p>5 The column has a default value that is the hex string found in the DEFAULTVALUE column of this table row.</p> <p>S The column has a default value that is the value of the SQLID of the process at the time a default value is used.</p> <p>U The column has a default value that is the value of the USER special register at the time a default value is used.</p>	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	G
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																														
KEYSEQ	SMALLINT NOT NULL	The column's numeric position within the table's primary key. 0 if it is not part of a primary key.	G																												
FOREIGNKEY	CHAR(1) NOT NULL	<p>Applies to character columns only, where it indicates the subtype of the data. A value of B indicates BIT data, and if value of the field MIXED DATA on installation panel DSNTIPF is:</p> <ul style="list-style-type: none"> • NO, any other value indicates SBCS data • YES, an S indicates SBCS and any other value indicates MIXED. <p>This is an updatable column.</p>	G																												

|

#

Column Name	Data Type	Description	Use	
FLDPROC	CHAR(1) NOT NULL	Whether the column has a field procedure: N No Y Yes This column is blank for views.	G	
LABEL	VARCHAR(30) NOT NULL	The column label provided by the user with a LABEL ON statement; otherwise it is an empty string.	G	
STATSTIME	TIMESTAMP NOT NULL WITH 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'. If the value is '0001-01-02.00.00.00.000000', RUNSTATS should be run to update the statistics before they are used. This is an updatable column.	G	
# # # # # #	DEFAULTVALUE	VARCHAR(512) NOT NULL WITH DEFAULT When the DEFAULT column is 1, 2, 3, 4, or 5, this field contains the default value of the column. If the default value is a string constant or a hexadecimal constant (DEFAULT is 1 or 5, respectively), the value is stored without delimiters, except for a graphic string constant which will be enclosed by the shift-out and shift-in characters. 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. 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 NOT NULL WITH DEFAULT -1	Estimated number of distinct values in the column. The value is -1 if statistics have not been gathered. This is an updatable column.	S

SYSIBM.SYSCOPY Table

Contains information needed for recovery.

Column Name	Data Type	Description	Use
DBNAME	CHAR(8) NOT NULL	Name of the database.	G
TSNAME	CHAR(8) NOT NULL	Name of the table space.	G
DSNUM	INTEGER 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.	G
# ICTYPE	CHAR(1) NOT NULL	Operation type: A ALTER TABLE F COPY FULL YES I COPY FULL NO P RECOVER TOCOPY or RECOVER TORBA (partial recovery point) Q QUIESCE R LOAD REPLACE LOG(YES) S LOAD REPLACE LOG(NO) W REORG LOG(NO) X REORG LOG(YES) Y LOAD LOG(NO) Z LOAD LOG(YES) T TERM UTILITY command (terminated utility)	G
ICDATE	CHAR(6) NOT NULL	Date of the entry in the form <i>yyymmdd</i> .	G
START_RBA	CHAR(6) NOT NULL FOR BIT DATA	A 48-bit positive integer containing the LRSN of a point in the DB2 recovery log. (The LRSN is the RBA in a non-data-sharing environment.) <ul style="list-style-type: none"> For ICTYPE I or F, the starting point for all updates since the image copy was taken For ICTYPE P, the point after the log-apply phase of point-in-time recovery For ICTYPE Q, the point after all data sets have been successfully quiesced For ICTYPE R or S, the end of the log before the start of the LOAD utility and before any data is changed For ICTYPE T, the end of the log when the utility is terminated For other values of ICTYPE, the end of the log before the start of the RELOAD phase of the LOAD or REORG utility. 	G
FILESEQNO	INTEGER NOT NULL	Tape file sequence number of the copy.	G
DEVTYPE	CHAR(8) NOT NULL	Device type the copy is on.	G
IBMREQD	CHAR(1) NOT NULL	Whether the row came from the basic machine-readable material (MRM) tape: N No Y Yes	G
DSNAME	CHAR(44) 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 in the form, <i>database-nametable-space-name</i> , or DSNAME is blank for any row migrated from a DB2 release prior to Version 5.	G

Column Name	Data Type	Description	Use
ICTIME	CHAR(6) 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.	G
SHRLEVEL	CHAR(1) NOT NULL	SHRLEVEL parameter on COPY (for ICTYPE F or I only): C Change R Reference blank Does not describe an image copy or was migrated from Version 1 Release 1 of DB2.	G
DSVOLSER	VARCHAR(1784) 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 NOT NULL WITH 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.	G
ICBACKUP	CHAR(2) NOT NULL WITH DEFAULT	Specifies the type of image copy contained in the data set: blank LOCALSITE primary copy (first data set named with COPYDDN) LB LOCALSITE backup copy (second data set named with COPYDDN) RP RECOVERYSITE primary copy (first data set named with RECOVERYDDN) RB RECOVERYSITE backup copy (second data set named with RECOVERYDDN)	G
ICUNIT	CHAR(1) NOT NULL WITH DEFAULT	Indicates the media that the image copy data set is stored on: D DASD T Tape blank Medium is neither tape nor DASD, or the image copy is from a DB2 release prior to Version 2 Release 3, or ICTYPE is neither "I" nor "F."	G
# STYPE #	CHAR(1) NOT NULL WITH DEFAULT	When ICTYPE=A, the length of a VARCHAR column in a table was increased, the value is V. When ICTYPE=F, the values are: C DFSMS concurrent copy blank DB2 image copy R LOAD REPLACE(YES) S LOAD REPLACE(NO) W REORG LOG(NO) X REORG LOG(YES) The MERGECOPY utility, when used to merge an embedded copy with subsequent incremental copies, will also produce a record that will contain ICTYPE=F and the STYPE of the original image copy (R, S, W, or X). 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=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: F COPY FULL YES I COPY FULL NO For other values of ICTYPE, the value is blank.	G

SYSIBM.SYSCOPY

Column Name	Data Type	Description	Use
PIT_RBA	CHAR(6) NOT NULL WITH DEFAULT FOR BIT DATA	<p>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'.</p> <p>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 state 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.</p>	G
GROUP_MEMBER	CHAR(8) NOT NULL WITH 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

SYSIBM.SYSDATABASE Table

Contains one row for each database, except for database DSNDB01.

Column Name	Data Type	Description	Use
NAME	CHAR(8) NOT NULL	Database name.	G
CREATOR	CHAR(8) NOT NULL	Authorization ID of the owner of the database.	G
STGROUP	CHAR(8) NOT NULL	Name of the default storage group of the database; blank for a system database.	G
BPOOL	CHAR(8) NOT NULL	Name of the default buffer pool of the database; blank for a system database.	G
DBID	SMALLINT NOT NULL	Internal identifier of the database.	S
IBMREQD	CHAR(1) NOT NULL	Whether the row came from the basic machine-readable material (MRM) tape: N No Y Yes E V2R3 dependency indicator; not from MRM tape G V4 dependency indicator; not from MRM tape	G
CREATEDBY	CHAR(8) NOT NULL WITH DEFAULT	Primary authorization ID of the user who created the database.	G
ROSHARE	CHAR(1) NOT NULL WITH DEFAULT	Whether this database is shared, through shared read-only data, with other DB2 subsystems. Value can be: blank Not shared O Shared. Local DB2 subsystem is the owner. R Shared. Local DB2 subsystem is a read-only user.	G
TIMESTAMP	TIMESTAMP NOT NULL WITH DEFAULT	The time the database became shared on the owning system through shared read-only data. If the database is not shared, the value is 0001-01-01-00.00.000000	G
TYPE	CHAR(1) NOT NULL WITH DEFAULT	Type of database. blank Database is not a work file database. W Database is 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) NOT NULL WITH 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 NOT NULL WITH DEFAULT	Time when the CREATE statement was executed for the database. For DSNDB04 and DSNDB06, the value is '1985-04-01.00.00.000000'.	G
ALTEREDTS	TIMESTAMP NOT NULL WITH 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) NOT NULL WITH DEFAULT 'E'	Default encoding scheme for the database. E EBCDIC A ASCII blank For DSNDB04 and work file databases.	G
SBCS_CCSID	INTEGER NOT NULL WITH DEFAULT	Default SBCS CCSID for the database. For databases created in a DB2 release prior to Version 5, the value is 0.	G

SYSIBM.SYSDATABASE

Column Name	Data Type	Description	Use
DBCS_CC SID	INTEGER NOT NULL WITH DEFAULT	Default DBCS CCSID for the database. For databases created in a DB2 release prior to Version 5, the value is 0.	G
MIXED_CC SID	INTEGER NOT NULL WITH DEFAULT	Default mixed CCSID for the database. For databases created in a DB2 release prior to Version 5, the value is 0.	G

SYSIBM.SYSDBAUTH Table

Records the privileges held by users over databases.

Column Name	Data Type	Description	Use	
GRANTOR	CHAR(8) NOT NULL	Authorization ID of the user who granted the privileges. Could also be PUBLIC or PUBLIC followed by an asterisk. ³⁵	G	
GRANTEE	CHAR(8) NOT NULL	Application ID of the user who holds the privilege. Could also be PUBLIC for a grant to PUBLIC.	G	
NAME	CHAR(8) NOT NULL	Database name.	G	
	CHAR(12) NOT NULL	Internal use only	I	
DATEGRANTED	CHAR(6) NOT NULL	Date the privileges were granted; in the form <i>yyymmdd</i> .	G	
TIMEGRANTED	CHAR(8) NOT NULL	Time the privileges were granted; in the form <i>hhmmssst</i> .	G	
	CHAR(1) NOT NULL	Not used	N	
AUTHHOWGOT	CHAR(1) 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.	G	
		blank		Not applicable
		C		DBCTL
		D		DBADM
		L		SYSCTRL
		M		DBMAINT
CREATETABAUTH	CHAR(1) NOT NULL	Whether the GRANTEE can create tables within the database:	G	
		blank		Privilege is not held
		G		Privilege held with the GRANT option
		Y		Privilege is held without the GRANT option
CREATETSAUTH	CHAR(1) NOT NULL	Whether the GRANTEE can create table spaces within the database:	G	
		blank		Privilege is not held
		G		Privilege held with the GRANT option
		Y		Privilege is held without the GRANT option
DBADMAUTH	CHAR(1) NOT NULL	Whether the GRANTEE has DBADM authority over the database:	G	
		blank		Privilege is not held
		G		Privilege held with the GRANT option
		Y		Privilege is held without the GRANT option
DBCTRLAUTH	CHAR(1) NOT NULL	Whether the GRANTEE has DBCTRL authority over the database:	G	
		blank		Privilege is not held
		G		Privilege held with the GRANT option
		Y		Privilege is held without the GRANT option
DBMAINTAUTH	CHAR(1) NOT NULL	Whether the GRANTEE has DBMAINT authority over the database:	G	
		blank		Privilege is not held
		G		Privilege held with the GRANT option
		Y		Privilege is held without the GRANT option

³⁵ PUBLIC followed by an asterisk (PUBLIC*) denotes PUBLIC AT ALL LOCATIONS. For the conditions where GRANTOR can be PUBLIC or PUBLIC*, see Section 3 (Volume 1) of *Administration Guide*.

SYSIBM.SYSDBAUTH

Column Name	Data Type	Description	Use
DISPLAYDBAUTH	CHAR(1) NOT NULL	Whether the GRANTEE can issue the DISPLAY command for the database: blank Privilege is not held G Privilege held with the GRANT option Y Privilege is held without the GRANT option	G
DROPAUTH	CHAR(1) NOT NULL	Whether the GRANTEE can issue the ALTER DATABASE and DROP DATABASE statement: blank Privilege is not held G Privilege held with the GRANT option Y Privilege is held without the GRANT option	G
IMAGCOPYAUTH	CHAR(1) NOT NULL	Whether the GRANTEE can use the COPY, MERGECOPY, MODIFY, and QUIESCE utilities on the database: blank Privilege is not held G Privilege held with the GRANT option Y Privilege is held without the GRANT option	G
LOADAUTH	CHAR(1) NOT NULL	Whether the GRANTEE can use the LOAD utility to load tables in the database: blank Privilege is not held G Privilege held with the GRANT option Y Privilege is held without the GRANT option	G
REORGAUTH	CHAR(1) NOT NULL	Whether the GRANTEE can use the REORG utility to reorganize table spaces and indexes in the database: blank Privilege is not held G Privilege held with the GRANT option Y Privilege is held without the GRANT option	G
RECOVERDBAUTH	CHAR(1) NOT NULL	Whether the GRANTEE can use the RECOVER and REPORT utilities on table spaces in the database: blank Privilege is not held G Privilege held with the GRANT option Y Privilege is held without the GRANT option	G
REPAIRAUTH	CHAR(1) NOT NULL	Whether the GRANTEE can use the DIAGNOSE and REPAIR utilities on table spaces and indexes in the database: blank Privilege is not held G Privilege held with the GRANT option Y Privilege is held without the GRANT option	G
STARTDBAUTH	CHAR(1) NOT NULL	Whether the GRANTEE can use the START command against the database: blank Privilege is not held G Privilege held with the GRANT option Y Privilege is held without the GRANT option	G
STATSAUTH	CHAR(1) NOT NULL	Whether the GRANTEE can use the CHECK and RUNSTATS utilities against the database: blank Privilege is not held G Privilege held with the GRANT option Y Privilege is held without the GRANT option	G
STOPAUTH	CHAR(1) NOT NULL	Whether the GRANTEE can issue the STOP command against the database: blank Privilege is not held G Privilege held with the GRANT option Y Privilege is held without the GRANT option	G
IBMREQD	CHAR(1) NOT NULL	Whether the row came from the basic machine-readable material (MRM) tape: N No Y Yes	G

Column Name	Data Type	Description	Use
GRANTEDTS	TIMESTAMP NOT NULL WITH 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) NOT NULL	Name of the DBRM.	G
TIMESTAMP	CHAR(8) NOT NULL FOR BIT DATA	Consistency token.	S
PDSNAME	CHAR(44) NOT NULL	Name of the partitioned data set of which the DBRM is a member.	G
PLNAME	CHAR(8) NOT NULL	Name of the application plan of which this DBRM is a part.	G
PLCREATOR	CHAR(8) NOT NULL	Authorization ID of the owner of the application plan.	G
PRECOMPTIME	CHAR(8) NOT NULL	Time of precompilation in the form <i>hhmmssst</i> . If the LEVEL precompiler option is used, then this value does not represent the precompile time.	G
PRECOMPDATE	CHAR(6) NOT NULL	Date of precompilation in the form <i>yyymmdd</i> . If the LEVEL precompiler option is used, then this value does not represent the precompile date.	G
QUOTE	CHAR(1) NOT NULL	SQL string delimiter for the SQL statements in the DBRM: N Apostrophe Y Quotation mark	G
COMMA	CHAR(1) NOT NULL	Decimal point representation for SQL statements in the DBRM: N Period Y Comma	G
HOSTLANG	CHAR(1) NOT NULL	The host language used: B Assembler language C OS/VS COBOL D C F FORTRAN P PL/I 2 VS COBOL II or IBM COBOL Release 1 (formerly called COBOL/370) 3 IBM COBOL (Release 2 or subsequent releases) 4 C++	G
IBMREQD	CHAR(1) NOT NULL	Whether the row came from the basic machine-readable material (MRM) tape: N No Y Yes	G
CHARSET	CHAR(1) NOT NULL WITH DEFAULT	Indicates whether the system CCSID for SBCS data was 290 (Katakana) when the program was precompiled. A No K Yes	G
MIXED	CHAR(1) NOT NULL WITH 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 57): N No Y Yes	G

Column Name	Data Type	Description	Use
DEC31	CHAR(1) NOT NULL WITH 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 94): blank No Y Yes	G
VERSION	VARCHAR(64) NOT NULL WITH DEFAULT	Version identifier for the DBRM.	G
PRECOMPTS	TIMESTAMP NOT NULL WITH 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) NOT NULL	Whether the row came from the basic machine-readable material (MRM) tape: N No Y Yes	G

SYSIBM.SYSFIELDS Table

Contains one row for every column that has a field procedure.

Column Name	Data Type	Description	Use
TBCREATOR	CHAR(8) NOT NULL	Authorization ID of the owner of the table that contains the column.	G
TBNAME	VARCHAR(18) NOT NULL	Name of the table that contains the column.	G
COLNO	SMALLINT NOT NULL	Numeric place of this column in the table.	G
NAME	VARCHAR(18) NOT NULL	Name of the column.	G
FLDTYPE	CHAR(8) NOT NULL	Data type of the encoded values in the field ³⁶ . INTEGER Large integer SMALLINT Small integer FLOAT Floating-point CHAR Fixed-length character string VARCHAR Varying-length character string DECIMAL Decimal GRAPHIC Fixed-length graphic string VARG Varying-length graphic string	G
LENGTH	SMALLINT NOT NULL	The length attribute of the field; or, for a decimal field, its precision ³⁶ . The number does not include the internal prefixes that can be used to record actual length and null state. INTEGER 4 SMALLINT 2 FLOAT 8 CHAR Length of string VARCHAR Maximum length of string DECIMAL Precision of number GRAPHIC Number of DBCS characters VARG Maximum number of DBCS characters	G
SCALE	SMALLINT NOT NULL	Scale if FLDTYPE is DECIMAL; otherwise, 0.	G
FLDPROC	CHAR(8) NOT NULL	For a row describing a field procedure, the name of the procedure ³⁶ .	G
WORKAREA	SMALLINT 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 ³⁶ .	G
IBMREQD	CHAR(1) NOT NULL	Whether the row came from the basic machine-readable material (MRM) tape: N No Y Yes	G
EXITPARML	SMALLINT NOT NULL	For a row describing a field procedure, the length of the field procedure parameter value block ³⁶ .	G
PARMLIST	VARCHAR(254) 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 ³⁶ .	G
EXITPARAM	VARCHAR(1530) NOT NULL 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) ³⁶ .	G

³⁶ 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) NOT NULL	Authorization ID of the owner of the table that contains the column.	G
TBNAME	VARCHAR(18) NOT NULL	Name of the table that contains the column.	G
RELNAME	CHAR(8) NOT NULL	Constraint name for the constraint for which the column is part of the foreign key.	G
COLNAME	VARCHAR(18) NOT NULL	Name of the column.	G
COLNO	SMALLINT NOT NULL	Numeric place of the column in its table.	G
COLSEQ	SMALLINT NOT NULL	Numeric place of the column in the foreign key.	G
IBMREQD	CHAR(1) NOT NULL	Whether the row came from the basic machine-readable material (MRM) tape: N No Y Yes	G

SYSIBM.SYSINDEXES Table

Contains one row for every index.

Column Name	Data Type	Description	Use
NAME	VARCHAR(18) NOT NULL	Name of the index.	G
CREATOR	CHAR(8) NOT NULL	Authorization ID of the owner of the index.	G
TBNAME	VARCHAR(18) NOT NULL	Name of the table on which the index is defined.	G
TBCREATOR	CHAR(8) NOT NULL	Authorization ID of the owner of the table.	G
UNIQUERULE	CHAR(1) NOT NULL	Whether the index is unique: D No (duplicates are allowed) U Yes 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.) C Yes, and it is an index used to enforce UNIQUE constraint N Yes, and it is defined with UNIQUE WHERE NOT NULL R Yes, and it is an index used to enforce the uniqueness of a non-primary parent key	G
COLCOUNT	SMALLINT NOT NULL	The number of columns in the key.	G
CLUSTERING	CHAR(1) NOT NULL	Whether CLUSTER was specified when the index was created: N No Y Yes	G
CLUSTERED	CHAR(1) NOT NULL	Whether the table is actually clustered by the index: N No: a significant number of rows are not in clustering order, or statistics were not gathered. Y Yes: most of the rows are in clustering order. The entry can be changed by the RUNSTATS utility.	G
DBID	SMALLINT NOT NULL	Internal identifier of the database.	S
OBID	SMALLINT NOT NULL	Internal identifier of the index fan set descriptor.	S
ISOBID	SMALLINT NOT NULL	Internal identifier of the index page set descriptor.	S
DBNAME	CHAR(8) NOT NULL	Name of the database that contains the index.	G
INDEXSPACE	CHAR(8) NOT NULL	Name of the index space.	G
	INTEGER NOT NULL	Not used	N
	INTEGER NOT NULL	Not used	N
NLEAF	INTEGER NOT NULL	Number of active leaf pages in the index. The value is -1 before statistics are gathered. This is an updateable column.	S
NLEVELS	SMALLINT 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. Before statistics are gathered, the value is -1. This is an updateable column.	S
BPOOL	CHAR(8) NOT NULL	Name of the buffer pool used for the index.	G

SYSIBM.SYSINDEXES

Column Name	Data Type	Description	Use
PGSIZE	SMALLINT NOT NULL	Size, in bytes, of the subpages in the index: 256, 512, 1024, 2048, or 4096	G
ERASERULE	CHAR(1) NOT NULL	Whether the data sets are erased when dropped. The value is meaningless if the index is partitioned. N No Y Yes	G
DSETPASS	CHAR(8) NOT NULL	The password for the data sets of the index.	G
CLOSERULE	CHAR(1) NOT NULL	Whether the data sets are candidates for closure when the limit on the number of open data sets is reached. N No Y Yes	G
SPACE	INTEGER 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) NOT NULL	Whether the row came from the basic machine-readable material (MRM) tape: N No Y Yes C V2R1 dependency indicator; not from MRM tape D V2R2 dependency indicator; not from MRM tape E V2R3 dependency indicator; not from MRM tape G V4 dependency indicator; not from MRM tape	G
CLUSTERRATIO	SMALLINT NOT NULL WITH DEFAULT	Percentage of rows that are in clustering order. For a partitioned index, it is the weighted average of all index partitions in terms of the number of rows in the partition. 0 before statistics are gathered. This column is updateable.	S
CREATEDBY	CHAR(8) NOT NULL WITH DEFAULT	Primary authorization ID of the user who created the index.	G
#	SMALLINT NOT NULL	Internal use only	I
	SMALLINT NOT NULL	Not used	N
STATSTIME	TIMESTAMP NOT NULL WITH 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 updateable column.	G
INDEXTYPE	CHAR(1) NOT NULL WITH DEFAULT	The index type. 2 Type 2 index blank Type 1 index	G
FIRSTKEYCARDF	FLOAT NOT NULL WITH 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 before statistics are gathered. This is an updateable column.	S
FULLKEYCARDF	FLOAT NOT NULL WITH DEFAULT -1	Number of distinct values of the key. The value is -1 before statistics are gathered. This is an updateable column.	S
CREATEDTS	TIMESTAMP NOT NULL WITH 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

Column Name	Data Type	Description	Use
ALTEREDTS	TIMESTAMP NOT NULL WITH 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
PIECESIZE	INTEGER NOT NULL WITH DEFAULT	<p>Maximum size of a data set storage piece (in kilobytes, KB) that will be used by DB2 for non-partitioned indexes.</p> <p>The value of this column is one of the following:</p> <p>256 512 1024 2048 4096 8192 16384 32768 65536 131072 262144 524288 1048576 2097152 4194304 0</p> <p>The value of zero (0) indicates that this index is a partitioning index or that this index was created in a DB2 release prior to Version 5.</p>	G

#

SYSIBM.SYSINDEXPART Table

Contains one row for each nonpartitioned index and one row for each partition of a partitioned index.

Column Name	Data Type	Description	Use
PARTITION	SMALLINT NOT NULL	Partition number; 0 if index is not partitioned.	G
IXNAME	VARCHAR(18) NOT NULL	Name of the index.	G
IXCREATOR	CHAR(8) NOT NULL	Authorization ID of the owner of the index.	G
# PQTY # # #	INTEGER 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 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
STORATYPE	CHAR(1) NOT NULL	Type of storage allocation: E explicit and STORNAME names an integrated catalog facility catalog I implicit and STORNAME names a storage group	G
STORNAME	CHAR(8) NOT NULL	Name of storage group or integrated catalog facility catalog used for space allocation.	G
VCATNAME	CHAR(8) NOT NULL	Name of integrated catalog facility catalog used for space allocation.	G
	INTEGER NOT NULL	Not used	S
	INTEGER NOT NULL	Not used	S
LEAFDIST	INTEGER 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 NOT NULL	Not used	S
IBMREQD	CHAR(1) NOT NULL	Whether the row came from the basic machine-readable material (MRM) tape: N No Y Yes	G
LIMITKEY	VARCHAR(512) NOT NULL FOR BIT DATA	The high value of the limit key of the partition in an internal format. 0 if the index is not partitioned. If any column of the key has a field procedure, the internal format is the encoded form of the value.	S
FREEPAGE	SMALLINT NOT NULL	The number of pages that are loaded before a page is left as free space.	G
PCTFREE	SMALLINT NOT NULL	The percentage of each subpage or nonleaf page that is left as free space.	G
# SPACE # # # #	INTEGER NOT NULL WITH DEFAULT	Number of kilobytes of DASD storage allocated to the index space partition, as determined by the last execution of the STOSPACE or RUNSTATS 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).	G

Column Name	Data Type	Description	Use
STATSTIME	TIMESTAMP NOT NULL WITH 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) NOT NULL	Not used	N
GBPCACHE	CHAR(1) NOT NULL WITH DEFAULT	Group buffer pool cache option specified for this index or index partition.	G
		blank Only changed pages are cached in the group buffer pool. A Changed and unchanged pages are cached in the group buffer pool.	
FAROFFPOSF	FLOAT NOT NULL WITH 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.	S
NEAROFFPOSF	FLOAT NOT NULL WITH DEFAULT -1	Number of referred to rows near, but not at optimal position, because of an insert into a full page.	S
CARDF	FLOAT NOT NULL WITH DEFAULT -1	Number of rows referred to by the index or partition. The value is -1 if statistics have not been gathered.	S

SYSIBM.SYSINDEXSTATS Table

Contains one row for each partition of a partitioned index.

Column Name	Data Type	Description	Use
FIRSTKEYCARD	INTEGER NOT NULL	For the index partition, number of distinct values of the first key column. This is an updateable column.	S
FULLKEYCARD	INTEGER NOT NULL	For the index partition, number of distinct values of the key. This is an updateable column.	S
NLEAF	INTEGER NOT NULL	Number of active leaf pages in the index partition. This is an updateable column.	S
NLEVELS	SMALLINT NOT NULL	Number of levels in the partition index tree. This is an updateable column.	S
	SMALLINT NOT NULL	Not used	N
	SMALLINT NOT NULL	Not used	N
CLUSTERRATIO	SMALLINT NOT NULL	For the index partition, the percentage of rows that are in clustering order. 0 before statistics are gathered. This is an updateable column.	G
STATSTIME	TIMESTAMP NOT NULL	If RUNSTATS updated the statistics, the date and time when the last invocation of RUNSTATS updated the statistics. This is an updateable column.	G
IBMREQD	CHAR(1) NOT NULL	Whether the row came from the basic machine-readable material (MRM) tape. N No Y Yes	G
PARTITION	SMALLINT NOT NULL	Partition number of the index.	G
OWNER	CHAR(8) NOT NULL	Authorization ID of the owner of the index.	G
NAME	VARCHAR(18) NOT NULL	Name of the index.	G
KEYCOUNT	INTEGER NOT NULL	Total number of rows in the partition. This is an updateable column.	S

SYSIBM.SYSKEYS Table

Contains one row for each column of an index key.

Column Name	Data Type	Description	Use
IXNAME	VARCHAR(18) NOT NULL	Name of the index.	G
IXCREATOR	CHAR(8) NOT NULL	Authorization ID of the owner of the index.	G
COLNAME	VARCHAR(18) NOT NULL	Name of the column of the key.	G
COLNO	SMALLINT NOT NULL	Numeric position of the column in the table; for example, 4 (out of 10).	G
COLSEQ	SMALLINT NOT NULL	Numeric position of the column in the key; for example, 4 (out of 4).	G
ORDERING	CHAR(1) NOT NULL	Order of the column in the key: A Ascending D Descending	G
IBMREQD	CHAR(1) NOT NULL	Whether the row came from the basic machine-readable material (MRM) tape: N No Y Yes	G

SYSIBM.SYSPACKAGE Table

Contains a row for every package.

Column Name	Data Type	Description	Use
LOCATION	CHAR(16) NOT NULL	Always contains blanks	S
COLLID	CHAR(18) NOT NULL	Name of the package collection.	G
NAME	CHAR(8) NOT NULL	Name of the package.	G
CONTOKEN	CHAR(8) NOT NULL	Consistency token for the package. For a package derived from a DB2 DBRM, this is either: <ul style="list-style-type: none"> • The "level" as specified by the LEVEL option when the package's program was precompiled • The timestamp indicating when the package's program was precompiled, in an internal format. 	S
OWNER	CHAR(8) NOT NULL	Authorization ID of the package owner.	G
CREATOR	CHAR(8) NOT NULL	Authorization ID of the creator of the package version.	G
TIMESTAMP	TIMESTAMP NOT NULL	Timestamp indicating when the package was created.	G
BINDTIME	TIMESTAMP NOT NULL	Timestamp indicating when the package was last bound.	G
QUALIFIER	CHAR(8) NOT NULL	Implicit qualifier for the unqualified table, view, index, and alias names in the static SQL statements of the package.	G
PKSIZE	INTEGER NOT NULL	Size of the base section ³⁷ of the package, in bytes.	G
AVGSIZE	INTEGER NOT NULL	Average size, in bytes, of those sections ³⁷ of the plan that contain SQL statements processed at bind time.	G
SYSENTRIES	SMALLINT 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) NOT NULL	Whether the package is valid: <ul style="list-style-type: none"> A The description of the table or base table of a view referenced by the package was changed by the ALTER TABLE statement. The change did not require the invalidation of the package. H The description of the table or base table of a view referenced by the package was changed by the ALTER TABLE statement. The change will invalidate the package for a DB2 release prior to Version 5. N No Y Yes 	G
OPERATIVE	CHAR(1) NOT NULL	Whether the package can be allocated: <ul style="list-style-type: none"> N No; an explicit BIND or REBIND is required before the package can be allocated. Y Yes 	G

³⁷ 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.

Column Name	Data Type	Description	Use
VALIDATE	CHAR(1) NOT NULL	Whether validity checking can be deferred until run time: B All checking must be performed at bind time. R Validation is done at run time for tables, views, and privileges that do not exist at bind time.	G
ISOLATION	CHAR(1) NOT NULL	Isolation level when the package was last bound or rebound R RR (repeatable read) T RS (read stability) S CS (cursor stability) blank Not specified, and therefore at the level specified for the plan executing the package U UR (uncommitted read)	G
RELEASE	CHAR(1) NOT NULL	The value used for RELEASE when the package was last bound or rebound: C Value used was COMMIT. D Value used was DEALLOCATE. blank Not specified, and therefore the value specified for the plan executing the package.	G
EXPLAIN	CHAR(1) 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: N No Y Yes	G
QUOTE	CHAR(1) NOT NULL	SQL string delimiter for SQL statements in the package: N Apostrophe Y Quotation mark	G
COMMA	CHAR(1) NOT NULL	Decimal point representation for SQL statements in package: N Period Y Comma	G
HOSTLANG	CHAR(1) NOT NULL	Host language for the package's DBRM: B Assembler language C OS/VS COBOL D C F FORTRAN P PL/I 2 VS COBOL II or IBM COBOL Release 1 (formerly called COBOL/370) 3 IBM COBOL (Release 2 or subsequent releases) 4 C++ blank For remotely bound packages	G
CHARSET	CHAR(1) NOT NULL	Indicates whether the system CCSID for SBCS data was 290 (Katakana) when the program was precompiled: K Yes A No	G
MIXED	CHAR(1) 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 57): N No Y Yes	G
DEC31	CHAR(1) 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 94): N No Y Yes	G

SYSIBM.SYSPACKAGE

Column Name	Data Type	Description	Use
DEFERPREP	CHAR(1) NOT NULL	Indicates the CURRENTDATA option when the package was bound or rebound: A Data currency is required for all cursors. Inhibit blocking for all cursors. B Data currency is not required for ambiguous cursors. C Data currency is required for ambiguous cursors. blank The package was created before the CURRENTDATA option was available.	G
SQLERROR	CHAR(1) NOT NULL	Indicates the SQLERROR option on the most recent subcommand that bound or rebound the package: C CONTINUE N NOPACKAGE	G
REMOTE	CHAR(1) NOT NULL	Source of the package: C Package was created by BIND COPY. D Package was created by BIND COPY with the OPTIONS(COMMAND) option. K The package was copied from a package that was originally bound on behalf of a remote requester. L The package was copied with the OPTIONS(COMMAND) option from a package that was originally bound on behalf of a remote requester. N Package was locally bound from a DBRM. Y Package was bound on behalf of a remote requester.	G
PCTIMESTAMP	TIMESTAMP NOT NULL	Date and time the application program was precompiled, or 0001-01-01-00.00.000000 if the LEVEL precompiler option was used, or if the package came from a non-DB2 location.	G
IBMREQD	CHAR(1) NOT NULL	Whether the row came from the basic machine readable material (MRM) tape: N No Y Yes B V1R3 dependency indicator; not from MRM tape C V2R1 dependency indicator; not from MRM tape D V2R2 dependency indicator; not from MRM tape E V2R3 dependency indicator; not from MRM tape F V3R1 dependency indicator; not from MRM tape G V4 dependency indicator; not from MRM tape H V5 dependency indicator; not from MRM tape	G
VERSION	VARCHAR(64) NOT NULL	Version identifier for the package	G
PDSNAME	VARCHAR(44) 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: • The requester's location name if the product is DB2 • Otherwise, the requester's LU name enclosed in angle brackets; for example, "<LUSQLDS>."	G
DEGREE	CHAR(3) NOT NULL WITH DEFAULT	The DEGREE option used when the package was last bound: ANY DEGREE(ANY) 1 or blank DEGREE(1). Blank if the package was migrated.	G
GROUP_MEMBER	CHAR(8) NOT NULL WITH 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) NOT NULL WITH DEFAULT	B Dynamic SQL statements are handled like static SQL statements at run time.	G
		R Dynamic SQL statements are handled like dynamic SQL statements at run time.	
		blank 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.	
REOPTVAR	CHAR(1) NOT NULL WITH DEFAULT 'N'	Whether the access path is determined again at execution time using input variable values.	G
		N Bind option NOREOPT(VARS) indicates that the access path is determined at bind time.	
		Y Bind option REOPT(VARS) indicates that the access path is determined at execution time for SQL statements with variable values.	
DEFERPREPARE	CHAR(1) NOT NULL WITH DEFAULT	Whether PREPARE processing is deferred until OPEN is executed.	G
		N Bind option NODEFER(PREPARE) indicates that PREPARE processing is not deferred until OPEN is executed.	
		Y Bind option DEFER(PREPARE) indicates that PREPARE processing is deferred until OPEN is executed.	
KEEPDYNAMIC	CHAR(1) NOT NULL WITH DEFAULT 'N'	Whether prepared dynamic statements are to be purged at each commit point.	G
		N The bind option is KEEPDYNAMIC(NO). Prepared dynamic SQL statements are destroyed at commit.	
		Y The bind option is KEEPDYNAMIC(YES). Prepared dynamic SQL statements are kept past commit.	

SYSIBM.SYSPACKAUTH Table

Records the privileges held by users over packages.

Column Name	Data Type	Description	Use
GRANTOR	CHAR(8) NOT NULL	Authorization ID of the user who granted the privilege. Could also be PUBLIC or PUBLIC followed by an asterisk ³⁸ .	G
GRANTEE	CHAR(8) 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) NOT NULL	Always contains blanks	S
COLLID	CHAR(18) NOT NULL	Collection name for the package or packages on which the privilege was granted.	G
NAME	CHAR(8) 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) NOT NULL	Not used	N
TIMESTAMP	TIMESTAMP NOT NULL	Timestamp indicating when the privilege was granted.	G
GRANTEETYPE	CHAR(1) NOT NULL	Type of grantee: blank Authorization ID P Application plan	G
AUTHHOWGOT	CHAR(1) 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.	G
		blank Not applicable	
		A PACKADM (on collection *)	
		C DBCTL	
		D DBADM	
		L SYCTRL	
		M DBMAINT	
P PACKADM (on a specific collection)			
S SYSADM			
BINDAUTH	CHAR(1) NOT NULL	Whether GRANTEE can use the BIND and REBIND subcommands against the package: blank Privilege is not held G Privilege is held with the GRANT option Y Privilege is held without the GRANT option	G
COPYAUTH	CHAR(1) NOT NULL	Whether GRANTEE can COPY the package: blank Privilege is not held G Privilege is held with the GRANT option Y Privilege is held without the GRANT option	G
EXECUTEAUTH	CHAR(1) NOT NULL	Whether GRANTEE can execute the package: blank Privilege is not held G Privilege is held with the GRANT option Y Privilege is held without the GRANT option	G
IBMREQD	CHAR(1) NOT NULL	Whether the row came from the basic machine readable material (MRM) tape: N No Y Yes	G

³⁸ PUBLIC followed by an asterisk (PUBLIC*) denotes PUBLIC AT ALL LOCATIONS. For the conditions where GRANTOR can be PUBLIC or PUBLIC*, see Section 3 (Volume 1) of *Administration Guide*.

SYSIBM.SYSPACKDEP Table

Records the dependencies of packages on local tables, views, synonyms, table spaces, indexes, and aliases.

Column Name	Data Type	Description	Use
BNAME	VARCHAR(18) NOT NULL	The name of an object that a package depends on.	G
BQUALIFIER	CHAR(8) NOT NULL	If BNAME identifies a table space, the name of its database. Otherwise, the authorization ID of the owner of BNAME.	G
BTYPE	CHAR(1) NOT NULL	Type of object identified by BNAME and BQUALIFIER: A Alias I Index P Partitioned table space R Table space S Synonym T Table V View	G
DLOCATION	CHAR(16) NOT NULL	Always contains blanks	S
DCOLLID	CHAR(18) NOT NULL	Name of the package collection	G
DNAME	CHAR(8) NOT NULL	Name of the package	G
DCONTOKEN	CHAR(8) NOT NULL	Consistency token for the package. This is either: <ul style="list-style-type: none"> The "level" as specified by the LEVEL option when the package's program was precompiled The timestamp indicating when the package's program was precompiled, in an internal format. 	S
IBMREQD	CHAR(1) NOT NULL	Whether the row came from the basic machine readable material (MRM) tape: N No Y Yes	G

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) NOT NULL	Name of the application plan	G
SEQNO	SMALLINT NOT NULL	Sequence number of the entry in the package list	G
LOCATION	CHAR(16) 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) NOT NULL	Collection name for the package. An asterisk (*) indicates that the collection name is determined at run time.	G
NAME	CHAR(8) NOT NULL	Name of the package. An asterisk (*) indicates an entire collection.	G
TIMESTAMP	TIMESTAMP NOT NULL	Timestamp indicating when the row was created.	G
IBMREQD	CHAR(1) NOT NULL	Whether the row came from the basic machine-readable material (MRM) tape: N No Y Yes	G

SYSIBM.SYSPACKSTMT Table

Contains one or more rows for each statement in a package.

Column Name	Data Type	Description	Use
LOCATION	CHAR(16) NOT NULL	Always contains blanks	S
COLLID	CHAR(18) NOT NULL	Name of the package collection.	G
NAME	CHAR(8) NOT NULL	Name of the package.	G
CONTOKEN	CHAR(8) NOT NULL	Consistency token for the package. This is either: <ul style="list-style-type: none"> • The "level" as specified by the LEVEL option when the package's program was precompiled • The timestamp indicating when the package's program was precompiled, in an internal format 	S
SEQNO	SMALLINT NOT NULL	Sequence number of the row with respect to a statement in the package ³⁹ . The numbering starts with 0.	G
STMTNO	SMALLINT NOT NULL	Statement number of the statement in the package's source program. A statement number greater than 32767 will be displayed as zero ³⁹ or a negative number ⁴⁰ .	G
SECTNO	SMALLINT NOT NULL	Section number of the statement ³⁹ .	G
BINDERROR	CHAR(1) NOT NULL	Whether an SQL error was detected at bind time: N No Y Yes	G
IBMREQD	CHAR(1) NOT NULL	Whether the row came from the basic machine readable material (MRM) tape: N No Y Yes	G
VERSION	VARCHAR(64) NOT NULL	Version identifier for the package.	G
STMT	VARCHAR(254) NOT NULL	All or a portion of the text for the SQL statement that the row represents.	S
ISOLATION	CHAR(1) NOT NULL WITH DEFAULT	Isolation level for the SQL statement. 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. The isolation level is recorded in SYSPACKAGE.ISOLATION and in SYSPLAN.ISOLATION.	G

³⁹ Rows in which the value of SEQNO, STMTNO, and SECTNO are zero are for internal use.

⁴⁰ 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).

Column Name	Data Type	Description	Use
STATUS	CHAR(1) NOT NULL WITH DEFAULT	Status of binding the statement.	S
		A 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.	
		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.	
		C Compiled - statement was bound successfully using defaults for input variables during access path selection.	
		E Explain - statement is an SQL EXPLAIN statement. The explain is done at bind time using defaults for input variables during access path selection.	
		F 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.	
		G 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.	
		H 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.	
		I Indefinite - statement is dynamic. The statement will be bound at execution time using defaults for input variables during access path selection.	
		J Indefinite - statement is dynamic. The statement will be bound at execution time using values for input variables during access path selection.	
		K Control - CALL statement.	
		L Bad - the statement has some allowable error. The bind continues but the statement cannot be executed.	
		blank The statement is non-executable, or was bound in a DB2 release prior to Version 5.	

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) NOT NULL	Always contains blanks	S
COLLID	CHAR(18) NOT NULL	Name of the package collection.	G
NAME	CHAR(8) NOT NULL	Name of the package.	G
CONTOKEN	CHAR(8) NOT NULL	Consistency token for the package. This is either: <ul style="list-style-type: none"> The "level" as specified by the LEVEL option when the package's program was precompiled The timestamp indicating when the package's program was precompiled, in an internal format. 	S
SYSTEM	CHAR(8) NOT NULL	Environment. Values can be: <p>BATCH TSO batch CICS Customer Information Control System DB2CALL DB2 call attachment facility DLIBATCH DLI batch support facility IMSBMP IMS BMP region IMSMPP IMS MPP and IFP region REMOTE remote application server</p>	G
ENABLE	CHAR(1) NOT NULL	Indicates whether the connections represented by the row are enabled or disabled: <p>N Disabled Y Enabled</p>	G
CNAME	CHAR(20) 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"> Blank if SYSTEM=BATCH or SYSTEM=DB2CALL The LU name for an application server if SYSTEM=REMOTE Either the requester's location (if the product is DB2) or the requester's LU name enclosed in angle brackets if SYSTEM=REMOTE. The name of a single connection if SYSTEM has any other value. <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) NOT NULL	Whether the row came from the basic machine readable material (MRM) tape: <p>N No Y Yes</p>	G

SYSIBM.SYSPLAN Table

Contains one row for each application plan.

Column Name	Data Type	Description	Use
NAME	CHAR(8) NOT NULL	Name of the application plan.	G
CREATOR	CHAR(8) NOT NULL	Authorization ID of the owner of the application plan.	G
BINDDATE	CHAR(6) NOT NULL	The date on which the plan was last bound, in the form <i>yymmdd</i> .	G
VALIDATE	CHAR(1) NOT NULL	Whether validity checking can be deferred until run time: B All checking must be performed during BIND. R Validation is done at run time for tables, views, and privileges that do not exist at bind time.	G
ISOLATION	CHAR(1) NOT NULL	Isolation level for the plan: R RR (repeatable read) T RS (read stability) S CS (cursor stability) U UR (uncommitted read)	G
VALID	CHAR(1) NOT NULL	Whether the application plan is valid: A The description of the table or base table of a view referenced by the application plan was changed by the ALTER TABLE statement. The change did not require the invalidation of the application plan. H The description of the table or base table of a view referenced by the application plan was changed by the ALTER TABLE statement. The change will invalidate the plan for a DB2 release prior to Version 5. N No Y Yes	G
OPERATIVE	CHAR(1) NOT NULL	Whether the application plan can be allocated: N No; an explicit BIND or REBIND is required before the plan can be allocated Y Yes	G
BINDTIME	CHAR(8) NOT NULL	Time of the BIND in the form <i>hhmmssstth</i> .	G
PLSIZE	INTEGER NOT NULL	Size of the base section ⁴¹ of the plan, in bytes.	G
IBMREQD	CHAR(1) NOT NULL	Whether the row came from the basic machine-readable material (MRM) tape: N No Y Yes B V1R3 dependency indicator; not from MRM tape C V2R1 dependency indicator; not from MRM tape D V2R2 dependency indicator; not from MRM tape E V2R3 dependency indicator; not from MRM tape F V3R1 dependency indicator; not from MRM tape G V4 dependency indicator; not from MRM tape H V5 dependency indicator; not from MRM tape	G
AVGSIZE	INTEGER NOT NULL	Average size, in bytes, of those sections ⁴¹ of the plan that contain SQL statements processed at bind time.	G

⁴¹ 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.

Column Name	Data Type	Description	Use
ACQUIRE	CHAR(1) NOT NULL	When resources are acquired: A At allocation U At first use	G
RELEASE	CHAR(1) NOT NULL	When resources are released: C At commit D At deallocation	G
	CHAR(1) NOT NULL	Not used	N
	CHAR(1) NOT NULL	Not used	N
	CHAR(1) NOT NULL	Not used	N
EXPLAN	CHAR(1) 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: N No Y Yes	G
EXPREDICATE	CHAR(1) NOT NULL	Indicates the CURRENTDATA option when the plan was bound or rebound: B Data currency is not required for ambiguous cursors. Allow blocking for ambiguous cursors. C Data currency is required for ambiguous cursors. Inhibit blocking for ambiguous cursors. N Blocking is inhibited for ambiguous cursors, but the plan was created before the CURRENTDATA option was available.	G
BOUNDBY	CHAR(8) NOT NULL WITH DEFAULT	Primary authorization ID of the binder of the plan.	G
QUALIFIER	CHAR(8) NOT NULL WITH DEFAULT	Implicit qualifier for the unqualified table, view, index, and alias names in the static SQL statements of the plan.	G
CACHESIZE	SMALLINT NOT NULL WITH 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 NOT NULL WITH 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 fallback.	G
DEFERPREP	CHAR(1) NOT NULL WITH DEFAULT	Whether the package was last bound with the DEFER(PREPARE) option: N No Y Yes	G
CURRENTSERVER	CHAR(16) NOT NULL WITH 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 NOT NULL WITH 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 fallback. A negative value or zero means that all connections are enabled.	G
DEGREE	CHAR(3) NOT NULL WITH DEFAULT	The DEGREE option used when the plan was last bound: ANY DEGREE(ANY) 1 or blank DEGREE(1). Blank if the plan was migrated.	G

SYSIBM.SYSPLAN

Column Name	Data Type	Description	Use
SQLRULES	CHAR(1) NOT NULL WITH DEFAULT	The SQLRULES option used when the plan was last bound: D or blank SQLRULES(DB2) S SQLRULES(STD) blank A migrated plan	G
DISCONNECT	CHAR(1) NOT NULL WITH DEFAULT	The DISCONNECT option used when the plan was last bound: E or blank DISCONNECT(EXPLICIT) (EXPLICIT) A DISCONNECT(AUTOMATIC) (AUTOMATIC) C DISCONNECT(CONDITIONAL) (CONDITIONAL) blank a migrated plan.	G
GROUP_MEMBER	CHAR(8) NOT NULL WITH 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
DYNAMICRULES	CHAR(1) NOT NULL WITH DEFAULT	B Dynamic SQL statements are handled like static SQL statements at run time. blank Dynamic SQL statements are handled like dynamic SQL statements at run time.	G
BOUNDTS	TIMESTAMP NOT NULL WITH DEFAULT	Time when the plan was bound.	G
REOPTVAR	CHAR(1) NOT NULL WITH DEFAULT 'N'	Whether the access path is determined again at execution time using input variable values. N Bind option NOREOPT(VARS) indicates that the access path is determined at bind time. Y Bind option REOPT(VARS) indicates that the access path is determined at execution time for SQL statements with variable values.	G
KEEPDYNAMIC	CHAR(1) NOT NULL WITH DEFAULT 'N'	Whether prepared dynamic statements are to be purged at each commit point. N The bind option is KEEPDYNAMIC(NO). Prepared dynamic SQL statements are destroyed at commit or rollback. Y The bind option is KEEPDYNAMIC(YES). Prepared dynamic SQL statements are kept past commit or rollback.	G

SYSIBM.SYSPLANAUTH Table

Records the privileges held by users over application plans.

Column Name	Data Type	Description	Use	
GRANTOR	CHAR(8) NOT NULL	Authorization ID of the user who granted the privileges.	G	
GRANTEE	CHAR(8) NOT NULL	Authorization ID of the user who holds the privileges. Could also be PUBLIC for a grant to PUBLIC.	G	
NAME	CHAR(8) NOT NULL	Name of the application plan on which the privileges are held.	G	
	CHAR(12) NOT NULL	Internal use only	I	
DATEGRANTED	CHAR(6) NOT NULL	Date the privileges were granted; in the form <i>yyymmdd</i> .	G	
TIMEGRANTED	CHAR(8) NOT NULL	Time the privileges were granted; in the form <i>hhmmssst</i> .	G	
	CHAR(1) NOT NULL	Not used	N	
AUTHHOWGOT	CHAR(1) 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.	G	
		blank		Not applicable
		C		DBCTL
		D		DBADM
		L		SYSCTRL
		M		DBMAINT
		S		SYSADM
BINDAUTH	CHAR(1) NOT NULL	Whether the GRANTEE can use the BIND, REBIND, or FREE subcommands against the plan:	G	
		blank		Privilege is not held
		G		Privilege is held with the GRANT option
		Y		Privilege is held without the GRANT option
EXECUTEAUTH	CHAR(1) NOT NULL	Whether the GRANTEE can run application programs that use the application plan:	G	
		blank		Privilege is not held
		G		Privilege is held with the GRANT option
		Y		Privilege is held without the GRANT option
IBMREQD	CHAR(1) NOT NULL	Whether the row came from the basic machine-readable material (MRM) tape:	G	
		N		No
		Y		Yes
GRANTEDTS	TIMESTAMP NOT NULL WITH DEFAULT	Time when the GRANT statement was executed.	G	

SYSIBM.SYSPLANDEP Table

Records the dependencies of plans on tables, views, aliases, synonyms, table spaces, and indexes.

Column Name	Data Type	Description	Use
BNAME	VARCHAR(18) NOT NULL	Name of an object the plan is dependent on.	G
BCREATOR	CHAR(8) NOT NULL	If BNAME is a table space, its database. Otherwise, the authorization ID of the owner of BNAME.	G
BTYPE	CHAR(1) NOT NULL	Type of object BNAME: A Alias I Index P Partitioned table space R Table space S Synonym T Table V View	G
DNAME	CHAR(8) NOT NULL	Name of the plan.	G
IBMREQD	CHAR(1) NOT NULL	Whether the row came from the basic machine-readable material (MRM) tape: N No Y Yes	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) NOT NULL	Name of the plan	G
SYSTEM	CHAR(8) NOT NULL	Environment. Values can be: BATCH TSO batch DB2CALL DB2 call attachment facility CICS Customer Information Control System DLIBATCH DLI batch support facility IMSBMP IMS BMP region IMSMPP IMS MPP or IFP region	G
ENABLE	CHAR(1) NOT NULL	Indicates whether the connections represented by the row are enabled or disabled: N Disabled Y Enabled	G
CNAME	CHAR(8) 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"> • Blank if SYSTEM=BATCH or SYSTEM=DB2CALL • The name of a single connection if SYSTEM has any other value 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) NOT NULL	Whether the row came from the basic machine-readable material (MRM) tape: N No Y Yes	G

SYSIBM.SYSPROCEDURES Table

Contains one row for every stored procedure. Rows in this table must be inserted, updated, and deleted; they are not automatically inserted and maintained by DB2.

One method of adding rows to this table is by using the LOAD utility.

Column Name	Data Type	Description	Use
PROCEDURE	CHAR(18) NOT NULL	The name of the stored procedure specified on the SQL CALL statement.	G
AUTHID	CHAR(8) NOT NULL WITH DEFAULT	The 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. If AUTHID is blank, values in this row apply to all authorization IDs.	G
LUNAME	CHAR(8) NOT NULL WITH DEFAULT	The LUNAME of the system that issued the SQL CALL statement. <ul style="list-style-type: none"> If the LUNAME column contains the local DB2 system's LUNAME, this row applies to local applications that issue the SQL CALL statement. If the LUNAME column contains the LUNAME of a remote client, this row applies to SQL CALL statements received from that remote client. 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. To ease migration to future releases of DB2, specify blanks in this field.	G
# LOADMOD # # #	CHAR(8) NOT NULL	The program that runs when the CALL statement is executed. When the value of LANGUAGE is COMPJAVA, this column value is not used.	G
# LINKAGE # # # # # # # # # # # #	CHAR(1) NOT NULL WITH DEFAULT	The linkage convention used to pass parameters to the stored procedure: blank The SIMPLE linkage convention is used where input parameters cannot be null. N The SIMPLE WITH NULLS convention is used where an indicator array is passed to the stored procedure. Null input parameters are allowed. This value must be N when the value of LANGUAGE is COMPJAVA. Conventions for passing parameters to stored procedures are described in Section 6 of <i>Application Programming and SQL Guide</i> .	G
COLLID	CHAR(18) NOT NULL	The name of the package collection to use when the stored procedure is executed. 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) NOT NULL	The programming language used to create the stored procedure. Possible values are 'ASSEMBLE', 'PLI', 'COBOL', 'C', 'REXX', or 'COMPJAVA'.	G

Column Name	Data Type	Description	Use
ASUTIME	INTEGER NOT NULL WITH DEFAULT	The 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. If a stored procedure uses more service units than allowed by the ASUTIME value, DB2 cancels the stored procedure.	G
STAYRESIDENT	CHAR(1) NOT NULL WITH DEFAULT	Determines whether the stored procedure load module is deleted from memory when the stored procedure ends. Y The load module remains resident in memory after the stored procedure ends. blank The load module is deleted from memory after the stored procedure ends.	G
IBMREQD	CHAR(1) NOT NULL	Indicates whether the row came from the basic machine readable material (MRM) tape: N No Y Yes	G
# RUNOPTS	VARCHAR(254) NOT NULL	The Language Environment (Language Environment for MVS & VM) run-time options to use for this stored procedure. For a REXX stored procedure, this value is the Language Environment run-time options for the REXX interface to DB2. If this column contains an empty string, the installation default Language Environment run-time options are used. An example Language Environment run-time option list follows: 'MSGFILE(OUTFILE),RPTSTG(ON),RPTOPTS(ON)' When the value of LANGUAGE is COMPJAVA, this column value is the stored procedure program name, in the format <i>class.method</i> or <i>package.class.method</i> .	G
PARMLIST	VARCHAR(3000) 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 Section 6 of <i>Application Programming and SQL Guide</i> .	G
RESULT_SETS	SMALLINT NOT NULL WITH DEFAULT	The maximum number of query result sets that can be returned by this stored procedure. Zero indicates there are no query result sets.	G
# WLM_ENV	CHAR(18) NOT NULL WITH DEFAULT	The name of the WLM environment to be used to run this stored procedure. A blank value results in the stored procedure being run in the DB2-established stored procedures address space. For a REXX or COMPJAVA stored procedure, this value must be non-blank.	G
# PGM_TYPE	CHAR(1) NOT NULL WITH DEFAULT 'M'	Whether the stored procedure runs as a main routine or a subroutine. M The stored procedure runs as a main routine. S The stored procedure runs as a subroutine. For a REXX stored procedure, this value is ignored.	G
EXTERNAL_SECURITY	CHAR(1) NOT NULL WITH DEFAULT 'N'	Whether a special RACF environment is required to control access to non-SQL resources. N 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. Y 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) WITH DEFAULT 'N'	Indicates that the unit of work is always to be committed immediately upon successful return (non-negative SQLCODE) from this stored procedure. N The unit of work is to continue. Y The unit of work is to be committed. 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) NOT NULL	Authorization ID of the owner of the dependent table of the referential constraint.	G
TBNAME	VARCHAR(18) NOT NULL	Name of the dependent table of the referential constraint.	G
RELNAME	CHAR(8) NOT NULL	Constraint name	G
REFTBNAME	VARCHAR(18) NOT NULL	Name of the parent table of the referential constraint.	G
REFTBCREATOR	CHAR(8) NOT NULL	Authorization ID of the owner of the parent table.	G
COLCOUNT	SMALLINT NOT NULL	Number of columns in the foreign key.	G
DELETERULE	CHAR(1) NOT NULL	Type of delete rule for the referential constraint. C CASCADE N SET NULL R RESTRICT A NO ACTION	G
IBMREQD	CHAR(1) NOT NULL	Whether the row came from the basic machine-readable material (MRM) tape: N No Y Yes	G
RELOBID1	SMALLINT NOT NULL WITH DEFAULT	Internal identifier of the constraint with respect to the database that contains the parent table.	S
RELOBID2	SMALLINT NOT NULL WITH DEFAULT	Internal identifier of the constraint with respect to the database that contains the dependent table.	S
TIMESTAMP	TIMESTAMP NOT NULL WITH DEFAULT	The 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.' :epsc..	G
IXOWNER	CHAR(8) NOT NULL WITH 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) NOT NULL WITH 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 USE privileges for buffer pools, storage groups, and table spaces, and CREATE IN and PACKADM ON privileges for collections.

Column Name	Data Type	Description	Use
GRANTOR	CHAR(8) NOT NULL	Authorization ID of the user who granted the privilege.	G
GRANTEE	CHAR(8) NOT NULL	Authorization ID of the user who holds the privilege. Could also be PUBLIC for a grant to PUBLIC.	G
QUALIFIER	CHAR(8) NOT NULL	The qualifier of the table space (the database name), if the row describes a privilege over a table space. Blank otherwise.	G
NAME	CHAR(18) NOT NULL	Name of the buffer pool, collection, DB2 storage group, or table space. Could also be ALL when USE OF ALL BUFFERPOOLS is granted.	G
	CHAR(1) NOT NULL	Not used	N
AUTHHOWGOT	CHAR(1) 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.	G
		blank Not applicable	
		C DBCTL	
		D DBADM	
		L SYCTRL	
		M DBMAINT	
		S SYSADM	
P PACKADM (on a specific collection)			
A PACKADM (on collection *)			
OBTYP	CHAR(1) NOT NULL	Object type:	G
		B Buffer pool C Collection S Storage group R Table space	
	CHAR(12) NOT NULL	Internal use only	I
DATEGRANTED	CHAR(6) NOT NULL	Date the privilege was granted; in the form <i>yymmdd</i> .	G
TIMEGRANTED	CHAR(8) NOT NULL	Time the privilege was granted; in the form <i>hhmmssst</i> .	G
USEAUTH	CHAR(1) NOT NULL	Whether the privilege is held with the GRANT option:	G
		G Privilege is held with the GRANT option Y Privilege is held without the GRANT option	
		The authority held is PACKADM when the OBTYP is C (a collection) and QUALIFIER is PACKADM. The authority held is CREATE IN when the OBTYP is C and QUALIFIER is blank.	
IBMREQD	CHAR(1) NOT NULL	Whether the row came from the basic machine-readable material (MRM) tape:	G
		N No Y Yes	
GRANTEDTS	TIMESTAMP NOT NULL WITH DEFAULT	Time when the GRANT statement was executed.	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) NOT NULL	Name of the DBRM.	G
PLNAME	CHAR(8) NOT NULL	Name of the application plan.	G
PLCREATOR	CHAR(8) NOT NULL	Authorization ID of the owner of the application plan.	G
SEQNO	SMALLINT NOT NULL	The sequence number of this row with respect to a statement of the DBRM ⁴² . The numbering starts with zero.	G
STMTNO	SMALLINT NOT NULL	Statement number of the SQL statement in the source program. A statement number greater than 32767 will be displayed as zero ⁴² .	G
SECTNO	SMALLINT NOT NULL	The section number of the section within the DBRM identified in the NAME column ⁴² .	G
IBMREQD	CHAR(1) NOT NULL	Whether the row came from the basic machine-readable material (MRM) tape: N No Y Yes	G
TEXT	VARCHAR(254) NOT NULL	The text or portion of the text of the SQL statement.	S
ISOLATION	CHAR(1) NOT NULL WITH DEFAULT	Isolation level for the SQL statement. 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. The isolation level is recorded in SYSPACKAGE.ISOLATION and in SYSPLAN.ISOLATION.	G

⁴² Rows in which the values of SEQNO, STMTNO, and SECTNO are zero are for internal use.

Column Name	Data Type	Description	Use
STATUS	CHAR(1) NOT NULL WITH DEFAULT	Status of binding the statement.	S
		A 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.	
		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.	
		C Compiled - statement was bound successfully using defaults for input variables during access path selection.	
		E Explain - statement is an SQL EXPLAIN statement. The explain is done at bind time using defaults for input variables during access path selection.	
		F 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.	
		G 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.	
		H 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.	
		I Indefinite - statement is dynamic. The statement will be bound at execution time using defaults for input variables during access path selection.	
		J Indefinite - statement is dynamic. The statement will be bound at execution time using values for input variables during access path selection.	
		K Control - CALL statement.	
		L Bad - the statement has some allowable error. The bind continues but the statement cannot be executed.	
		blank The statement is non-executable, or was bound in a DB2 release prior to Version 5.	

SYSIBM.SYSSTOGROUP Table

Contains one row for each storage group.

Column Name	Data Type	Description	Use
NAME	CHAR(8) NOT NULL	Name of the storage group.	G
CREATOR	CHAR(8) NOT NULL	Authorization ID of the owner of the storage group.	G
VCATNAME	CHAR(8) NOT NULL	Name of the integrated catalog facility catalog.	G
VPASSWORD	CHAR(8) NOT NULL	Password for the integrated catalog facility catalog.	G
SPACE	INTEGER 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) NOT NULL	Date when the SPACE column was last updated, in the form <i>yyddd</i> .	G
IBMREQD	CHAR(1) NOT NULL	Whether the row came from the basic machine-readable material (MRM) tape: N No Y Yes	G
CREATEDBY	CHAR(8) NOT NULL WITH DEFAULT	Primary authorization ID of the user who created the storage group.	G
STATSTIME	TIMESTAMP NOT NULL WITH DEFAULT	If STOSPACE utility was executed for the storage group, date and time when last executed.	G
CREATEDTS	TIMESTAMP NOT NULL WITH DEFAULT	Time when the CREATE statement was executed for the storage group.	G
ALTEREDTS	TIMESTAMP NOT NULL WITH 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.

Column Name	Data Type	Description	Use
INCCSID	INTEGER NOT NULL	The source CCSID for the character conversion represented by this row.	G
OUTCCSID	INTEGER NOT NULL	The target CCSID for the character conversion represented by this row.	G
TRANSTYPE	CHAR(2) NOT NULL	Indicates the nature of the conversion. Values can be: GG GRAPHIC to GRAPHIC MM EBCDIC MIXED to EBCDIC MIXED MS EBCDIC MIXED to SBCS PM ASCII MIXED to EBCDIC MIXED PS ASCII MIXED to SBCS SM SBCS to EBCDIC MIXED SS SBCS to SBCS MP EBCDIC MIXED to ASCII MIXED PP ASCII MIXED to ASCII MIXED SP SBCS to ASCII MIXED	G
ERRORBYTE	CHAR(1) FOR BIT DATA (Nulls are allowed)	The byte used in the conversion table as an error byte. Null indicates the absence of an error byte.	S
SUBBYTE	CHAR(1) FOR BIT DATA (Nulls are allowed)	The byte used in the conversion table as a substitution character. Null indicates the absence of a substitution character.	S
TRANSPROC	CHAR(8) NOT NULL WITH 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) NOT NULL	Whether the row came from the basic machine-readable material (MRM) tape (see the information following this table): N No Y Yes	G
TRANSTAB	VARCHAR(256) FOR BIT DATA NOT NULL WITH 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 would then use 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 *Installation Guide*.

SYSIBM.SYSSYNONYMS Table

Contains one row for each synonym of a table or view.

Column Name	Data Type	Description	Use
NAME	VARCHAR(18) NOT NULL	Synonym for the table or view.	G
CREATOR	CHAR(8) NOT NULL	Authorization ID of the owner of the synonym.	G
TBNAME	VARCHAR(18) NOT NULL	Name of the table or view.	G
TBCREATOR	CHAR(8) NOT NULL	Authorization ID of the owner of the table or view.	G
IBMREQD	CHAR(1) NOT NULL	Whether the row came from the basic machine-readable material (MRM) tape: N No Y Yes	G
CREATEDBY	CHAR(8) NOT NULL WITH DEFAULT	Primary authorization ID of the user who created the synonym.	G
CREATEDTS	TIMESTAMP NOT NULL WITH DEFAULT	Time when the CREATE statement was executed for the synonym. The value will be '0001-01.01.00.00.00.000000' for synonyms created in a DB2 release prior to Version 5.	G

SYSIBM.SYSTABAUTH Table

Records the privileges held by users on tables and views.

Column Name	Data Type	Description	Use
GRANTOR	CHAR(8) NOT NULL	Authorization ID of the user who granted the privileges. Could also be PUBLIC, or PUBLIC followed by an asterisk. ⁴³	G
GRANTEE	CHAR(8) 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) NOT NULL	Meaning: blank GRANTEE is an authorization ID P GRANTEE is an application plan or a package. It is a package if COLLID is not blank.	G
DBNAME	CHAR(8) 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) 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) 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) NOT NULL	Authorization ID of the owner of the table or view.	G
TTNAME	VARCHAR(18) NOT NULL	Name of the table or view.	G
AUTHHOWGOT	CHAR(1) 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. blank Not applicable C DBCTL D DBADM L SYSCTRL M DBMAINT S SYSADM	G
	CHAR(12) NOT NULL	Internal use only	I
DATEGRANTED	CHAR(6) NOT NULL	Date the privileges were granted, in the form <i>yymmdd</i> .	G
TIMGRANTED	CHAR(8) NOT NULL	Time the privileges were granted, in the form <i>hhmmssst</i> .	G
UPDATECOLS	CHAR(1) 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

⁴³ PUBLIC followed by an asterisk (PUBLIC*) denotes PUBLIC AT ALL LOCATIONS. For the conditions where GRANTOR can be PUBLIC or PUBLIC*, see Section 3 (Volume 1) of *Administration Guide*.

Column Name	Data Type	Description	Use
ALTERAUTH	CHAR(1) NOT NULL	Whether the GRANTEE can alter the table: blank Privilege not held G Privilege is held with the GRANT option Y Privilege is held without the GRANT option	G
DELETEAUTH	CHAR(1) NOT NULL	Whether the GRANTEE can delete rows from the table or view: blank Privilege not held G Privilege is held with the GRANT option Y Privilege is held without the GRANT option	G
INDEXAUTH	CHAR(1) NOT NULL	Whether the GRANTEE can create indexes on the table: blank Privilege not held G Privilege is held with the GRANT option Y Privilege is held without the GRANT option	G
INSERTAUTH	CHAR(1) NOT NULL	Whether the GRANTEE can insert rows into the table or view: blank Privilege not held G Privilege is held with the GRANT option Y Privilege is held without the GRANT option	G
SELECTAUTH	CHAR(1) NOT NULL	Whether the GRANTEE can select rows from the table or view: blank Privilege not held G Privilege is held with the GRANT option Y Privilege is held without the GRANT option	G
UPDATEAUTH	CHAR(1) NOT NULL	Whether the GRANTEE can update rows of the table or view: blank Privilege not held G Privilege is held with the GRANT option Y Privilege is held without the GRANT option	G
IBMREQD	CHAR(1) NOT NULL	Whether the row came from the basic machine-readable material (MRM) tape: N No Y Yes	G
	CHAR(16) NOT NULL WITH DEFAULT	Not used	N
	CHAR(16) NOT NULL WITH DEFAULT	Not used	N
COLLID	CHAR(18) NOT NULL WITH DEFAULT	If the GRANTEE is a package, its collection name. Otherwise, blank.	G
CONTOKEN	CHAR(8) NOT NULL WITH DEFAULT	If the GRANTEE is a package, the consistency token of the DBRM from which the package was derived. Otherwise, blank.	S
	CHAR(1) NOT NULL WITH DEFAULT	Not used	N
REFERENCESAUTH	CHAR(1) NOT NULL WITH DEFAULT	Whether the GRANTEE can create or drop referential constraints in which the table is a parent. blank Privilege not held G Privilege held with the GRANT option Y Privilege held without the GRANT option	G
REFCOLS	CHAR(1) NOT NULL WITH 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

SYSIBM.SYSTABAUTH

Column Name	Data Type	Description	Use
GRANTEDTS	TIMESTAMP NOT NULL WITH DEFAULT	Time when the GRANT statement was executed.	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 NOT NULL	Partition number; 0 if table space is not partitioned.	G
TSNAME	CHAR(8) NOT NULL	Name of the table space.	G
DBNAME	CHAR(8) NOT NULL	Name of the database containing the table space.	G
IXNAME	VARCHAR(18) NOT NULL	Name of the partitioned index. This column is blank if the table space is not partitioned.	G
IXCREATOR	CHAR(8) NOT NULL	Authorization ID of the owner of the partitioned index. This column is blank if the table space is not partitioned.	G
# PQTY	INTEGER 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 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
STORATYPE	CHAR(1) NOT NULL	Type of storage allocation: E Explicit (storage group not used) I Implicit (storage group used)	G
STORNAME	CHAR(8) NOT NULL	Name of storage group used for space allocation. Blank if storage group not used.	G
VCATNAME	CHAR(8) NOT NULL	Name of integrated catalog facility catalog used for space allocation.	G
CARD	INTEGER NOT NULL	Number of rows in the table space or partition. The value is 2147483647 if the number of rows is greater than or equal to 2147483647. The value is -1 if statistics have not been gathered.	G
FARINDREF	INTEGER NOT NULL	Number of rows that have been relocated far from their original page. The value is -1 if statistics have not been gathered.	S
NEARINDREF	INTEGER NOT NULL	Number of rows that have been relocated near their original page. The value is -1 if statistics have not been gathered.	S
PERCACTIVE	SMALLINT NOT NULL	Percentage of space occupied by rows of data from active tables. The value is -1 if statistics have not been gathered.	S
PERCDROP	SMALLINT NOT NULL	Percentage of space occupied by rows of dropped tables. The value is -1 if statistics have not been gathered. 0 for segmented table spaces.	S
IBMREQD	CHAR(1) NOT NULL	Whether the row came from the basic machine-readable material (MRM) tape: N No Y Yes	G
LIMITKEY	VARCHAR(512) NOT NULL	The high value of the partition in external format. 0 if the table space is not partitioned.	G

SYSIBM.SYSTABLEPART

Column Name	Data Type	Description	Use
FREEPAGE	SMALLINT NOT NULL	The number of pages loaded before a page is left as free space.	G
PCTFREE	SMALLINT NOT NULL	The percentage of each page left as free space.	G
CHECKFLAG	CHAR(1) NOT NULL WITH DEFAULT	C the table space partition is in a check pending state and there are rows in the table that can violate referential constraints, table check constraints, or both. blank The table space is not a partition, or does not contain rows that may violate referential constraints, table check constraints, or both.	G
	CHAR(4) NOT NULL WITH DEFAULT FOR BIT DATA	Not used	N
SPACE	INTEGER NOT NULL WITH DEFAULT	Number of kilobytes of DASD storage allocated to the table space partition, as determined by the last execution of the STOSPACE 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).	G
#			
#			
#			
#			
#			
COMPRESS	CHAR(1) NOT NULL WITH DEFAULT	Indicates the following: <ul style="list-style-type: none"> For a table space partition, whether the COMPRESS attribute for the partition is YES. For a nonpartitioned table space, whether the COMPRESS attribute is YES for the table space. Values for the column can be: blank No compression Y Compression is defined for the table space	G
PAGESAVE	SMALLINT NOT NULL WITH 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 NOT NULL WITH 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
GBPCACHE	CHAR(1) NOT NULL WITH DEFAULT	Group buffer pool cache option specified for this table space or table space partition. blank Only changed pages are cached in the group buffer pool. A Changed and unchanged pages are cached in the group buffer pool.	G

Column Name	Data Type	Description	Use
CHECKRID5B	CHAR(5) NOT NULL WITH DEFAULT	Blank if the table or partition is not in a check pending state (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

SYSIBM.SYSTABLES Table

Contains one row for each table, view, or alias.

Column Name	Data Type	Description	Use
NAME	VARCHAR(18) NOT NULL	Name of the table, view, or alias.	G
CREATOR	CHAR(8) NOT NULL	Authorization ID of the owner of the table, view, or alias.	G
TYPE	CHAR(1) NOT NULL	Type of object: A Alias G Temporary table T Table V View	G
DBNAME	CHAR(8) 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 DSNDDB06.	G
TSNAME	CHAR(8) 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 SYSPKAGE. For a view of a view, the value is SYSVIEWS. For an alias, it is SYSDBAUT.	G
DBID	SMALLINT NOT NULL	Internal identifier of the database; 0 if the row describes a view, an alias, or a temporary table.	S
OBID	SMALLINT NOT NULL	Internal identifier of the table; 0 if the row describes a view, an alias, or a temporary table.	S
COLCOUNT	SMALLINT NOT NULL	Number of columns in the table or view. The value is 0 if the row describes an alias.	G
EDPROC	CHAR(8) 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) 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) NOT NULL	Whether RESTRICT ON DROP applies: blank No Y Yes. Neither the table nor any table space or database containing the table can be dropped.	G
	INTEGER NOT NULL	Not used	N
	INTEGER NOT NULL	Not used	N
NPAGES	INTEGER 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, or a temporary table. This is an updatable column.	S
PCTPAGES	SMALLINT 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, an alias, or a temporary table. This is an updatable column.	S

Column Name	Data Type	Description	Use
IBMREQD	CHAR(1) NOT NULL	Whether the row came from the basic machine-readable material (MRM) tape: N no Y Yes B V1R3 dependency indicator; not from MRM tape C V2R1 dependency indicator; not from MRM tape D V2R2 dependency indicator; not from MRM tape E V2R3 dependency indicator; not from MRM tape F V3R1 dependency indicator; not from MRM tape G V4 dependency indicator; not from MRM tape H V5 dependency indicator; not from MRM tape	G
REMARKS	VARCHAR(254) NOT NULL	A character string provided by the user with the COMMENT ON statement.	G
PARENTS	SMALLINT NOT NULL	The number of relationships in which the table is a dependent. The value is 0 if the row describes a view, an alias, or a temporary table.	G
CHILDREN	SMALLINT NOT NULL	The number of relationships in which the table is a parent. The value is 0 if the row describes a view, an alias, or a temporary table.	G
KEYCOLUMNS	SMALLINT NOT NULL	The number of columns in the table's primary key. The value is 0 if the row describes a view, an alias, or a temporary table.	G
RECLENGTH	SMALLINT NOT NULL	For user tables, the maximum length of any record in the table. Length is 8+N+L, where: <ul style="list-style-type: none"> The number 8 accounts for the header (6 bytes) and the id map entry (2 bytes). N is 10 if the table has an edit procedure, or 0 otherwise. L is the sum of the maximum column lengths. In determining a column's maximum length, add a byte for the null indicator if the column allows nulls. Add 2 bytes for its length indicator if the column has a varying-length data type (for example, VARCHAR). For more on column lengths, see "Data Types" on page 57. The value is 0 if the row describes a view or alias. For maximum row and record sizes, see "Maximum record size" on page 324.	G
STATUS	CHAR(1) NOT NULL	I The definition of the table is incomplete because it lacks a parent index. X Table has a parent index. blank Table has no parent index, or is a catalog table, or the row describes a view or alias.	G
KEYOBID	SMALLINT 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) NOT NULL	The label as given by a LABEL ON statement; otherwise an empty string.	G
CHECKFLAG	CHAR(1) NOT NULL WITH DEFAULT	C The table space containing the table is in a check pending state and there are rows in the table that can violate referential constraints, table check constraints, or both. blank The table contains no rows that violate referential constraints, table check constraints, or both; or the row describes a view, alias, or temporary table.	G
	CHAR(4) NOT NULL WITH DEFAULT FOR BIT DATA	Not used	N

SYSIBM.SYSTABLES

Column Name	Data Type	Description	Use
AUDITING	CHAR(1) NOT NULL WITH DEFAULT	Value of the audit option: A AUDIT ALL C AUDIT CHANGE blank AUDIT NONE, or the row describes a view, an alias, or a temporary table.	G
CREATEDBY	CHAR(8) NOT NULL WITH DEFAULT	Primary authorization ID of the user who created the table, view, or alias.	G
LOCATION	CHAR(16) NOT NULL WITH DEFAULT	The 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) NOT NULL WITH DEFAULT	For an alias, the authorization ID of the owner of the referred to table or view; blank otherwise.	G
TBNAME	VARCHAR(18) NOT NULL WITH DEFAULT	For an alias, the name for the referred to table or view; blank otherwise.	G
CREATEDTS	TIMESTAMP NOT NULL WITH DEFAULT	Time when the CREATE statement was executed for the table, view, or alias	G
ALTEREDTS	TIMESTAMP NOT NULL WITH 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
DATAcapture	CHAR(1) NOT NULL WITH DEFAULT	Records the value of the DATA CAPTURE option for a table. blank No Y Yes For a temporary table, DATAcapture is always blank.	G
RBA1	CHAR(6) NOT NULL WITH DEFAULT 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 temporary table. In a data sharing environment, RBA1 is the LRSN (Log Record Sequence Number) value.	S
RBA2	CHAR(6) NOT NULL WITH DEFAULT 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 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 NOT NULL WITH 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 the row describes a view, an alias, or a temporary table, or statistics have not been gathered. This is an updatable column.	S
STATSTIME	TIMESTAMP NOT NULL WITH 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 temporary table, the value of STATSTIME is always the default value. This is an updatable column.	G
CHECKS	SMALLINT NOT NULL WITH DEFAULT	The number of check constraints defined on the table. The value is 0 if the row describes a view, an alias, or a temporary table, or if no constraints are defined on the table.	G
CARDF	FLOAT NOT NULL WITH DEFAULT -1	Total number of rows in the table. The value is -1 if statistics have not been gathered or the row describes a view, alias, or temporary table. This is an updatable column.	S

Column Name	Data Type	Description	Use
CHECKRID5B	CHAR(5) NOT NULL WITH DEFAULT	Blank if the table or partition is not in a check pending state (CHECKFLAG is blank), if the table space is not partitioned, or if the table is a 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'000000000', indicating that any row can violate referential constraints.	S
ENCODING_SCHEME	CHAR(1) NOT NULL WITH DEFAULT 'E'	Default encoding scheme for tables, views, and local aliases: E EBCDIC A ASCII blank For remote aliases The value will be '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

SYSIBM.SYSTABLESPACE Table

Contains one row for each table space.

Column Name	Data Type	Description	Use
NAME	CHAR(8) NOT NULL	Name of the table space.	G
CREATOR	CHAR(8) NOT NULL	Authorization ID of the owner of the table space.	G
DBNAME	CHAR(8) NOT NULL	Name of the database containing the table space.	G
DBID	SMALLINT NOT NULL	Internal identifier of the database which contains the table space.	S
OBID	SMALLINT NOT NULL	Internal identifier of the table space file descriptor.	S
PSID	SMALLINT NOT NULL	Internal identifier of the table space page set descriptor.	S
BPOOL	CHAR(8) NOT NULL	Name of the buffer pool used for the table space.	G
PARTITIONS	SMALLINT NOT NULL	Number of partitions of the table space; 0 if the table space is not partitioned.	G
LOCKRULE	CHAR(1) NOT NULL	Lock size of the table space: A Any P Page R Row S Table space T Table	G
PGSIZE	SMALLINT NOT NULL	Size of pages in the table space in kilobytes.	G
ERASERULE	CHAR(1) NOT NULL	Whether the data sets are to be erased when dropped. The value is meaningless if the table space is partitioned. N No erase Y Erase	G
STATUS	CHAR(1) NOT NULL	Availability status of the table space: A Available C Definition is incomplete because no partitioned index has been created. P Table space is in a check pending state. S Table space is in a check pending state with the scope less than the entire table space. T Definition is incomplete because no table has been created.	G
IMPLICIT	CHAR(1) NOT NULL	Whether the table space was created implicitly: Y Yes N No	G
NTABLES	SMALLINT NOT NULL	Number of tables defined in the table space.	G
NACTIVE	INTEGER 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
DSETPASS	CHAR(8) NOT NULL	The password for the data sets of the table space.	G

Column Name	Data Type	Description	Use
CLOSERULE	CHAR(1) NOT NULL	Whether the data sets are candidates for closure when the limit on the number of open data sets is reached. Y Yes N No	G
SPACE	INTEGER 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) NOT NULL	Whether the row came from the basic machine-readable material (MRM) tape: N No Y Yes C V2R1 dependency indicator; not from MRM tape F V3R1 dependency indicator; not from MRM tape G V4 dependency indicator; not from MRM tape H V5 dependency indicator; not from MRM tape	G
	VARCHAR(18) NOT NULL	Internal use only	I
	CHAR(8) NOT NULL	Internal use only	I
SEGSIZE	SMALLINT NOT NULL WITH DEFAULT	The 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) NOT NULL WITH DEFAULT	Primary authorization ID of the user who created the table space.	G
STATSTIME	TIMESTAMP NOT NULL WITH 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. 0 Lock escalation does not occur. n n, where n > 0, is the maximum number of locks (row or page locks for the table or table space) an application process can acquire before lock escalation occurs. -1 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) NOT NULL WITH DEFAULT	The type of table space: blank The table space is not large and was not defined with MEMBER CLUSTER. I The table space is not large and was defined with MEMBER CLUSTER. K The table space is large and was defined with MEMBER CLUSTER. L The table space is large and was not defined with MEMBER CLUSTER	G

SYSIBM.SYSTABLESPACE

Column Name	Data Type	Description	Use
CREATEDTS	TIMESTAMP NOT NULL WITH 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 '0001-01-01.00.00.00.000000'.	G
ALTEREDTS	TIMESTAMP NOT NULL WITH 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 '0001-01-01.00.00.00.000000'.	G
ENCODING_SCHEME	CHAR(1) NOT NULL WITH DEFAULT 'E'	Default encoding scheme for the table space. E EBCDIC A ASCII blank For tables spaces in work file databases. The value will be '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 NOT NULL WITH DEFAULT	Default SBCS CCSID for the table space. For databases created in a DB2 release prior to Version 5, the value is 0.	G
DBCS_CCSID	INTEGER NOT NULL WITH DEFAULT	Default DBCS CCSID for the table space. For databases created in a DB2 release prior to Version 5, the value is 0.	G
MIXED_CCSID	INTEGER NOT NULL WITH DEFAULT	Default mixed CCSID for the table space. For databases created in a DB2 release prior to Version 5, the value is 0.	G
MAXROWS	SMALLINT NOT NULL DEFAULT 255	The maximum number of rows that DB2 will place on a data page. The default value is 255.	G
LOCKPART	CHAR(1) NOT NULL WITH DEFAULT	Y LOCKPART YES is specified for the table space. blank LOCKPART NO is specified, or LOCKPART is not specified or not a partitioned table space.	G

SYSIBM.SYSTABSTATS Table

Contains one row for each partition of a partitioned table space.

Column Name	Data Type	Description	Use
CARD	INTEGER NOT NULL	Total number of rows in the partition. This is an updatable column.	S
NPAGES	INTEGER NOT NULL	Total number of pages on which rows of the partition appear. This is an updatable column.	S
PCTPAGES	SMALLINT NOT NULL	Percentage of total active pages in the partition that contain rows of the table. This is an updatable column.	S
NACTIVE	INTEGER NOT NULL	Number of active pages in the partition. This is an updatable column.	S
PCTROWCOMP	SMALLINT 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. This is an updatable column.	S
STATSTIME	TIMESTAMP NOT NULL	If RUNSTATS updated the statistics, the date and time when the last invocation of RUNSTATS updated the statistics. This is an updatable column.	G
IBMREQD	CHAR(1) NOT NULL	Whether the row came from the basic machine-readable material (MRM) tape. N No Y Yes	G
DBNAME	CHAR(8) NOT NULL	Database that contains the table space named in TSNAME.	G
TSNAME	CHAR(8) NOT NULL	Table space that contains the table.	G
PARTITION	SMALLINT NOT NULL	Partition number of the table space that contains the table.	G
OWNER	CHAR(8) NOT NULL	Authorization ID of the owner of the table.	G
NAME	VARCHAR(18) NOT NULL	Name of the table.	G

SYSIBM.SYSUSERAUTH Table

Records the system privileges held by users.

Column Name	Data Type	Description	Use
GRANTOR	CHAR(8) NOT NULL	Authorization ID of the user who granted the privileges.	G
GRANTEE	CHAR(8) NOT NULL	Authorization ID of the user that holds the privilege. Could also be PUBLIC for a grant to PUBLIC.	G
	CHAR(12) NOT NULL	Internal use only	I
DATEGRANTED	CHAR(6) NOT NULL	Date the privileges were granted; in the form <i>yymmdd</i> .	G
TIMEGRANTED	CHAR(8) NOT NULL	Time the privileges were granted; in the form <i>hhmmssth</i> .	G
	CHAR(1) NOT NULL	Not used	N
AUTHHOWGOT	CHAR(1) 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.	G
		blank Not applicable C DBCTL D DBADM L SYCTRL M DBMAINT S SYSADM	
	CHAR(1) NOT NULL	Not used	N
BINDADDAUTH	CHAR(1) NOT NULL	Whether the GRANTEE can use the BIND subcommand with the ADD option:	G
		blank Privilege is not held G Privilege is held with the GRANT option Y Privilege is held without the GRANT option	
BSDSAUTH	CHAR(1) NOT NULL	Whether the GRANTEE can issue the RECOVER BSDS command:	G
		blank Privilege is not held G Privilege is held with the GRANT option Y Privilege is held without the GRANT option	
CREATEDBAAUTH	CHAR(1) NOT NULL	Whether the GRANTEE can create databases and automatically receive DBADM authority over the new databases:	G
		blank Privilege is not held G Privilege is held with the GRANT option Y Privilege is held without the GRANT option	
CREATEDBCAUTH	CHAR(1) NOT NULL	Whether the GRANTEE can execute the CREATE DATABASE statement to create new databases and automatically receive DBCTRL authority over the new databases:	G
		blank Privilege is not held G Privilege is held with the GRANT option Y Privilege is held without the GRANT option	
CREATESGAUTH	CHAR(1) NOT NULL	Whether the GRANTEE can execute the CREATE STOGROUP statement to create new storage groups:	G
		blank Privilege is not held G Privilege is held with the GRANT option Y Privilege is held without the GRANT option	

Column Name	Data Type	Description	Use
DISPLAYAUTH	CHAR(1) NOT NULL	Whether the GRANTEE can use the DISPLAY commands: blank Privilege is not held G Privilege is held with the GRANT option Y Privilege is held without the GRANT option	G
RECOVERAUTH	CHAR(1) NOT NULL	Whether the GRANTEE can use the RECOVER INDOUBT command: blank Privilege is not held G Privilege is held with the GRANT option Y Privilege is held without the GRANT option	G
STOPALLAUTH	CHAR(1) NOT NULL	Whether the GRANTEE can use the STOP command: blank Privilege is not held G Privilege is held with the GRANT option Y Privilege is held without the GRANT option	G
STOSPACEAUTH	CHAR(1) NOT NULL	Whether the GRANTEE can use the STOSPACE utility: blank Privilege is not held G Privilege is held with the GRANT option Y Privilege is held without the GRANT option	G
SYSADMAUTH	CHAR(1) NOT NULL	Whether the GRANTEE has system administration authority: blank Privilege is not held G Privilege was granted with the GRANT option Y Privilege was granted without the GRANT option GRANTEE has the privilege with the GRANT option for a value of either Y or G.	G
SYSOPRAUTH	CHAR(1) NOT NULL	Whether the GRANTEE has system operator authority: blank Privilege is not held G Privilege is held with the GRANT option Y Privilege is held without the GRANT option	G
TRACEAUTH	CHAR(1) NOT NULL	Whether the GRANTEE can issue the START TRACE and STOP TRACE commands: blank Privilege is not held G Privilege is held with the GRANT option Y Privilege is held without the GRANT option	G
IBMREQD	CHAR(1) NOT NULL	Whether the row came from the basic machine-readable material (MRM) tape: N No Y Yes	G
MON1AUTH	CHAR(1) NOT NULL WITH DEFAULT	Whether the GRANTEE can obtain IFC serviceability data. blank Privilege is not held G Privilege is held with the GRANT option Y Privilege is held without the GRANT option	G
MON2AUTH	CHAR(1) NOT NULL WITH DEFAULT	Whether the GRANTEE can obtain IFC data. blank Privilege is not held G Privilege is held with the GRANT option Y Privilege is held without the GRANT option	G
CREATEALIASAUTH	CHAR(1) NOT NULL WITH DEFAULT	Whether the GRANTEE can execute the CREATE ALIAS statement. blank Privilege not held G Privilege held with the GRANT option Y Privilege held without the GRANT option	G

SYSIBM.SYSUSERAUTH

Column Name	Data Type	Description	Use
SYSCTRLAUTH	CHAR(1) NOT NULL WITH DEFAULT	Whether the GRANTEE has SYSCTRL authority: blank Privilege is not held G Privilege is held with the GRANT option Y Privilege is held without the GRANT option GRANTEE has the privilege with the GRANT option for a value of either Y or G.	G
BINDAGENTAUTH	CHAR(1) NOT NULL WITH DEFAULT	Whether the GRANTEE has BINDAGENT privilege: blank Privilege is not held G Privilege is held with the GRANT option Y Privilege is held without the GRANT option See "GRANT (System Privileges)" on page 409 for a description of the BINDAGENT privilege.	G
ARCHIVEAUTH	CHAR(1) NOT NULL WITH DEFAULT	Whether the GRANTEE is privileged to use the ARCHIVE LOG command: blank Privilege is not held G Privilege is held with the GRANT option Y Privilege is held without the GRANT option	G
	CHAR(1) NOT NULL WITH DEFAULT	Not used	N
	CHAR(1) NOT NULL WITH DEFAULT	Not used	N
GRANTEDTS	TIMESTAMP NOT NULL WITH 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) NOT NULL WITH DEFAULT	Whether the GRANTEE has CREATETMTABAUTH privilege: blank Privilege is not held G Privilege is held with the GRANT option Y Privilege is held without the GRANT option	G

SYSIBM.SYSVIEWDEP Table

Records the dependencies of views on tables and other views.

Column Name	Data Type	Description	Use
BNAME	VARCHAR(18) NOT NULL	Name of a table or view on which the view is dependent.	G
BCREATOR	CHAR(8) NOT NULL	Authorization ID of the owner of BNAME.	G
BTYPE	CHAR(1) NOT NULL	Type of object BNAME: T Table V View	G
DNAME	VARCHAR(18) NOT NULL	Name of the view.	G
DCREATOR	CHAR(8) NOT NULL	Authorization ID of the owner of the view.	G
IBMREQD	CHAR(1) NOT NULL	Whether the row came from the basic machine-readable material (MRM) tape: N No Y Yes	G

SYSIBM.SYSVIEWS Table

Contains one or more rows for each view.

Column Name	Data Type	Description	Use
NAME	VARCHAR(18) NOT NULL	Name of the view.	G
CREATOR	CHAR(8) NOT NULL	Authorization ID of the owner of the view.	G
SEQNO	SMALLINT 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) NOT NULL	Whether the WITH CHECK OPTION clause was specified in the CREATE VIEW statement: N No C Yes with the <i>cascaded</i> semantic Y Yes with the <i>local</i> semantic The value is N if the view has no WHERE clause.	G
IBMREQD	CHAR(1) NOT NULL	Whether the row came from the basic machine-readable material (MRM) tape: N No Y Yes B V1R3 dependency indicator; not from MRM tape C V2R1 dependency indicator; not from MRM tape D V2R2 dependency indicator; not from MRM tape E V2R3 dependency indicator; not from MRM tape F V3R1 dependency indicator; not from MRM tape G V4 dependency indicator; not from MRM tape H V5 dependency indicator; not from MRM tape	G
TEXT	VARCHAR(254) NOT NULL	The text or portion of the text of the CREATE VIEW statement.	G

SYSIBM.SYSVOLUMES Table

Contains one row for each volume of each storage group.

Column Name	Data Type	Description	Use
SGNAME	CHAR(8) NOT NULL	The name of the storage group.	G
SGCREATOR	CHAR(8) NOT NULL	Authorization ID of the owner of the storage group.	G
VOLID	CHAR(6) NOT NULL	The serial number of the volume or * if SMS-managed.	G
IBMREQD	CHAR(1) NOT NULL	Whether the row came from the basic machine-readable material (MRM) tape: N No Y Yes	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) NOT NULL	How the row is to be used: O For outbound translation. I For inbound translation and “come from” checking.	G
AUTHID	CHAR(8) NOT NULL WITH DEFAULT	Authorization ID to be translated. Applies to any authorization ID if blank.	G
LINKNAME	CHAR(8) 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. If a nonblank LINKNAME is specified, one or both of the following statements must be true: <ul style="list-style-type: none"> • 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. • 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. Inbound name translation and “come from” checking are not performed for TCP/IP clients.	G
NEWAUTHID	CHAR(8) NOT NULL WITH DEFAULT	Translated value of AUTHID. Blank specifies no translation.	G
PASSWORD	CHAR(8) NOT NULL WITH 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) NOT NULL WITH DEFAULT 'N'	Whether the row came from the basic machine-readable material (MRM) tape: N No Y Yes	G

Appendix E. SQL Reserved Words

The following words cannot be used as ordinary identifiers in any context where they could also be interpreted as SQL keywords. For example, COUNT cannot be used as a column name in a SELECT statement. Each word can, however, be used as an ordinary identifier in any other context; for example, in statements where the word can never be an SQL keyword. The word can also be used, as a delimited identifier, in contexts where it otherwise could not be used. Assume, for example, that double quotation marks (") are used to begin and end delimited identifiers. Then "COUNT" can appear as a column name in a SELECT statement.

#	ADD	DESCRIPTOR	JOIN	PROCEDURE
#	ALL	DISTINCT	KEY	PSID
#	ALLOCATE	DO	LEAVE	REFERENCES
#	ALTER	DOUBLE	LEFT	RENAME
#	AND	DROP	LIKE	REPEAT
#	ANY	EDITPROC	LOCAL	RIGHT
#	AS	ELSE	LOCATOR	SECOND
#	ASSOCIATE	ELSEIF	LOCATORS	SECONDS
#	AUDIT	END	LOCKMAX	SECQTY
#	BETWEEN	END-EXEC ¹	LOCKSIZE	SELECT
#	BUFFERPOOL	ERASE	LOOP	SET
	BY	ESCAPE	MICROSECOND	SOME
	CALL	EXCEPT	MICROSECONDS	STOGROUP
	CASE	EXECUTE	MINUTE	SUBPAGES
#	CAPTURE	EXISTS	MINUTES	SYNONYM
#	CASCADE	EXIT	MONTH	TABLE
#	CCSID	FIELDPROC	MONTHS	TABSPACE
#	CHAR	FOR	NO	THEN
#	CHARACTER	FROM	NOT	TO
#	CHECK	FULL	NULL	UNDO
	CLUSTER	GO	NUMPARTS	UNION
#	COLLECTION	GOTO	OBID	UNIQUE
#	COLUMN	GRANT	OF	UNTIL
#	CONCAT	GROUP	ON	UPDATE
#	CONDITION	HANDLER	OPTIMIZE	USER
#	CONSTRAINT	HAVING	OR	USING
#	CONTINUE	HOUR	ORDER	VALIDPROC
#	COUNT	HOURS	OUT	VALUES
#	CURRENT	IF	OUTER	VCAT
	CURRENT_DATE	IMMEDIATE	PACKAGE	VIEW
	CURRENT_TIME	IN	PART	VOLUMES
#	CURRENT_TIMESTAMP	INDEX	PIECESIZE	WHEN
#	CURSOR	INNER	PLAN	WHERE
#	DATABASE	INOUT	PRECISION	WHILE
	DAY	INSERT	PRIQTY	WITH
	DAYS	INTO	PRIVILEGES	YEAR
#	DEFAULT	IS	PROGRAM	YEARS
#	DELETE	ISOBID		

IBM SQL has additional reserved words. These additional reserved words are not enforced by DB2 for OS/390, but we suggest that you do not use them as ordinary

¹ COBOL only

SQL Reserved Words

identifiers in names that will have a continuing use. See *IBM SQL Reference* for a list of these words.

SQL Reserved Words

Appendix F. DB2 Objects Required by the DB2 for OS/390 SQL Procedure Processor

The DB2 for OS/390 SQL procedure processor (DSNTPSMP) uses the tables and indexes that are described in the following sections. You can create these objects by customizing and running job DSNTIJSQ, which is in data set DSN510.SDSNSAMP. DSNTIJSQ creates the objects in database DSNPSM and table space DSNPSM.

Table Spaces and Indexes

625 shows the table spaces to which the SQL procedure tables are assigned, and which indexes are defined on the tables.

Table 40. Table spaces and indexes for SQL procedure tables

TABLE SPACE DSNPSM. ...	TABLE SYSIBM. ...	Page	INDEX SYSIBM. ...	INDEX FIELDS
DSNPSM	SYSPSM	625	DSNPSMX1	PROCEDURENAME
			DSNPSMX2	SCHEMA PROCEDURENAME SEQNO
	SYSPSMOPTS	625	DSNPSMOX1	SCHEMA PROCEDURENAME

The SQL Procedure Source Table (SYSIBM.SYSPSM)

SYSIBM.SYSPSM is used by the SQL procedure processor and IBM DB2 Stored Procedure Builder to hold the source code for a stored procedure.

SYSIBM.SYSPSM contains at least one row for each SQL procedure that is prepared by the SQL procedure processor or SQL Procedure Builder. The number of rows that represent an SQL procedure is

$\text{CEILING}(n/3800)$

n is the number of bytes in the SQL procedure source statement.

Column Name	Data Type	Description	Use
SCHEMA	CHAR(8)	Schema of the SQL procedure. Blank for SQL procedures created before DB2 Version 6.	G
PROCEDURENAME	CHAR(18) NOT NULL	Name of the SQL procedure.	G
SEQNO	SMALLINT NOT NULL	Number of the SQL statement piece in PROCCREATESTMT. SEQNO is between 1 and $\text{CEILING}(n/3800)$, where n is the number of bytes in the SQL procedure source statement.	G
PSMDATE	DATE NOT NULL	The date on which the SQL procedure was created.	G
PSMTIME	TIME NOT NULL	The time at which the SQL procedure was created.	G
PSMTIME	TIME NOT NULL	The time at which the SQL procedure was created.	G

Column Name	Data Type	Description	Use
PROCCREATESTMT	VARCHAR(3800) NOT NULL	All or part of an SQL procedure source statement. If the SQL procedure statement is more than 3800 bytes, this field contains the portion of the source statement indicated by SEQNO.	G

The SQL Procedure Options Table (SYSIBM.SYSPSMOPTS)

SYSIBM.SYSPSMOPTS is used by the SQL procedure processor and IBM DB2 Stored Procedure Builder to hold the program preparation options for an SQL procedure. SYSIBM.SYSPSMOPTS contains one row for each SQL procedure that is prepared by the SQL procedure processor or SQL Procedure Builder.

Column Name	Data Type	Description	Use
SCHEMA	CHAR(8)	Schema of the SQL procedure. Blank for SQL procedures created before DB2 Version 6.	G
PROCEDURENAME	CHAR(18) NOT NULL	Name of the SQL procedure.	G
BUILDSHEMA	CHAR(8)	The schema name that is the qualifier for the procedure name that is specified in the BUILDNAME column. The schema name is SYSPROC.	G
BUILDNAME	CHAR(18)	A procedure name that is associated with stored procedure DSNTPSMP. Users of DSNTPSMP might create several stored procedure definitions for DSNTPSMP so that they can run DSNTPSMP in different WLM environments. The caller specifies the environment in which DSNTPSMP runs by specifying the procedure name that is associated with that environment in the SQL CALL statement.	G
BUILDOWNER	CHAR(8)	The authorization ID that was used to create the SQL procedure.	G
PRECOMPILE_OPTS	VARCHAR(255)	The options that were specified in the <i>precompiler-options</i> parameter in the most recent invocation of DSNTPSMP for the SQL procedure specified in this row.	G
COMPILE_OPTS	VARCHAR(255)	The options that were specified in the <i>compiler-options</i> parameter in the most recent invocation of DSNTPSMP for the SQL procedure specified in this row.	G
PRELINK_OPTS	VARCHAR(255)	The options that were specified in the <i>prelink-edit-options</i> parameter in the most recent invocation of DSNTPSMP for the SQL procedure specified in this row.	G
LINK_OPTS	VARCHAR(255)	The options that were specified in the <i>link-edit-options</i> parameter in the most recent invocation of DSNTPSMP for the SQL procedure specified in this row.	G
BIND_OPTS	VARCHAR(1024)	The options that were specified in the <i>bind-options</i> parameter in the most recent invocation of DSNTPSMP for the SQL procedure specified in this row.	G
SOURCEDSN	VARCHAR(255)	If the SQL procedure source code that is input to DSNTPSMP is stored in a data set, the name of that data set.	G

Temporary Table **SYSIBM.SYSPSMOUT**

SYSIBM.SYSPSMOUT is used by the SQL procedure processor and IBM DB2 Stored Procedure Builder to hold error information that is returned in a result set.

This SQL statement creates the temporary table:

```
CREATE GLOBAL TEMPORARY TABLE SYSIBM.SYSPSMOUT
  (STEP          VARCHAR(16),
   FILE         VARCHAR(8),
   SEQN         INTEGER,
   LINE         VARCHAR(255) )
CCSID EBCDIC;
```

Bibliography

Glossary

The following terms and abbreviations are defined as they are used in the DB2 library. If you do not find the term you are looking for, refer to the index or to *Dictionary of Computing*.

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 *Messages and Codes*.

abnormal end of task (abend). Termination of a task, a job, or a subsystem because of an error condition that cannot be resolved during execution by recovery facilities.

access method services. A utility program that defines and manages VSAM data sets (or files).

access path. The path used to get to data specified in SQL statements. An access path can involve an index or a sequential search.

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 will 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 is failed or quiesced.

alias. An alternate 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 originating at the local DB2 subsystem that may access data at a remote DB2 subsystem.

allocated cursor. A cursor defined for stored procedure results sets by using `ALLOCATE CURSOR`.

already verified. An LU 6.2 security option which allows DB2 to provide the user's verified authorization ID when allocating a conversation. The user is not validated by the partner DB2.

ambiguous cursor. A database cursor that is not defined with either the clauses `FOR FETCH ONLY` or `FOR UPDATE OF`, 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 SQL statements `PREPARE` or `EXECUTE IMMEDIATE`.

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 used to define DB2 to VTAM as an application program using SNA LU 6.2 protocols.

application. A program or set of programs that perform a task; for example, a payroll application.

application plan. The control structure produced during the bind process and used by DB2 to process SQL statements encountered 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 program interface (API). A functional interface supplied by the operating system or by a separately orderable licensed program that allows an application program written in a high-level language to use specific data or functions of the operating system or licensed program.

application requester (AR). See *requester*.

application server. See *server*.

AR. application requester. See *requester*.

archive log. The portion of the DB2 log that contains log records that have been copied from the active log.

AS. Application server. See *server*.

ASCII. An encoding scheme used to represent strings in many environments, typically on PCs and workstations. Contrast with *EBCDIC*.

attachment facility • check clause

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 are allowed. It can represent an individual, an organizational group, or a function, but DB2 does not determine this representation.

B

base table. A table created by the SQL CREATE TABLE statement that is used to hold persistent data. Contrast with *result table* and *temporary table*.

basic predicate. A predicate that compares two values.

binary integer. A basic data type that can be further classified as small integer or large integer.

bind. The process by which the output from the DB2 precompiler is converted to a usable control structure called a package or an application plan. During the process, access paths to the data are selected and some authorization checking is performed.

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. Contrast with *dynamic bind*.

bit data. Data that is not associated with a coded character set.

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 reserved to satisfy the buffering requirements for one or more table spaces or indexes.

built-in function. Scalar function or column 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 program interface (API) for database access, which is an alternative to using embedded SQL. In contrast to embedded SQL, DB2 CLI 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 enforcement of referential constraints by DB2 when it deletes all descendent rows of a deleted parent row.

CASE expression. Allows an expression to be selected based on the evaluation of one or more conditions.

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. See *communications database*.

CDRA. Character data representation architecture.

character data representation architecture (CDRA). An architecture used to achieve consistent representation, processing, and interchange of string data.

character set. A defined set of characters.

check clause. An extension to the SQL CREATE TABLE and SQL ALTER TABLE statements that specifies a table check constraint.

check constraint. See *table check constraint*.

check integrity. The condition that exists when each row in a table conforms to the table check constraints defined on that table. Maintaining check integrity requires enforcing 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 it can contain rows that violate referential constraints, table check constraints, or both.

checkpoint. A point at which DB2 records internal status information on the DB2 log that would be used in the recovery process if DB2 should abend.

CICS. Represents (in this publication) CICS/MVS and CICS/ESA.

CICS/MVS: Customer Information Control System/Multiple Virtual Storage.

CICS/ESA: Customer Information Control System/Enterprise Systems Architecture.

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. See *call level interface*.

client. See *requester*.

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.

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. An SQL operation that derives its result from a collection of values across one or more rows. Contrast with *scalar function*.

"come from" checking. An LU 6.2 security option which 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. Distinct from an SQL statement.

commit. The operation that ends a unit of work by releasing locks so that the database changes 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 multi-site 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 =, >, <) 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. The existence of a communication path between two partner LUs that allows information to be exchanged (for example, two DB2s connected and communicating by way of a conversation).

consistency token. A timestamp 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*, *uniqueness constraint*, and *table check constraint*.

conversation. (1) A VTAM term for a dialog between two application processes, on different DB2 subsystems, that is specified by a particular *session name*, *mode name*, and *LU name*. (2) An LU 6.2

correlated subquery • DB2 command

security option which allows DB2 to require the user's authorization ID and password when allocating a conversation to a partner DB2. The user is validated by the partner DB2.

correlated subquery. A subquery (part of a WHERE or HAVING clause) applied to a row or group of rows of a table or view named in an outer sub-SELECT statement.

correlation ID. An identifier 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.

current data. Data within a host structure that is current with (identical to) the data within the base table.

cursor. A named control structure used by an application program 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 is a descendent of the last. 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 accessing data at the local subsystem on behalf of a remote subsystem.

database administrator (DBA). An individual responsible for the design, development, operation, safeguarding, maintenance, and use of a database.

database descriptor (DBD). An internal representation of DB2 database definition which reflects the data definition found in the DB2 catalog. The objects 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 access to the data stored within it.

database request module (DBRM). A data set member created by the DB2 precompiler that contains information about SQL statements. DBRMs are used in the bind process.

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 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 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.

DBCS. Double-byte character set.

DBD. Database descriptor.

DBID. Database identifier.

DBMS. Database management system.

DBRM. Database request module.

DB2 catalog. Tables maintained by DB2 that 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.

DB2I. DATABASE 2 Interactive.

DB2 private protocol access. A method of accessing distributed data by which you can direct a query to another DB2 system by using an alias or a three-part name to identify the DB2 subsystems at which the statements are executed. Contrast with *DRDA access*.

DB2 private protocol connection. A DB2 private connection of the application process. See also *private connection*.

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.

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. 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.

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.

delimited identifier. A sequence of characters enclosed within 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 shown in syntax diagrams.

dependent. An object (row, table, or table space) is a dependent if it 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 dependent in at least one referential constraint.

descendent. An object is a descendent of another object if it is a dependent of the object, or if it is the dependent of a descendent of that object.

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

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

direct access storage device (DASD). A device in which access time is independent of the location of the data.

directory. The system database that contains internal objects such as database descriptors and skeleton cursor tables.

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.

domain name. The name used by TCP/IP applications to 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 set (DBCS). A set of characters used by national languages such as Japanese and Chinese that have more symbols than can be represented by a single byte. Each character is two bytes in length, and therefore requires special hardware to be displayed or printed.

double-precision floating point number. A 64-bit approximate representation of a real number.

DRDA. Distributed relational database architecture.

DRDA access. A method of accessing distributed data by which you can explicitly connect to another location, using an SQL statement, to execute packages that have been previously bound at that location. The SQL

DSN • graphic string

CONNECT statement is used to identify application servers, and SQL statements are executed using packages that were previously bound at those servers. Contrast with *DB2 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

EBCDIC. Extended binary coded decimal interchange code. An encoding scheme used to represent character data in the MVS, VM, VSE, and OS/400 environments. Contrast with *ASCII*.

EDM pool. A pool of main storage used for database descriptors and application plans.

embedded SQL. SQL statements coded within an application program. See *static SQL*.

encoding scheme. A set of rules to represent character data (ASCII or EBCDIC).

equi-join. A join operation in which the join-condition has the form *expression = expression*.

escape character. The symbol used to enclose an SQL delimited identifier. The escape character is the quotation mark ("), except in COBOL applications, where the symbol (either a quotation mark or an apostrophe) can be assigned by the user.

EUR. IBM European Standards.

exclusive lock. A lock that prevents concurrently executing application processes from reading or changing data. Contrast with *shared 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. Names specified in a FROM clause are exposed or non-exposed. An exposed name is a correlation name or a table or view name for which a correlation name is not specified.

expression. An operand or a collection of operators and operands that yields a single value.

F

failed member state. A state of a member of a data sharing group. A failed member has permanent status recording with XCF, and its task, address space, or MVS system has 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 designed to receive a single value and transform (encode or decode) it in any way the user can specify.

fixed-length string. A character or graphic string whose length is specified and cannot be changed. Contrast with *varying-length string*.

foreign key. A key that is specified in the definition of a referential constraint. Because of the foreign key, the table is a dependent table. The key must have the same number of columns, with the same descriptions, as the primary key of the parent table.

free space. The total unused space in a page, that is, the space not used to store records or control information.

full outer join. The result of a join operation that includes the matched rows of both tables being joined and preserves the unmatched rows of both tables. See also *join*.

function. A scalar function or column function. Same as *built-in function*.

G

GB. Gigabyte (1,073,741,824 bytes).

GBP-dependent. A page set or page set partition status when it is dependent upon the group bufferpool. There is either inter-DB2 read/write interest active for this page set or the page set has changed pages in the group buffer pool that have not yet been castout 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. A coupling facility cache structure 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 either locks or the shared communications area have been lost.

H

host. The set of programs and resources that are available on a given TCP/IP instance.

host identifier. A name declared in the host program.

host language. A programming language in which you can embed SQL statements.

host program. An application program written in a host language that contains embedded SQL statements.

host structure. In an application program, a structure referenced by embedded SQL statements.

host variable. In an application program, an application variable referenced by embedded SQL statements.

I

ICF. Integrated catalog facility.

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) is independent if it 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 used to determine the order of index entries.

index partition. A VSAM data set that is contained within a partitioned index space.

index space. A page set used to store the entries of one index.

indicator variable. A variable 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 this unit of recovery is to be committed or rolled back. At emergency restart, if DB2 does not have the information needed to make this decision, its unit of recovery is *indoubt* until DB2 obtains this information from the coordinator.

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*.

internal resource lock manager (IRLM). An MVS subsystem used by DB2 to control communication and database locking.

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.

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*, *repeatable read*, *uncommitted read*, and *read stability*.

J

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 *full outer join*, *inner join*, *left outer join*, *outer join*, *right outer join*, *equi-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 partitioned table space. A table space that allows the partitioned table it contains to exceed 64 GB of data in either compressed or uncompressed format.

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 being joined, and preserves the unmatched rows of the first table. See also *join*.

L-lock. See *logical lock*.

local. Refers to any object maintained by the local DB2 subsystem. A *local table*, for example, is a table 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 name. The name by which DB2 refers to a particular DB2 subsystem in a network of subsystems. Contrast with *LU name*.

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 or page lock to a table space lock because the number of page locks 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 concurrently running programs can have to a resource held by a DB2 lock.

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 table, or a table space.

log. A collection of records that describe the events that occur during DB2 execution and 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. The lock type used by transactions to control intra- and inter-DB2 data concurrency between transactions.

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). In IMS, the processing that 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 DB2 generates and associates with each log record. DB2 also uses the LRSN for page versioning. The LRSNs generated by a given DB2 data sharing group 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 will 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.

LRSN. See *log record sequence number*.

LU name. From *logical unit name*, 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*.

multi-site 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 subselect 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.

nonpartitioned index. Any index that is not a partitioned index.

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 which evaluates two passed expressions, returning 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

OBID. Data object identifier.

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.

outer join. The result of a join operation that includes the matched rows of both tables being joined and preserves some or all of the unmatched rows of the tables being joined. See also *join*.

P

package. Also *application package*. An object containing a set of SQL statements that have been bound statically and that are available for processing.

package list. An ordered list of package names that may be used to extend an application plan.

package name. The name given an object 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 (4KB or 32KB) or index space (4KB). In a table space, a page contains one or more rows of a table.

page set. A table space or index space consisting of pages that are either 4KB or 32KB in size. Each page set is made from 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 was a static SQL statement.

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 • quiesced member state

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 gigabytes, depending upon 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, called members, each of which can contain a program, part of a program, or data. Synonymous with program library.

partitioned table space. A table space subdivided into parts (based upon index key range), each of which may be processed by utilities independently.

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 to execute it.

plan name. The name of an application plan.

point of consistency. A time when all recoverable data an application accesses is consistent with other data. 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. 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.

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 recorded for that ID in the DB2 catalog.

process. A general term for a unit that depends on the environment, but has the same basic properties in every environment. A process involves the execution of one or more programs, and is the unit to which resources and locks are allocated. The execution of an SQL statement is always associated with some process.

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.

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. OS/VS2 MVS Resource Access Control Facility.

RDB. See *relational database*.

RDBMS. Relational database management system.

RDBNAM. See *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, known as *phantom rows*, that were inserted and committed by a concurrently executing application process.

rebind. To create a new application plan for an application program that has been bound previously. If, for example, you have added an index for a table accessed by your application, 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 their sequence. The information recorded 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 condition that exists when all intended references from data in one column of a table to data in another column of the same or a different table are valid. Maintaining referential integrity requires enforcing referential constraints on all LOAD, RECOVER, INSERT, UPDATE, and DELETE operations.

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. 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 relational database manager that operates consistently across supported IBM systems.

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. Refers to any object maintained by a remote DB2 subsystem; that is, by a DB2 subsystem other than the local one. A *remote view*, for instance, is a view 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 may even operate on the same processor under the same MVS system.

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 submitted to the prepare phase if the participant has modified data and is prepared to commit or roll back.

requester. Also *application requester (AR)*. The source of a request to a remote RDBMS, the system that requests the data.

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.

result set. The set of rows returned to a client application by a stored procedure.

result set locator. A 4-byte value used by DB2 to uniquely identify a query result set returned by a stored procedure.

result table. The set of rows specified by a SELECT statement.

right outer join • SQL communication area (SQLCA)

right outer join. The result of a join operation that includes the matched rows of both tables 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*.

row. The horizontal component of a table. A row consists of a sequence of values, one for each column of the table.

S

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 enclosed in parentheses. See also *column function*.

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.

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.

server. Also *application server (AS)*. The target for a request from a remote RDBMS, the RDBMS that provides the data.

shared lock. A lock that prevents concurrently executing application processes from changing data, but not from reading data.

shift-in character. A special control character (X'0F') used in EBCDIC systems to denote that the following

bytes represent SBCS characters. See *shift-out character*.

shift-out character. A special control character (X'0E') used in EBCDIC systems to denote that the following bytes, up to the next shift-in control character, represent DBCS characters.

short string. A string whose actual length, or a varying-length string whose maximum length, is 255 bytes (127 double-byte characters) or less.

sign-on. A request made on behalf of an individual CICS or IMS application process by an attach 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.

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.

source program. A set of host language statements and SQL statements that is processed by an SQL precompiler.

space. A sequence of one or more blank characters.

special register. A storage area that is defined for a process by DB2 and is used to store information that can be referenced in SQL statements. Examples of special registers are USER, CURRENT DATE, and CURRENT TIME.

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. Structured Query Language.

SQL authorization ID (SQL ID). The authorization ID that is used for checking dynamic SQL statements in some situations.

SQL communication area (SQLCA). A structure used to provide an application program with information about the execution of its SQL statements.

SQL descriptor area (SQLDA). A structure that describes input variables, output variables, or the columns of a result table.

SQL escape character. The symbol used to enclose an SQL delimited identifier. This symbol is the quotation mark ("). See *escape character*.

SQL ID. SQL authorization ID.

SQL return code. Either SQLCODE or SQLSTATE.

SQL string delimiter. A symbol used to enclose an SQL string constant. The SQL string delimiter is the apostrophe ('), except in COBOL applications, in which case the symbol (either an apostrophe or a quotation mark) may be assigned by the user.

SQLCA. SQL communication area.

SQLDA. SQL descriptor area.

SQL/DS. SQL/Data System. Also known as *DB2/VSE & VM*.

SSI. MVS subsystem interface.

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 specified by the statement might change).

storage group. A named set of DASD volumes 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*.

Structured Query Language (SQL). A standardized language for defining and manipulating data in a relational database.

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 RDBMS.

sync point. See *commit point*.

synonym. In SQL, an alternative name for a table or view. Synonyms can only be used to refer to objects at the subsystem in which the synonym is defined.

system administrator. The person having the second highest level of authority within DB2. System administrators make decisions about how DB2 is to be used and implement those decisions by choosing system parameters. They monitor the system and change its characteristics to meet changing requirements and new data processing goals.

system conversation. The conversation that two DB2s must establish to process system messages before any distributed processing can begin.

T

table. A named data object consisting of a specific number of columns and some number of unordered rows. Synonymous with *base table* or *temporary table*.

table check constraint. A user-defined constraint that specifies the values that specific columns of a base table can contain.

table space. A page set used to store the records in one or more tables.

TCP/IP. A network communication protocol used by computer systems 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 created by the SQL CREATE GLOBAL TEMPORARY TABLE statement that is used to hold temporary data. Contrast with *result table* and *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 • VTAM

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). Provides interactive time sharing from remote terminals.

timestamp. A seven-part value that consists of a date and time 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 will be the other half of the conversation.

TSO. Time-sharing option.

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*.

type 2 indexes. A new type of indexes available in Version 4. They differ from *type 1 indexes* in several respects; for example, they are the only indexes allowed on a table space that uses *row locks*.

U

uncommitted read (UR). The isolation level that allows an application to read uncommitted data.

underlying view. The view upon which another view is directly or indirectly defined.

UNION. An SQL operation that combines the results of two select statements. UNION is often used to merge lists of values obtained from several tables.

unique index. An index which ensures that no identical key values are stored in a table.

uniqueness constraint. The rule that no two values in a primary key or key of a unique index can be the same.

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 *multi-site update* operation, a single unit of work can include several *units of recovery*.

UT. Utility-only access.

V

value. The smallest unit of data 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*.

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, or packages.

A version of a program is the source code 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 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.

view. An alternative representation of data from one or more tables. A view can include all or some of the columns 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 CREATE VIEW.

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. MVS Virtual telecommunication access method.

Bibliography

DB2 for OS/390 Version 5

- *Administration Guide*, SC26-8957
- *Application Programming and SQL Guide*, SC26-8958
- *Call Level Interface Guide and Reference*, SC26-8959
- *Command Reference*, SC26-8960
- *Data Sharing: Planning and Administration*, SC26-8961
- *Data Sharing Quick Reference Card*, SX26-3841
- *Diagnosis Guide and Reference*, LY27-9659
- *Diagnostic Quick Reference Card*, LY27-9660
- *Installation Guide*, GC26-8970
- *Application Programming Guide and Reference for Java™*, SC26-9547
- *Licensed Program Specifications*, GC26-8969
- *Messages and Codes*, GC26-8979
- *Reference for Remote DRDA Requesters and Servers*, SC26-8964
- *Reference Summary*, SX26-3842
- *Release Guide*, SC26-8965
- *SQL Reference*, SC26-8966
- *Utility Guide and Reference*, SC26-8967
- *What's New?*, GC26-8971
- *Program Directory*

DB2 PM for OS/390 Version 5

- *Batch User's Guide*, SC26-8991
- *Command Reference*, SC26-8987
- *General Information*, GC26-8982
- *Getting Started on the Workstation*, SC26-8989
- *Master Index*, SC26-8984
- *Messages Manual*, SC26-8988
- *Online Monitor User's Guide*, SC26-8990
- *Report Reference Volume 1*, SC26-8985
- *Report Reference Volume 2*, SC26-8986
- *Program Directory*

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

- *DB2 for OS/400 SQL Programming*, SC41-4611
- *DB2 for OS/400 SQL Reference*, SC41-4612

BASIC

- *IBM BASIC/MVS Language Reference*, GC26-4026
- *IBM BASIC/MVS Programming Guide*, SC26-4027

C/370

- *IBM SAA AD/Cycle C/370 Programming Guide*, SC09-1356
- *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*, SC09-2190

CICS/ESA

- *CICS/ESA Application Programming Guide*, SC33-1169
- *CICS/ESA Application Programming Reference*, SC33-1170
- *CICS/ESA CICS - RACF Security Guide*, SC33-1185
- *CICS/ESA CICS-Supplied Transactions*, SC33-1168
- *CICS/ESA Customization Guide*, SC33-1165
- *CICS/ESA Data Areas*, LY33-6083
- *CICS/ESA Installation Guide*, SC33-1163
- *CICS/ESA Intercommunication Guide*, SC33-1181
- *CICS/ESA Messages and Codes*, SC33-1177
- *CICS/ESA Operations and Utilities Guide*, SC33-1167
- *CICS/ESA Performance Guide*, SC33-1183
- *CICS/ESA Problem Determination Guide*, SC33-1176
- *CICS/ESA Resource Definition Guide*, SC33-1166
- *CICS/ESA System Definition Guide*, SC33-1164
- *CICS/ESA System Programming Reference*, GC33-1171

CICS/MVS

- *CICS/MVS Application Programming Primer*, SC33-0139
- *CICS/MVS Application Programmer's Reference*, SC33-0512
- *CICS/MVS Facilities and Planning Guide*, SC33-0504
- *CICS/MVS Installation Guide*, SC33-0506
- *CICS/MVS Operations Guide*, SC33-0510
- *CICS/MVS Problem Determination Guide*, SC33-0516
- *CICS/MVS Resource Definition (Macro)*, SC33-0509
- *CICS/MVS Resource Definition (Online)*, SC33-0508

IBM C/C++ for MVS/ESA or OS/390

- *IBM C/C++ for MVS/ESA Library Reference*, SC09-1995
- *IBM C/C++ for MVS/ESA Programming Guide*, SC09-1994
- *IBM C/C++ for OS/390 User's Guide*, SC09-2361

IBM COBOL for MVS & VM

- *IBM COBOL for MVS & VM Language Reference*, SC26-4769
- *IBM COBOL for MVS & VM Programming Guide*, SC26-4767

Conversion Guides

- *DBMS Conversion Guide: DATACOM/DB to DB2*, GH20-7564
- *DBMS Conversion Guide: IDMS to DB2*, GH20-7562
- *DBMS Conversion Guide: Model 204 to DB2 or SQL/DS*, GH20-7565
- *DBMS Conversion Guide: VSAM to DB2*, GH20-7566
- *IMS-DB and DB2 Migration and Coexistence Guide*, GH21-1083

Cooperative Development Environment

- *CoOperative Development Environment/370: Debug Tool*, SC09-1623

DATABASE 2 for Common Servers

- *DATABASE 2 Administration Guide for common servers*, S20H-4580
- *DATABASE 2 Application Programming Guide for common servers*, S20H-4643
- *DATABASE 2 Software Developer's Kit for AIX: Building Your Applications*, S20H-4780
- *DATABASE 2 Software Developer's Kit for OS/2: Building Your Applications*, S20H-4787
- *DATABASE 2 SQL Reference for common servers*, S20H-4665
- *DATABASE 2 Call Level Interface Guide and Reference for common servers*, S20H-4644

Data Extract (DXT)

- *Data Extract Version 2: General Information*, GC26-4666
- *Data Extract Version 2: Planning and Administration Guide*, SC26-4631

DataPropagator NonRelational

- *DataPropagator NonRelational MVS/ESA Administration Guide*, SH19-5036
- *DataPropagator NonRelational MVS/ESA Reference*, SH19-5039

DataPropagator Relational

- *DataPropagator Relational User's Guide*, SC26-3399
- *IBM An Introduction to DataPropagator Relational*, GC26-3398

Data Facility Data Set Services

- *Data Facility Data Set Services: User's Guide and Reference*, SC26-4125

Database Design

- *DB2 Database Design and Implementation Using DB2*, SH24-6101
- *DB2 Design and Development Guide*, Gabrielle Wiorkowski and David Kull, Addison Wesley
- *Handbook of Relational Database Design*, C. Fleming and B Von Halle, Addison Wesley
- *Principles of Database Systems*, Jeffrey D. Ullman, Computer Science Press

DataHub

- *IBM DataHub General Information*, GC26-4874

DB2 Universal Database

- *DB2 Universal Database Administration Guide*, S10J-8157
- *DB2 Universal Database API Reference*, S10J-8167
- *DB2 Universal Database Application Development Guide*, SC09-2845
- *DB2 Universal Database Building Applications for UNIX Environments*, S10J-8161
- *DB2 Universal Database Building Applications for Windows and OS/2 Environments*, S10J-8160
- *DB2 Universal Database CLI Guide and Reference*, S10J-8159
- *DB2 Universal Database SQL Reference*, S10J-8165

Device Support Facilities

- *Device Support Facilities User's Guide and Reference*, GC35-0033

DFSMS/MVS

- *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: DFSMSshm Managing Your Own Data*, SH21-1077
- *DFSMS/MVS: Diagnosis Reference for DFSMSdfp*, LY27-9606
- *DFSMS/MVS: Macro Instructions for Data Sets*, SC26-4913
- *DFSMS/MVS: Managing Catalogs*, SC26-4914
- *DFSMS/MVS: Program Management*, SC26-4916
- *DFSMS/MVS: Storage Administration Reference for DFSMSdfp*, SC26-4920
- *DFSMS/MVS: Using Advanced Services for Data Sets*, SC26-4921
- *DFSMS/MVS: Utilities*, SC26-4926
- *MVS/DFP: Managing Non-VSAM Data Sets*, SC26-4557

DFSORT

- *DFSORT Application Programming: Guide*, SC33-4035

Distributed Relational Database

- *Data Stream and OPA Reference*, SC31-6806
- *Distributed Relational Database Architecture: Application Programming Guide*, SC26-4773
- *Distributed Relational Database Architecture: Connectivity Guide*, SC26-4783
- *Distributed Relational Database Architecture: Evaluation and Planning Guide*, SC26-4650
- *Distributed Relational Database Architecture: Problem Determination Guide*, SC26-4782
- *Distributed Relational Database: Every Manager's Guide*, GC26-3195
- *IBM SQL Reference*, SC26-8416
- *Open Group Technical Standard (the Open Group presently makes the following books available through their website at www.opengroup.org):*
 - *DRDA Volume 1: Distributed Relational Database Architecture (DRDA)*, ISBN 1-85912-295-7
 - *DRDA Volume 3: Distributed Database Management (DDM) Architecture*, ISBN 1-85912-206-X

Education

- *Dictionary of Computing*, SC20-1699
- *IBM Enterprise Systems Training Solutions Catalog*, GR28-5467

Enterprise System/9000 and Enterprise System/3090

- *Enterprise System/9000 and Enterprise System/3090 Processor Resource/System Manager Planning Guide*, GA22-7123

FORTRAN

- *VS FORTRAN Version 2: Language and Library Reference*, SC26-4221
- *VS FORTRAN Version 2: Programming Guide for CMS and MVS*, SC26-4222

High Level Assembler

- *High Level Assembler/MVS and VM and VSE Language Reference*, SC26-4940
- *High Level Assembler/MVS and VM and VSE Programmer's Guide*, SC26-4941

Parallel Sysplex Library

- *System/390 MVS Sysplex Application Migration*, GC28-1211
- *System/390 MVS Sysplex Hardware and Software Migration*, GC28-1210
- *System/390 MVS Sysplex Overview: An Introduction to Data Sharing and Parallelism*, GC28-1208
- *System/390 MVS Sysplex Systems Management*, GC28-1209
- *System/390 MVS 9672/9674 System Overview*, GA22-7148

ICSF/MVS

- *ICSF/MVS General Information*, GC23-0093

IMS/ESA

- *IMS Batch Terminal Simulator General Information*, GH20-5522
- *IMS/ESA Administration Guide: System*, SC26-8013
- *IMS/ESA Application Programming: Database Manager*, SC26-8727
- *IMS/ESA Application Programming: Design Guide*, SC26-8016
- *IMS/ESA Application Programming: Transaction Manager*, SC26-8729
- *IMS/ESA Customization Guide*, SC26-8020
- *IMS/ESA Installation Volume 1: Installation and Verification*, SC26-8023
- *IMS/ESA Installation Volume 2: System Definition and Tailoring*, SC26-8024
- *IMS/ESA Messages and Codes*, SC26-8028
- *IMS/ESA Operator's Reference*, SC26-8030
- *IMS/ESA Utilities Reference: System*, SC26-8035

ISPF

- *ISPF Version 4 Messages and Codes*, SC34-4450
- *ISPF Version 4 for MVS Dialog Management Guide*, SC34-4213
- *ISPF/PDF Version 4 for MVS Guide and Reference*, SC34-4258

- *ISPF and ISPF/PDF Version 4 for MVS Planning and Customization*, SC34-4134

Language Environment for MVS & VM

- *Language Environment for MVS & VM Concepts Guide*, GC26-4786
- *Language Environment for MVS & VM Debugging and Run-Time Messages Guide*, SC26-4829
- *Language Environment for MVS & VM Installation and Customization*, SC26-4817
- *Language Environment for MVS & VM Programming Guide*, SC26-4818
- *Language Environment for MVS & VM Programming Reference*, SC26-3312

MVS/ESA

- *MVS/ESA Analyzing Resource Measurement Facility Monitor I and Monitor II Reference and User's Guide*, LY28-1007
- *MVS/ESA Analyzing Resource Measurement Facility Monitor III Reference and User's Guide*, LY28-1008
- *MVS/ESA Application Development Reference: Assembler Callable Services for OpenEdition MVS*, SC23-3020
- *MVS/ESA Data Administration: Utilities*, SC26-4516
- *MVS/ESA Diagnosis: Procedures*, LY28-1844
- *MVS/ESA Diagnosis: Tools and Service Aids*, LY28-1845
- *MVS/ESA Initialization and Tuning Guide*, SC28-1451
- *MVS/ESA Initialization and Tuning Reference*, SC28-1452
- *MVS/ESA Installation Exits*, SC28-1459
- *MVS/ESA JCL Reference*, GC28-1479
- *MVS/ESA JCL User's Guide*, GC28-1473
- *MVS/ESA JES2 Initialization and Tuning Guide*, SC28-1453
- *MVS/ESA MVS Configuration Program*, GC28-1615
- *MVS/ESA Planning: Global Resource Serialization*, GC28-1450
- *MVS/ESA Planning: Operations*, GC28-1441
- *MVS/ESA Planning: Workload Management*, GC28-1493
- *MVS/ESA Programming: Assembler Services Guide*, GC28-1466
- *MVS/ESA Programming: Assembler Services Reference*, GC28-1474
- *MVS/ESA Programming: Authorized Assembler Services Guide*, GC28-1467
- *MVS/ESA Programming: Authorized Assembler Services Reference, Volumes 1-4*, GC28-1475, GC28-1476, GC28-1477, GC28-1478
- *MVS/ESA Programming: Extended Addressability Guide*, GC28-1468
- *MVS/ESA Programming: Sysplex Services Guide*, GC28-1495
- *MVS/ESA Programming: Sysplex Services Reference*, GC28-1496

- *MVS/ESA Programming: Workload Management Services*, GC28-1494
- *MVS/ESA Routing and Descriptor Codes*, GC28-1487
- *MVS/ESA Setting Up a Sysplex*, GC28-1449
- *MVS/ESA SPL: Application Development Guide*, GC28-1852
- *MVS/ESA System Codes*, GC28-1486
- *MVS/ESA System Commands*, GC28-1442
- *MVS/ESA System Management Facilities (SMF)*, GC28-1457
- *MVS/ESA System Messages Volume 1*, GC28-1480
- *MVS/ESA System Messages Volume 2*, GC28-1481
- *MVS/ESA System Messages Volume 3*, GC28-1482
- *MVS/ESA Using the Subsystem Interface*, SC28-1502

Net.Data for OS/390

- # • *Net.Data Language Environment Guide*, <http://www.ibm.com/software/net.data/docs>
- # • *Net.Data Programming Guide*, <http://www.ibm.com/software/net.data/docs>
- # • *Net.Data Reference Guide*, <http://www.ibm.com/software/net.data/docs>

NetView

- *NetView Installation and Administration Guide*, SC31-8043
- *NetView User's Guide*, SC31-8056

ODBC

- *ODBC 2.0 Programmer's Reference and SDK Guide*, ISBN 1-55615-658-8
- *Inside ODBC*, ISBN 1-55615-815-7

OS/390

- *OS/390 C/C++ Programming Guide*, SC09-2362
- *OS/390 C/C++ Run-Time Library Reference*, SC28-1663
- *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 *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) Security Administrator's Guide*, SC28-1915
- 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 *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 OpenEdition

- OS/390 *OpenEdition DCE Administration Guide*, SC28-1584
- OS/390 *OpenEdition DCE Introduction*, GC28-1581
- OS/390 *R1 OE DCE Messages and Codes*, ST01-0920
- OS/390 *OpenEdition Command Reference*, SC28-1892
- OS/390 *OpenEdition Messages and Codes*, SC28-1908
- OS/390 *OpenEdition Planning*, SC28-1890
- OS/390 *OpenEdition User's Guide*, SC28-1891

PL/I for MVS & VM

- *IBM PL/I MVS & VM Language Reference*, SC26-3114
- *IBM PL/I MVS & VM Programming Guide*, SC26-3113

OS PL/I

- OS *PL/I Programming Language Reference*, SC26-4308
- OS *PL/I Programming Guide*, SC26-4307

PROLOG

- *IBM SAA AD/Cycle Prolog/MVS & VM Programmer's Guide*, SH19-6892

Query Management Facility

- *Query Management Facility: Managing QMF for MVS*, SC26-8218
- *Query Management Facility: Reference*, SC26-4716
- *Query Management Facility: Using QMF*, SC26-8078

Remote Recovery Data Facility

- *Remote Recovery Data Facility Program Description and Operations*, LY37-3710

Resource Access Control Facility (RACF)

- *External Security Interface (RACROUTE) Macro Reference for MVS and VM*, GC28-1366
- *Resource Access Control Facility (RACF) Auditor's Guide*, SC28-1342
- *Resource Access Control Facility (RACF) Command Language Reference*, SC28-0733

- *Resource Access Control Facility (RACF) General Information Manual, GC28-0722*
- *Resource Access Control Facility (RACF) General User's Guide, SC28-1341*
- *Resource Access Control Facility (RACF) Security Administrator's Guide, SC28-1340*
- *Resource Access Control Facility (RACF) System Programmer's Guide, SC28-1343*

Storage Management

- *MVS/ESA Storage Management Library: Implementing System-Managed Storage, SC26-3123*
- *MVS/ESA Storage Management Library: Leading an Effective 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

- *IBM System/370 ESA Principles of Operation, SA22-7200*
- *IBM System/390 ESA Principles of Operation, SA22-7205*
- *System/390 MVS Sysplex Hardware and Software Migration, GC28-1210*

System Modification Program Extended (SMP/E)

- *System Modification Program Extended (SMP/E) Reference, SC28-1107*
- *System Modification Program Extended (SMP/E) User's Guide, SC28-1302*

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*

TSO Extensions

- *TSO/E CLISTS, SC28-1876*
- *TSO/E Command Reference, SC28-1881*
- *TSO/E Customization, SC28-1872*
- *TSO/E Messages, GC28-1885*
- *TSO/E Programming Guide, SC28-1874*
- *TSO/E Programming Services, SC28-1875*
- *TSO/E User's Guide, SC28-1880*

VS COBOL II

- *VS COBOL II Application Programming Guide for MVS and CMS, SC26-4045*
- *VS COBOL II Application Programming: Language Reference, SC26-4047*
- *VS COBOL II Installation and Customization for MVS, SC26-4048*

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*
- *VTAM for MVS/ESA Programming, SC31-6550*
- *VTAM for MVS/ESA Programming for LU 6.2, SC31-6551*
- *VTAM for MVS/ESA Resource Definition Reference, SC31-6552*

Index

Special Characters

- _ (underscore)
 - LIKE predicate 114
- , (comma) as decimal point 121
- : (colon) 89, 90
 - See *also* host variable
- ? (question mark) 382
- / (divide sign) 93
- . (period) as decimal point 121
- * (asterisk)
 - COUNT function 130
 - multiply sign 93
 - use in subselect 171
- % (percent sign)
 - LIKE predicate 114
- (minus sign) 94
- + (plus sign) 94, 521
- || (vertical bars) 92
 - See *also* CONCAT

A

- access path
 - description 21
- ACQUIRE
 - column of SYSPLAN catalog table 584
- ADD
 - clause of ALTER TABLE statement 219
- ADD VOLUMES clause of ALTER STOGROUP statement 215
- alias
 - creating 270
 - description 51
 - dropping 376
 - qualifying a column name 84
 - referencing another DB2 32
- ALIAS clause
 - COMMENT ON statement 255
 - CREATE ALIAS statement 270
 - DROP statement 376
 - LABEL ON statement 424
- ALL
 - clause of RELEASE statement 438
 - clause of subselect 171
 - keyword
 - AVG function 131
 - column functions 130
 - MAX function 133
 - MIN function 134
 - SUM function 135
 - quantified predicate 107
- ALL clause
 - ALTER INDEX statement 210
- ALL PRIVILEGES clause
 - GRANT statement 412
 - REVOKE statement 456
- ALL SQL clause of RELEASE statement 438
- ALLOCATE CURSOR statement
 - description 200
- alphabetic extender 45, 48
- ALTER DATABASE statement
 - description 202
- ALTER INDEX statement
 - description 205
- ALTER privilege
 - GRANT statement 412
 - REVOKE statement 456
- ALTER STOGROUP statement 214
- ALTER TABLE statement
 - description 217
- ALTER TABLESPACE statement
 - description 233
- ALTERAUTH column of SYSTABAUTH catalog table 600
- ALTEREDTS column
 - SYSDATABASE catalog table 557
 - SYSINDEXES catalog table 568
 - SYSSTOGROUP catalog table 597
 - SYSTABLESPACE catalog table 612
- ALTEREDTS column of SYSTABLES catalog table 608
- AND
 - truth table 118
- ANY
 - quantified predicate 107
 - USING clause of DESCRIBE statement 365
 - USING clause of PREPARE statement 434
- APOST option
 - precompiler 122
- apostrophe
 - string delimiter precompiler option 122
- APOSTSQL option
 - precompiler 122
- application plan
 - description 30
 - invalidated
 - ALTER TABLE statement 230
- application process 28
- application program
 - recovery 28
 - SQLCA 513
 - See *also* SQLCA (SQL communications area)
 - SQLDA 519
 - See *also* SQLDA (SQL descriptor area)

- ARCHIVE privilege
 - GRANT statement 409
 - REVOKE statement 453
- ARCHIVEAUTH column of SYSUSERAUTH catalog table 616
- arithmetic operators 93
- AS clause
 - CREATE VIEW statement 343
 - naming result columns 171
 - use in subselect 171
- AS WORKFILE clause of CREATE DATABASE statement 273
- ASC clause of select-statement 189
- ASCII and DBCS characters 58
- Assembler application program
 - host variable
 - EXECUTE IMMEDIATE statement 386
 - referencing 89
 - INCLUDE SQLCA 516
 - INCLUDE SQLDA 524
 - varying-length string variables 60
- assignment
 - datetime values 71
 - numbers 66
 - strings 69
- assignment statement
 - example 488
 - SQL procedure 486
- ASSOCIATE LOCATORS statement
 - description 243
- asterisk (*)
 - COUNT function 132
 - multiply sign 93
 - use in subselect 171
- ASUTIME clause
 - CREATE PROCEDURE statement 299
- ASUTIME column
 - SYS PROCEDURES catalog table 590
- AUDIT
 - clause of ALTER TABLE statement 222
 - clause of CREATE TABLE statement 322
- auditing
 - ALTER TABLE statement 222
 - CREATE TABLE statement 322
- AUDITING column of SYSTABLES catalog table 608
- AUTHHOWGOT column
 - SYSDBAUTH catalog table 559
 - SYSPACKAUTH catalog table 578
 - SYSPLANAUTH catalog table 587
 - SYSRESAUTH catalog table 594
 - SYSTABAUTH catalog table 600
 - SYSUSERAUTH catalog table 614
- AUTHID
 - column of MODESELECT catalog table 543
 - column of SYS PROCEDURES catalog table 590
 - column of USERNAMES catalog table 620

- authorization ID
 - description 52
 - primary
 - description 52
 - resulting from errors 463
 - secondary
 - description 52
 - translating
 - concepts 56
- AVG function 131
- AVGSIZE column
 - SYS PACKAGE catalog table 574
 - SYSPLAN catalog table 584

B

- base table 22
- basic operations in SQL 65
- basic predicate 106
- BCREATOR column
 - SYSPLANDEP catalog table 588
 - SYSVIEWDEP catalog table 617
- BEGIN DECLARE SECTION statement 246
- BETWEEN predicate 109
- BIND PACKAGE subcommand of DSN
 - options
 - QUALIFIER 51
- BIND PLAN subcommand of DSN
 - options
 - QUALIFIER 51
- BIND privilege
 - GRANT statement 406, 408
 - REVOKE statement 450, 452
- bind process 53
 - See *also* binding
- BIND_OPTS
 - column of SYSPSMOPTS table 626
- BINDADD privilege
 - binding a package 55
 - GRANT statement 409
 - REVOKE statement 453
- BINDADDAUTH column of SYSUSERAUTH catalog table 614
- BINDAGENT privilege
 - GRANT statement 409
 - REVOKE statement 453
- BINDAGENTAUTH column of SYSUSERAUTH catalog table 616
- BINDAUTH column
 - SYSPACKAUTH catalog table 578
 - SYSPLANAUTH catalog table 587
- BINDDATE column of SYSPLAN catalog table 584
- BINDERROR column of SYSPACKSTMT catalog table 581
- binding
 - description 21

- binding (*continued*)
 - process 53
- BINDTIME column
 - SYSPACKAGE catalog table 574
 - SYSPLAN catalog table 584
- bit data
 - conversion restrictions 40
 - description 57
- BNAME column
 - SYSPACKDEP catalog table 579
 - SYSPLANDEP catalog table 588
 - SYSVIEWDEP catalog table 617
- BOUNDBY column of SYSPLAN catalog table 585
- BOUNDTS column
 - SYSPLAN catalog table 586
- BPOOL column
 - SYSDATABASE catalog table 557
 - SYSINDEXES catalog table 567
 - SYSTABLESPACE catalog table 610
- BQUALIFIER column of SYSPACKDEP catalog table 579
- BSDS (bootstrap data set)
 - granting privilege to recover 409
 - revoking privilege to recover 453
- BSDS privilege
 - granting 409
 - revoking 453
- BSDSAUTH column of SYSUSERAUTH catalog table 614
- BTYPE column
 - SYSPACKDEP catalog table 579
 - SYSPLANDEP catalog table 588
 - SYSVIEWDEP catalog table 617
- buffer pool
 - naming convention 49
- BUFFERPOOL clause
 - ALTER DATABASE statement 202
 - ALTER INDEX statement 206
 - ALTER TABLESPACE statement 234
 - CREATE DATABASE statement 273
 - CREATE INDEX statement 290
 - CREATE TABLESPACE statement 335
- BUFFERPOOL privilege
 - GRANT statement 415
 - REVOKE statement 458
- BUILDNAME
 - column of SYSPSMOPTS table 626
- BUILDOWNER
 - column of SYSPSMOPTS table 626
- BUILDSHEMA
 - column of SYSPSMOPTS table 626
- built-in function 129
 - See *also* function
- BY clause of REVOKE statement 444

C

- C application program
 - host variable
 - EXECUTE IMMEDIATE statement 386
 - referencing 89
 - INCLUDE SQLCA 515
 - INCLUDE SQLDA 524
 - varying-length string 60
- CACHESIZE
 - column of SYSPLAN catalog table 585
- call level interface 22
- CALL statement 248
- capturing changed data
 - ALTER TABLE statement 227
 - CREATE TABLE statement 322
- CARD column
 - SYSTABLEPART catalog table
 - description 603
 - SYSTABSTATS catalog table
 - description 613
- CARDF column
 - SYSCOLDIST catalog table 547
 - SYSCOLDISTSTATS catalog table 548
 - SYSINDEXPART catalog table 571
 - SYSTABLES catalog table 608
- CASCADE delete rule
 - ALTER TABLE statement 226
 - CREATE TABLE statement 318
 - description 24
- cascade revoke 444
- CASE expression
 - description 103
- CASE statement
 - example 490
 - SQL procedure 489
- catalog name
 - naming convention 49
 - VCAT clause
 - ALTER INDEX statement 208
 - CREATE INDEX statement 285
 - CREATE TABLESPACE statement 330, 332
- catalog tables 529
 - description 27, 529
 - indexes 529
 - IPNAMES 537
 - LOCATIONS 538
 - LULIST 539
 - LUMODES 540
 - LUNAMES 541
 - MODESELECT 543
 - SQL statements allowed 532
 - SYSCHECKDEP 544
 - SYSCHECKS 545
 - SYSCOLAUTH 546
 - SYSCOLDIST
 - contents 547

catalog tables (*continued*)

- SYSCOLDISTSTATS
 - contents 548
- SYSCOLSTATS
 - contents 549
- SYSCOLUMNS
 - contents 550
- SYSCOPY
 - contents 554
- SYSDATABASE
 - contents 557
- SYSDBAUTH 559
- SYSDBRM 562
- SYSDUMMY1 464, 564
- SYSFIELDS 565
- SYSFOREIGNKEYS 566
- SYSINDEXES
 - contents 567
- SYSINDEXPART
 - contents 570
- SYSINDEXSTATS 572
- SYSKEYS 573
- SYSPACKAGE 574
- SYSPACKAUTH 578
- SYSPACKDEP 579
- SYSPACKLIST 580
- SYSPACKSTMT 581
- SYSPKSYSTEM 583
- SYSPLAN 584
- SYSPLANAUTH
 - contents 587
- SYSPLANDEP
 - contents 588
- SYSPLSYSTEM 589
- SYSPROCEDURES
 - contents 590
- SYSRELS
 - contents 593
- SYSRESAUTH 594
- SYSSTMT 595
- SYSSTOGROUP
 - contents 597
- SYSSTRINGS
 - contents 598
- SYSSYNONYMS 599
- SYSTABAUTH
 - contents 600
- SYSTABLEPART
 - contents 603
- SYSTABLES
 - contents 606
- SYSTABLESPACE
 - contents 610
- SYSTABSTATS
 - contents 613
- SYSUSERAUTH 614

catalog tables (*continued*)

- SYSVIEWDEP
 - contents 617
- SYSVIEWS 618
- SYSVOLUMES 619
 - table space 529
- USERNAMES 620

catalog, DB2

- description 27
- tables 529
 - See also* catalog tables

CCSID

- clause of ALTER DATABASE statement 203
- clause of ALTER TABLESPACE statement 241
- clause of CREATE DATABASE statement 273
- clause of CREATE GLOBAL TEMPORARY TABLE statement 277
- clause of CREATE PROCEDURE statement 298
- clause of CREATE TABLE statement 323
- clause of CREATE TABLESPACE statement 337

CCSID (coded character set identifier) 37

- See also* character conversion
- definition 37
- description 38
- system 39

CD-ROM, books on 7

CHANGED clause

- ALTER INDEX statement 210

CHAR

- function 137

CHAR VARYING data type 312

character 45

character conversion

- assignment rules 70
- character set 37
- code page 37
- code point 37
- coded character set 37
- comparison rules 73
- concatenation rules 185
- contracting conversion 41
- description 37
- encoding scheme 38
- expanding conversion 40
- substitution byte 38
- SYSIBM.SYSSTRINGS catalog table 598
- UNION and UNION ALL rules 185

CHARACTER data type

- CREATE TABLE statement 312
- DECLARE TABLE statement 354
- description 57

character set 37

character string

- assignment 69
- comparison 72
- constants 75

character string (*continued*)
 description 57
 empty 57
 CHARACTER VARYING data type 312
 CHARSET column
 SYSDBRM catalog table 562
 SYSPACKAGE catalog table 575
 CHECK
 clause of ALTER TABLE statement 223
 clause of CREATE TABLE statement 319
 column of SYSVIEWS catalog table 618
 check constraint
 See table check constraint
 CHECKCONDITION column
 SYSCHECKS catalog table 545
 CHECKFLAG column
 SYSTABLEPART catalog table 604
 SYSTABLES catalog table 607
 CHECKNAME column
 SYSCHECKDEP catalog table 544
 SYSCHECKS catalog table 545
 CHECKRID5B column
 SYSTABLEPART catalog table 605
 SYSTABLES catalog table 609
 CHECKS column
 SYSTABLES catalog table 608
 CHILDREN column of SYSTABLES catalog table 607
 CLOSE
 clause of ALTER INDEX statement 206
 clause of ALTER TABLESPACE statement 236
 clause of CREATE INDEX statement
 description 291
 clause of CREATE TABLESPACE statement
 description 336
 statement
 description 253
 closed state of cursor 430
 CLOSERULE column
 SYSINDEXES catalog table 568
 SYSTABLESPACE catalog table 611
 CLUSTER clause of CREATE INDEX statement
 description 289
 CLUSTERED column of SYSINDEXES catalog table
 description 567
 CLUSTERING column of SYSINDEXES catalog table
 description 567
 CLUSTERRATIO column
 SYSINDEXES catalog table
 description 568
 SYSINDEXSTATS catalog table
 description 572
 CLUSTERTTYPE column of SYSTABLES catalog
 table 606
 CNAME column
 SYSPKSYSTEM catalog table 583
 SYSPLSYSTEM catalog table 589
 COALESCE function 139
 COBOL application program
 host structure 90
 host variable
 description 89
 EXECUTE IMMEDIATE statement 386
 INCLUDE SQLCA 517
 varying-length string 60
 code page 37
 code point 37
 coded character set 37
 coded character set identifier 37
 See also CCSID (coded character set identifier)
 COLCARDATA column of SYSCOLSTATS catalog
 table 549
 COLCARDF column of SYSCOLUMNS catalog
 table 553
 COLCOUNT column
 SYSINDEXES catalog table 567
 SYSRELS catalog table 593
 SYSTABLES catalog table 606
 COLGROUPOCOLNO column
 SYSCOLDIST catalog table 547
 SYSCOLDISTSTATS catalog table 548
 collection, package
 granting privileges 403
 revoking privileges 447
 SET CURRENT PACKAGESET statement 470
 COLLID clause
 CREATE PROCEDURE statement 298
 COLLID column
 SYSCOLAUTH catalog table 546
 SYSPACKAGE catalog table 574
 SYSPACKAUTH catalog table 578
 SYSPACKLIST catalog table 580
 SYSPACKSTMT catalog table 581
 SYSPKSYSTEM catalog table 583
 SYSPROCEDURES catalog table 590
 SYSTABAUTH catalog table 601
 COLNAME column
 SYSCHECKDEP catalog table 544
 SYSCOLAUTH catalog table 546
 SYSFOREIGNKEYS catalog table 566
 SYSKEYS catalog table 573
 COLNO column
 SYSCOLUMNS catalog table 550
 SYSFIELDS catalog table 565
 SYSFOREIGNKEYS catalog table 566
 SYSKEYS catalog table 573
 colon 89
 See also host variable
 host variable in SQL 90
 COLSEQ column
 SYSFOREIGNKEYS catalog table 566
 SYSKEYS catalog table 573

COLTYPE column of SYSCOLUMNS catalog table 550

column

- derived
 - CREATE VIEW statement 343
 - functions 129
 - INSERT statement 420
 - null value 173
 - string comparison 73
 - UPDATE statement 479
- description 22
- name
 - ambiguous reference 85
 - correlated reference 86
 - in a result 173
 - undefined reference 85
- restricting values 26
- rules 183

COLUMN

- clause of COMMENT ON statement 255
- clause of LABEL ON statement 425

column function 130

- See *also* function

COLVALUE column

- SYSCOLDIST catalog table
 - description 547
- SYSCOLDISTSTATS catalog table
 - description 548

COMMA

- column of SYSDBRM catalog table 562
- column of SYSPACKAGE catalog table 575
- option of precompiler 121

comment

- adding 255
- replacing 255
- SQL 125

COMMENT ON statement

- column name qualification 84
- description 255

commit

- description 28

COMMIT ON RETURN clause

- CREATE PROCEDURE statement 300

COMMIT statement

- description 257

COMMIT_ON_RETURN column

- SYS PROCEDURES catalog table 591

comparison

- compatibility rules 65
- datetime values 74
- numbers 72
- strings 72

compatibility

- data types 65
- rules 65

COMPILE_OPTS

- column of SYSPSMOPTS table 626

compound statement

- example 495
- order of statements in 494
- SQL procedure 491

COMPRESS

- clause of ALTER TABLESPACE statement 240
- clause of CREATE TABLESPACE statement 337
- column of SYSTABLEPART catalog table 604

CONCAT operator 92

concatenation

- operator 92

concurrency

- application 28
- LOCK TABLE statement 426

CONNECT

- option of precompiler 119
- statement
 - differences, type 1 and type 2 259
 - type 1 262
 - type 2 267

connected state 35

connection

- DB2 private 32
- SQL 33

connection exit routine

- description 82

connection state

- application process 33, 264
- CONNECT (Type 1) statement 264
- SET CONNECTION statement 465
- SQL 33

constant

- character string 75
- decimal 75
- floating-point 75
- graphic string 77
- hexadecimal 75
- integer 74

constants

- datetime 76

constraint

- See table check constraint

CONSTRAINT

- clause of ALTER TABLE statement 223
- clause of CREATE TABLE statement 319

CONTINUE

- clause of WHENEVER statement 483

CONTINUE handler

- SQL procedure 493

CONTOKEN column

- SYSCOLAUTH catalog table 546
- SYSPACKAGE catalog table 574
- SYSPACKSTMT catalog table 581
- SYSPKSYSTEM catalog table 583

CONTOKEN column (*continued*)
 SYSTABAUTH catalog table 601
 conversion of numbers
 errors 463
 precision 68
 scale 68
 conversion, character 37
 See *also* character conversion
 CONVERT TO clause
 ALTER INDEX statement 211
 CONVERT TO clause of ALTER INDEX
 statement 205
 CONVLIMIT column of LUMODES catalog table
 description 540
 COPY privilege
 GRANT statement 406
 REVOKE statement 450
 COPYAUTH column of SYSPACKAUTH catalog
 table 578
 correlated reference
 correlation name
 defining 84
 FROM clause of subselect 174
 naming convention 49
 qualifying a column name 84
 description 86
 HAVING clause 178
 WHERE clause 177
 COUNT function 132
 CREATE ALIAS statement 270
 CREATE DATABASE statement
 description 272
 CREATE GLOBAL TEMPORARY TABLE statement
 description 275
 CREATE IN privilege
 binding a package 55
 GRANT statement 403
 REVOKE statement 447
 CREATE INDEX statement
 description 280
 CREATE PROCEDURE (SQL procedure) statement
 description 295
 CREATE PROCEDURE statement
 assignment statement 487
 SQL procedure body 486
 CREATE STOGROUP statement 303
 CREATE SYNONYM statement 306
 CREATE TABLE statement
 description 308
 CREATE TABLESPACE statement
 description 327
 CREATE VIEW statement
 description 341
 use 27
 CREATEALIAS privilege
 GRANT statement 409
 REVOKE statement 453
 CREATEALIASAUTH column of SYSUSERAUTH
 catalog table 615
 CREATEDBA privilege
 GRANT statement 409
 REVOKE statement 453
 CREATEDBAAUTH column of SYSUSERAUTH catalog
 table 614
 CREATEDBC privilege
 GRANT statement 409
 REVOKE statement 454
 CREATEDBCAUTH column of SYSUSERAUTH catalog
 table 614
 CREATEDBY column
 SYSDATABASE catalog table 557
 SYSINDEXES catalog table 568
 SYSSTOGROUP catalog table 597
 SYSSYNONYMS catalog table 599
 SYSTABLES catalog table 608
 SYSTABLESPACE catalog table 611
 CREATEDTS column
 SYSDATABASE catalog table 557
 SYSINDEXES catalog table 568
 SYSSTOGROUP catalog table 597
 SYSSYNONYMS catalog table 599
 SYSTABLESPACE catalog table 612
 CREATEDTS column of SYSTABLES catalog
 table 608
 CREATESG privilege
 GRANT statement 410
 REVOKE statement 454
 CREATESGAUTH column of SYSUSERAUTH catalog
 table 614
 CREATETAB privilege
 GRANT statement 404
 REVOKE statement 448
 CREATETABAUTH column of SYSDBAUTH catalog
 table 559
 CREATETMTAB privilege
 GRANT statement 410
 REVOKE statement 454
 CREATETMTABAUTH column
 SYSUSERAUTH catalog table 616
 CREATETS privilege
 GRANT statement 404
 REVOKE statement 448
 CREATETSAUTH column of SYSDBAUTH catalog
 table 559
 CREATOR column
 SYSCHECKS catalog table 545
 SYSCOLAUTH catalog table 546
 SYSDATABASE catalog table 557
 SYSFOREIGNKEYS catalog table 566
 SYSINDEXES catalog table 567
 SYSPACKAGE catalog table 574

- CREATOR column (*continued*)
 - SYSPLAN catalog table 584
 - SYSRELS catalog table 593
 - SYSSTOGROUP catalog table 597
 - SYSSYNONYMS catalog table 599
 - SYSTABLES catalog table 606
 - SYSTABLESPACE catalog table 610
 - SYSVIEWS catalog table 618
- CURRENT
 - clause of RELEASE statement 437
- current connection state 34
- CURRENT DATE special register 79
- CURRENT DEGREE special register
 - assigning value to 468
 - description 79
 - setting 468
- CURRENT PACKAGESET special register
 - assigning value to 470
 - description 80
 - stored procedures 471
- CURRENT PRECISION special register
 - assigning value to 472
 - description 80
- CURRENT RULES special register
 - assigning value to 473
 - description 81
- current server
 - description 260
 - designating
 - CONNECT (Type 1) statement 262
 - CONNECT (Type 2) statement 267
- CURRENT SERVER special register
 - description 82
- CURRENT SQLID special register
 - assigning value to 474
 - description 82
 - initial value 54, 82
- CURRENT TIME special register
 - description 82
- CURRENT TIMESTAMP special register
 - description 83
- CURRENT TIMEZONE special register 83
- CURRENTSERVER
 - column of SYSPLAN catalog table 585
 - option of BIND PLAN subcommand 261
 - option of REBIND PLAN subcommand 261
- cursor
 - See also* ALLOCATE CURSOR statement
 - See also* DECLARE CURSOR statement
 - associating 200
 - closed state 430
 - closing
 - CLOSE statement 253
 - CONNECT (Type 1) statement 262
 - CONNECT (Type 2) statement 267
 - error in FETCH 398
 - error in UPDATE 480

- cursor (*continued*)
 - declaring 347
 - open state 398
 - opening
 - errors 430
 - OPEN statement 428
 - using
 - current row 398
 - FETCH statement 397
 - positions 398

D

- DATA CAPTURE clause
 - ALTER TABLE statement 227
 - CREATE TABLE statement 322
- data compression
 - COMPRESS clause of ALTER TABLESPACE statement 240
 - COMPRESS clause of CREATE TABLESPACE statement 337
- data type
 - ALTER TABLE statement 220
 - character string 57
 - CREATE TABLE statement 311
 - datetime
 - DATE 62
 - TIME 62
 - DECLARE TABLE statement 354
 - graphic string 60
 - list of 57
 - numeric 61
 - result column 173
- database
 - altering
 - ALTER DATABASE statement 202
 - creating 272
 - default database 50
 - See also* default database (DSNDB04)
 - description 27
 - DSNDB04 50
 - See also* default database (DSNDB04)
 - limits 505
 - naming convention 49
- DATABASE
 - clause of ALTER DATABASE statement 202
 - clause of DROP statement 376
 - clause of GRANT statement 405
 - clause of REVOKE statement 449
- DATA CAPTURE column of SYSTABLES catalog table 608
- date
 - arithmetic 99
 - data type 62
 - duration 97
 - strings 63

DATE
 data type
 CREATE TABLE statement 313
 DECLARE TABLE statement 354
 description 62
 function 140
 DATE FORMAT field of panel DSNTIP4 124
 date routine
 CHAR function 137
 DATEGRANTED column
 SYSCOLAUTH catalog table 546
 SYSDBAUTH catalog table 559
 SYSPLANAUTH catalog table 587
 SYSRESAUTH catalog table 594
 SYSTABAUTH catalog table 600
 SYSUSERAUTH catalog table 614
 datetime
 arithmetic 98
 constants 76
 data types
 description 62
 string representation 63
 description 63
 format
 EUR 63
 ISO 63
 JIS 63
 LOCAL 63
 setting through the CHAR function 137
 USA 63
 DAY function 141
 day of week calculation 142
 DAYS function 142
 DB2 books on line 7
 DB2 catalog tables
 See catalog tables
 DB2 private protocol access
 authorization ID 55
 description 31, 32
 mixed environment 509
 DB2 system tables
 SYSPSM 625
 SYSPSMOPTS 626
 DB2 version identification, current server 263, 268
 DBADM authority
 GRANT statement 404
 REVOKE statement 448
 DBADMAUTH column of SYSDBAUTH catalog
 table 559
 DBCS (double-byte character set)
 ASCII 47, 58
 EBCDIC 46, 58
 site 59
 SQL ordinary identifier 45, 46
 DBCS_CCSID column
 SYSDATABASE catalog table 557
 SYSTABLESPACE catalog table 612
 DBCTRL authority
 GRANT statement 404
 REVOKE statement 448
 DBCTRLAUTH column of SYSDBAUTH catalog
 table 559
 DBID
 column of SYSCHECKS catalog table 545
 column of SYSDATABASE catalog table 557
 column of SYSINDEXES catalog table 567
 column of SYSTABLES catalog table 606
 column of SYSTABLESPACE catalog table 610
 DBMAINT authority
 GRANT statement 404
 REVOKE statement 448
 DBMAINTAUTH column of SYSDBAUTH catalog
 table 559
 DBNAME column
 SYSCOPY catalog table 554
 SYSINDEXES catalog table 567
 SYSTABAUTH catalog table 600
 SYSTABLEPART catalog table 603
 SYSTABLES catalog table 606
 SYSTABLESPACE catalog table 610
 SYSTABSTATS catalog table 613
 DBRM (database request module)
 description 30
 DCLGEN subcommand of DSN
 description 62
 DCOLLID column of SYSPACKDEP catalog table 579
 DCONTOKEN column of SYSPACKDEP catalog
 table 579
 DCREATOR column of SYSVIEWDEP catalog
 table 617
 deadlock
 locks and uncommitted changes 28
 DEC15 precompiler option 94
 DEC31
 column of SYSDBRM catalog table 562
 column of SYSPACKAGE catalog table 575
 precompiler option 94
 decimal
 constants 75
 data type
 CREATE TABLE statement 312
 DECLARE TABLE statement 354
 description 61
 function
 description 143
 numbers 61
 DECIMAL POINT IS field of panel DSNTIPF 121
 decimal point precompiler option 121
 DECLARE CURSOR statement
 description 347

DECLARE STATEMENT statement 352
 DECLARE TABLE statement
 description 354
 DEFAULT
 column of SYSCOLUMNS catalog table 551
 default database (DSNDB04)
 implicit specification 50
 DEFAULTVALUE column of SYSCOLUMNS catalog
 table 553
 DEFER
 clause of CREATE INDEX statement 291
 DEFERPREP column
 SYSPACKAGE catalog table 575
 SYSPLAN catalog table 585
 DEFERPREPARE
 column of SYSPACKAGE catalog table 577
 deferred embedded SQL 21
 DEGREE
 column of SYSPACKAGE catalog table 576
 column of SYSPLAN catalog table 585
 DELETE
 statement
 description 357
 DELETE privilege
 GRANT statement 412
 REVOKE statement 456
 delete rule 24, 360
 delete-connected 24
 DELETEAUTH column of SYSTABAUTH catalog
 table 601
 DELETERULE column of SYSRELS catalog table 593
 deleting
 rows from a table 357
 SQL objects 375
 delimited identifier in SQL 47
 delimiter
 SQL 47
 dependent
 row 24
 table 24
 DESC clause
 CREATE INDEX statement 284
 select-statement 189
 descendent table 24
 DESCRIBE CURSOR statement
 description 368
 DESCRIBE INPUT statement
 prepared statement 370
 DESCRIBE PROCEDURE statement
 description 372
 DESCRIBE statement
 prepared statement 362
 table or view 362
 variables 363
 descriptor name 49
 DEVTYPE column of SYSCOPY catalog table 554
 DFSMSHsm (Data Facility Hierarchical Storage
 Manager)
 dropping an index or table space 379
 digit, description in DB2 45
 DIGITS function 144
 DISCONNECT
 column of SYSPLAN catalog table 586
 DISPLAY privilege
 GRANT statement 410
 REVOKE statement 454
 DISPLAYAUTH column of SYSUSERAUTH catalog
 table 615
 DISPLAYDB privilege
 GRANT statement 404
 REVOKE statement 448
 DISPLAYDBAUTH column of SYSDBAUTH catalog
 table 559
 DISTINCT
 clause of subselect 171
 keyword
 AVG function 131
 column functions 130
 COUNT function 132
 MAX function 133
 MIN function 134
 SUM function 135
 distributed data
 CONNECT statement 259
 CURRENT SERVER special register 82
 description 31
 RELEASE statement 437
 SET CONNECTION statement 465
 Distributed Relational Database Architecture
 (DRDA) 32
 distributed unit of work 31
 See also &duw.
 DLOCATION column of SYSPACKDEP catalog
 table 579
 DNAME column
 SYSPACKDEP catalog table 579
 SYSPLANDEP catalog table 588
 SYSVIEWDEP catalog table 617
 dormant connection state 34
 DOUBLE data type
 CREATE TABLE statement 311
 DOUBLE PRECISION data type
 CREATE TABLE statement 311
 DECLARE TABLE statement 354
 double precision floating-point number 61
 double-byte character
 See also DBCS (double-byte character set)
 LABEL ON statement 425
 LIKE predicate 115
 strings 60
 truncated during assignment 70

- double-byte character set (DBCS) 45
 - See *also* DBCS (double-byte character set)
- DRDA (distributed relational database architecture) 32
- DRDA access
 - authorization ID 55
 - CONNECT (Type 1) statement 262
 - CONNECT (Type 2) statement 267
 - description 31, 32
 - mixed environment 509
 - restricted function 33
- DROP
 - statement
 - description 375
- DROP FOREIGN KEY clause of ALTER TABLE
 - statement 227
- DROP PRIMARY KEY clause of ALTER TABLE
 - statement 227
- DROP privilege
 - GRANT statement 404
 - REVOKE statement 448
- DROPAUTH column of SYSDBAUTH catalog
 - table 560
- DSETPASS
 - clause of ALTER INDEX statement 207
 - clause of ALTER TABLESPACE statement 236
 - clause of CREATE INDEX statement 291
 - clause of CREATE TABLESPACE statement 337
 - column of SYSINDEXES catalog table 568
 - column of SYSTABLESPACE catalog table 610
- DSNAME
 - column of SYSCOPY catalog table 554
- DSNUM
 - column of SYSCOPY catalog table 554
- DSVOLSER column of SYSCOPY catalog table 555
- duplicate rows, UNION clause 183
- duration
 - date 97
 - labeled 97
 - time 97
 - timestamp 97
- dynamic SQL
 - description 21, 195
 - EXECUTE IMMEDIATE statement 386
 - EXECUTE statement 382
 - execution 197
 - INTO clause
 - DESCRIBE statement 362
 - PREPARE statement 433
 - invocation of SELECT statement 198
 - preparation 197
 - SQLDA 519
 - statements allowed 509
- DYNAMICRULES
 - column of SYSPACKAGE catalog table 576
 - column of SYSPLAN catalog table 586

- DYNRULS installation parameter 121

E

- EBCDIC and DBCS characters 58
- EBCDIC CODED CHAR SET field of panel
 - DSNTIPF 123
- edit routine
 - named in CREATE TABLE statement 321
 - specified by EDITPROC option 321
- EDITPROC clause
 - CREATE TABLE statement 321
- EDPROC column of SYSTABLES catalog table 606
- ENABLE
 - column of SYSPKSYSTEM catalog table 583
 - column of SYSPLSYSTEM catalog table 589
- encoding scheme 38
- ENCODING_SCHEME column
 - SYSDATABASE catalog table 557
 - SYSTABLES catalog table 609
 - SYSTABLESPACE catalog table 612
- ENCRYPTPSWDS column of LUNAMES catalog
 - table 541
- END DECLARE SECTION statement 380
- ERASE clause
 - ALTER INDEX statement 210
 - ALTER TABLESPACE statement 239
 - CREATE INDEX statement 286
 - CREATE TABLESPACE statement 332
- ERASERULE column
 - SYSINDEXES catalog table 568
 - SYSTABLESPACE catalog table 610
- error
 - arithmetic expression 463
 - closes cursor 430
 - during FETCH 398
 - during update 480
 - numeric conversion 463
- ERRORBYTE column of SYSSTRINGS catalog
 - table 598
- escape character
 - LIKE predicate 113
- ESCAPE clause of LIKE predicate 113
- EUR (IBM European standard) 63
 - See *also* *datetime*
- evaluation order 102
- EXCLUSIVE
 - option of LOCK TABLE statement 427
- exclusive dependence 444
- executable statement 195, 196
- EXECUTE IMMEDIATE statement
 - description 386
- EXECUTE privilege
 - GRANT statement 406, 408
 - REVOKE statement 450, 452

EXECUTE statement
 description 382

EXECUTEAUTH column
 SYSPACKAUTH catalog table 578
 SYSPLANAUTH catalog table 587

EXISTS predicate 109

EXIT handler
 SQL procedure 494

exit procedure 222
 See *also* exit routine

exit routine 137, 321
 See *also* date routine
 See *also* edit routine
 See *also* field procedure
 See *also* validation routine
 named in ALTER TABLE statement 222
 named in CREATE TABLE statement 315, 321

EXITPARAM column of SYSFIELDS catalog table 565

EXITPARML column of SYSFIELDS catalog table 565

EXPLAIN
 column of SYSPACKAGE catalog table 575
 statement
 description 388

explainable statement
 description 388
 EXPLAIN statement 389
 using bind or rebind 390

EXPLAN column of SYSPLAN catalog table 585

exposed name 87

EXPREDICATE column of SYSPLAN catalog table 585

expression
 arithmetic operators 93
 CASE 103
 concatenation operator 92
 datetime operands 97
 decimal operands 94
 floating-point operands 97
 integer operands 94
 precedence of operation 102
 subselect statement 171
 without operators 92

EXTERNAL_SECURITY column
 SYSPROCEDURES catalog table 591

F

FARINDREF column of SYSTABLEPART catalog table
 description 603

FAROFFPOSF column
 SYSINDEXPART catalog table 571

FETCH statement
 description 397

field description 221

field procedure
 comparisons 73

field procedure (*continued*)
 named in ALTER TABLE statement 221
 named in CREATE TABLE statement 315

FIELDPROC clause
 ALTER TABLE statement 221
 CREATE TABLE statement 315

FILESEQNO column of SYSCOPY catalog table 554

FIRSTKEYCARD column
 SYSINDEXSTATS catalog table
 description 572

FIRSTKEYCARDF column
 SYSINDEXES catalog table 568

FLDPROC column
 SYSCOLUMNS catalog table 552
 SYSFIELDS catalog table 565

FLDTYPE column of SYSFIELDS catalog table 565

FLOAT
 data type
 CREATE TABLE statement 311
 DECLARE TABLE statement 354
 format 311
 function 145

floating-point
 constants 75
 double precision number 61
 single precision number 61

FOR
 clause of ALTER TABLE statement 220
 clause of CREATE ALIAS statement 271
 clause of CREATE SYNONYM statement 306
 clause of CREATE TABLE statement 313
 clause of EXPLAIN statement 389

FOR FETCH ONLY clause 190

FOR READ ONLY clause 190

FOR RESULT SET clause of ALLOCATE CURSOR statement 200

FOR UPDATE OF clause
 NOFOR precompiler option 126
 select-statement 189

foreign key
 description 24
 See *also* key, foreign

FOREIGN KEY clause
 ALTER TABLE statement
 description 225
 CREATE TABLE statement
 description 317

FOREIGNKEY column of SYSCOLUMNS catalog table 552

FORTRAN application program
 host variable 89
 INCLUDE SQLCA 517
 varying-length string 60

free space
 index 288
 table space 237

FREEPAGE
 clause of ALTER INDEX statement
 description 207
 clause of ALTER TABLESPACE statement
 description 237
 clause of CREATE INDEX statement
 description 288
 clause of CREATE TABLESPACE statement
 description 333
 column of SYSINDEXPART catalog table 570
 column of SYSTABLEPART catalog table 604
 FREQUENCYF column
 SYSCOLDIST catalog table 547
 SYSCOLDISTSTATS catalog table 548
 FROM clause
 DELETE statement 358
 PREPARE statement 434
 REVOKE statement 444
 subselect 174
 FULL OUTER JOIN 175
 See *also* join operation
 example 180
 FROM clause of subselect 175
 FULLKEYCARD column
 SYSINDEXSTATS catalog table 572
 FULLKEYCARDF column
 SYSINDEXES catalog table 568
 fullselect 183, 187
 function
 column
 AVG 131
 column name 84
 COUNT 132
 description 129
 example 130
 MAX 133
 MIN 134
 SUM 135
 maximum number in select 506
 nesting 136
 scalar
 CHAR 137
 COALESCE 139
 DATE 140
 DAY 141
 DAYS 142
 DECIMAL 143
 description 136
 DIGITS 144
 example 136
 FLOAT 145
 HEX 146
 HOUR 147
 INTEGER 148
 LENGTH 149
 MICROSECOND 150
 MINUTE 151

function (*continued*)
 scalar (*continued*)
 MONTH 152
 NULLIF 153
 SECOND 154
 STRIP 155
 SUBSTR 157
 TIME 159
 TIMESTAMP 160
 VALUE 162
 VARGRAPHIC 164
 YEAR 166

G

GBPCACHE clause
 ALTER INDEX statement 210
 ALTER TABLESPACE statement 240
 CREATE INDEX statement 288
 CREATE TABLESPACE statement 333
 GBPCACHE column
 SYSINDEXPART catalog table 571
 SYSTABLEPART catalog table 604
 GENERIC column of LUNAMES catalog table 542
 GET DIAGNOSTICS statement
 example 497
 SQL procedure 497
 GMT (Greenwich Mean Time) 79
 GO TO clause of WHENEVER statement 483
 GOTO statement
 example 498
 SQL procedure 498
 GRANT statement
 collection privileges 403
 database privileges 404
 description 400
 package privileges 406
 plan privileges 408
 system privileges 409
 table privileges 412
 use privileges 415
 view privileges 412
 GRANTEDTS column
 SYSCOLAUTH catalog table 546
 SYSDBAUTH catalog table 560
 SYSPLANAUTH catalog table 587
 SYSRESAUTH catalog table 594
 SYSTABAUTH catalog table 601
 SYSUSERAUTH catalog table 616
 GRANTTEE column
 SYSCOLAUTH catalog table 546
 SYSDBAUTH catalog table 559
 SYSPACKAUTH catalog table 578
 SYSPLANAUTH catalog table 587
 SYSRESAUTH catalog table 594
 SYSTABAUTH catalog table 600

GRANTEE column (*continued*)
 SYSUSERAUTH catalog table 614

GRANTEETYPE column
 SYSCOLAUTH catalog table 546
 SYSPACKAUTH catalog table 578
 SYSTABAUTH catalog table 600

GRANTOR column
 SYSCOLAUTH catalog table 546
 SYSDBAUTH catalog table 559
 SYSPACKAUTH catalog table 578
 SYSPLANAUTH catalog table 587
 SYSRESAUTH catalog table 594
 SYSTABAUTH catalog table 600
 SYSUSERAUTH catalog table 614

GRAPHIC
 data type
 CREATE TABLE statement 312
 DECLARE TABLE statement 354
 option of precompiler 59, 123

graphic string
 constants 77
 description 60

Greenwich Mean Time (GMT) 79

GROUP BY clause
 cannot join view 345
 subselect
 description 177
 results 172

GROUP_MEMBER column
 SYSCOPY catalog table 556
 SYSDATABASE catalog table 557
 SYSPACKAGE catalog table 576
 SYSPLAN catalog table 586

grouping column 177

H

handler
 SQL procedure 493

handling errors
 SQL procedure 493

HAVING clause of subselect
 description 178
 results 172

held connection state 34

HEX function 146

hexadecimal constants 75

HIGH2KEY column
 SYSCOLSTATS catalog table
 description 549
 SYSCOLUMNS catalog table
 description 550

HIGHKEY column of SYSCOLSTATS catalog
 table 549

host identifier 48

host structure
 description 90

host variable
 colon 90
 description 89
 EXECUTE IMMEDIATE statement 386
 EXPLAIN statement 389
 FETCH statement 397
 input 89
 naming convention 49
 output 89
 PREPARE statement 434
 SELECT
 assignment 462
 substitution for parameter markers 382

HOSTLANG column
 SYSDBRM catalog table 562
 SYSPACKAGE catalog table 575

HOUR function 147

I

I/O processing
 CURRENT DEGREE special register 79

IBM SQL 3

IBMREQD column
 IPNAMES catalog table 537
 LOCATIONS catalog table 538
 LULIST catalog table 539
 LUMODES catalog table 540
 MODESELECT catalog table 543
 SYSCHECKDEP catalog table 544
 SYSCHECKS catalog table 545
 SYSCOLAUTH catalog table 546
 SYSCOLDIST catalog table 547
 SYSCOLDISTSTATS catalog table 548
 SYSCOLSTATS catalog table 549
 SYSCOLUMNS catalog table 551
 SYSCOPY catalog table 554
 SYSDATABASE catalog table 557
 SYSDBAUTH catalog table 560
 SYSDBRM catalog table 562
 SYSDUMMY1 catalog table 564
 SYSFIELDS catalog table 565
 SYSFOREIGNKEYS catalog table 566
 SYSINDEXES catalog table 568
 SYSINDEXPART catalog table 570
 SYSINDEXSTATS catalog table 572
 SYSKEYS catalog table 573
 SYSPACKAGE catalog table 576
 SYSPACKAUTH catalog table 578
 SYSPACKDEP catalog table 579
 SYSPACKLIST catalog table 580
 SYSPACKSTMT catalog table 581
 SYSPKSYSTEM catalog table 583
 SYSPLAN catalog table 584

IBMREQD column (*continued*)
 SYSPLANAUTH catalog table 587
 SYSPLANDEP catalog table 588
 SYSPLSYSTEM catalog table 589
 SYSPROCEDURES catalog table 591
 SYSRELS catalog table 593
 SYSRESAUTH catalog table 594
 SYSSTMT catalog table 595
 SYSSTOGROUP catalog table 597
 SYSSTRINGS catalog table 598
 SYSSYNONYMS catalog table 599
 SYSTABAUTH catalog table 601
 SYSTABLEPART catalog table 603
 SYSTABLES catalog table 607
 SYSTABLESPACE catalog table 611
 SYSTABSTATS catalog table 613
 SYSUSERAUTH catalog table 615
 SYSVIEWDEP catalog table 617
 SYSVIEWS catalog table 618
 SYSVOLUMES catalog table 619
 USERNAMES catalog table 620
 ICBACKUP column of SYSCOPY catalog table 555
 ICDATE column of SYSCOPY catalog table 554
 ICTIME column of SYSCOPY catalog table 554
 ICTYPE column of SYSCOPY catalog table 554
 ICUNIT column of SYSCOPY catalog table 555
 identifier in SQL
 delimited 47
 long 47
 ordinary 46
 IF statement
 example 496
 SQL procedure 496
 IMAGCOPY privilege
 GRANT statement 404
 REVOKE statement 448
 IMAGCOPYAUTH column of SYSDBAUTH catalog table 560
 IMBREQD column
 LUNAMES catalog table 542
 IMPLICIT column of SYSTABLESPACE catalog table 610
 IN
 clause of CREATE PROCEDURE statement 297
 clause of CREATE TABLE statement 320
 clause of CREATE TABLESPACE statement 329
 predicate 111
 IN EXCLUSIVE MODE clause of LOCK TABLE statement 427
 IN SHARE MODE clause of LOCK TABLE statement 426
 INCCSID column of SYSSTRINGS catalog table 598
 INCLUDE statement
 assembler declarations 516
 description 417
 SQLCA
 C 516
 INCLUDE statement (*continued*)
 SQLCA (*continued*)
 COBOL 517
 FORTRAN 517
 SQLDA
 Assembler 524
 C 525, 526
 PL/I 518, 525
 index
 altering
 ALTER INDEX statement 205
 catalog table 529
 creating
 CREATE INDEX statement 280
 description 23
 dropping 376
 partitioned 289
 primary 23
 space
 description 26
 types
 changing 205
 default 282
 unique 23
 INDEX clause
 ALTER INDEX statement 205
 CREATE INDEX statement 283
 DROP statement 376
 INDEX privilege
 GRANT statement 412
 REVOKE statement 456
 INDEXAUTH column of SYSTABAUTH catalog table 601
 INDEXSPACE column of SYSINDEXES catalog table 567
 INDEXTYPE column
 SYSINDEXES catalog table 568
 indicator array 90
 indicator variable
 description 89
 string expression 386
 infix operators 94
 INNER JOIN 175
 See *also* join operation
 example 180
 FROM clause of subselect 175
 INOUT clause
 CREATE PROCEDURE statement 297
 input host variable 89
 INSERT privilege
 GRANT statement 412
 REVOKE statement 456
 insert rule 24, 421
 INSERT statement
 description 419

INSERTAUTH column of SYSTABAUTH catalog table 601
 inserting
 declaration in a program 417
 rows in a table 419
 INTEGER
 data type
 CREATE TABLE statement 311
 DECLARE TABLE statement 354
 large 61
 small 61
 function 148
 integer constants 74
 integrated catalog facility
 CREATE INDEX statement 287
 identifier 49
 interactive SQL 22, 198
 INTO clause
 DESCRIBE CURSOR statement 368
 DESCRIBE INPUT statement 370
 DESCRIBE PROCEDURE statement 373
 DESCRIBE statement 363
 FETCH statement 397
 INSERT statement 420
 PREPARE statement 433
 SELECT INTO statement 462
 IPADDR column of IPNAMES catalog table 537
 IS clause
 COMMENT ON statement 256
 LABEL ON statement 425
 ISO (International Standards Organization) 63
 See also datetime
 ISOBID column of SYSINDEXES catalog table 567
 ISOLATION
 column of SYSPACKAGE catalog table 575
 column of SYSPACKSTMT catalog table 581
 column of SYSPLAN catalog table 584
 column of SYSSTMT catalog table 595
 isolation level
 control by SQL statement
 DELETE statement 360
 INSERT statement 421
 SELECT INTO statement 463
 select-statement 191
 IXCREATOR column
 SYSINDEXPART catalog table 570
 SYSKEYS catalog table 573
 SYSTABLEPART catalog table 603
 IXNAME column
 SYSINDEXPART catalog table 570
 SYSKEYS catalog table 573
 SYSTABLEPART catalog table 603
 IXNAME column of SYSRELS catalog table 593
 IXOWNER column of SYSRELS catalog table 593

J

JIS (Japanese Industrial Standard) 63
 See also datetime
 join operation
 example 180, 182
 FROM clause of subselect 176
 FULL OUTER JOIN
 FROM clause of subselect 175
 INNER JOIN
 FROM clause of subselect 175
 joining tables 175
 LEFT OUTER JOIN
 FROM clause of subselect 175
 RIGHT OUTER JOIN
 FROM clause of subselect 175
 summary of results 176

K

Katakana character 46
 KATAKANA value for EBCDIC CODED CHAR SET 46
 KEEP DYNAMIC column
 SYSPACKAGE catalog table 577
 SYSPLAN catalog table 586
 key
 composite
 description 23
 description 23
 foreign
 description 24
 length
 maximum 506
 partitioned index 289, 479
 parent 24
 primary
 defining on a single column 313
 description 23
 unique 23
 KEYCOLUMNS column of SYSTABLES catalog table 607
 KEYCOUNT column of SYSINDEXSTATS catalog table 572
 KEYOBID column of SYSTABLES catalog table 607
 KEYSEQ column of SYSCOLUMNS catalog table 552
 keywords, reserved 621

L

LABEL
 column of SYSCOLUMNS catalog table 553
 column of SYSTABLES catalog table 607
 LABEL ON statement 424
 labeled duration 97
 LABELS
 USING clause of DESCRIBE statement 365
 USING clause of PREPARE statement 434

LANGUAGE
 clause of CREATE PROCEDURE statement 298
 column of SYSPROCEDURES catalog table 590
LEAFDIST column of SYSINDEXPART catalog table
 description 570
LEAVE statement
 example 500
 SQL procedure 500
LEFT OUTER JOIN 175
 See *also* join operation
 example 181
 FROM clause of subselect 175
LENGTH
 column of SYSCOLUMNS catalog table 550
 column of SYSFIELDS catalog table 565
 function 149
 length attribute of column 59
 letter, description in DB2 45
 library
 online 7
LIKE clause
 CREATE GLOBAL TEMPORARY TABLE
 statement 277
 CREATE TABLE statement 320
LIKE predicate 112
LIMITKEY column
 SYSINDEXPART catalog table 570
 SYSTABLEPART catalog table 603
 limits, DB2 505
LINK_OPTS
 column of SYSPSMOPTS table 626
LINKAGE column of SYSPROCEDURES catalog
 table 590
LINKNAME
 column of USERNAMES catalog table 620
LINKNAME column
 IPNAMES catalog table 537
 LOCATIONS catalog table 538
 LULIST catalog table 539
 literal 74
LOAD privilege
 GRANT statement 405
 REVOKE statement 449
LOADAUTH column of SYSDBAUTH catalog
 table 560
LOADMOD column
 SYSPROCEDURES catalog table 590
LOCAL 63
 See *also* datetime
 local DB2 31
LOCATION
 column of LOCATIONS catalog table 538
 column of SYSPACKLIST catalog table 580
 column of SYSTABLES catalog table 608
LOCATION column
 SYSPACKAGE catalog table 574
LOCATION column (*continued*)
 SYSPACKAUTH catalog table 578
 SYSPACKSTMT catalog table 581
 SYSPKSYSTEM catalog table 583
 location identifier 48
 lock
 ALTER TABLESPACE statement 235
 CREATE TABLESPACE statement 335
 description 28
 during update 480
 LOCK TABLE statement 426
 object
 table space (table) 426
LOCK TABLE statement
 description 426
LOCKMAX clause
 ALTER TABLESPACE statement
 description 235
 CREATE TABLESPACE statement
 description 336
LOCKMAX column
 SYSTABLESPACE catalog table 611
LOCKPART clause
 ALTER TABLESPACE statement 241
 CREATE TABLESPACE statement 338
LOCKPART column
 SYSTABLESPACE catalog table 612
LOCKRULE column of SYSTABLESPACE catalog
 table 610
LOCKSIZE clause
 ALTER TABLESPACE statement
 description 235
 CREATE TABLESPACE statement
 description 335
 logical operator 118
 logical unit of work 28
 See *also* unit of work
LONG VARCHAR data type
 CREATE TABLE statement 312
 DECLARE TABLE statement 354
LONG VARGRAPHIC data type
 CREATE TABLE statement 312
 DECLARE TABLE statement 354
LOOP statement
 example 501
 SQL procedure 501
LOW2KEY column
 SYSCOLSTATS catalog table 549
 SYSCOLUMNS catalog table
 description 551
 lowercase character folded to uppercase 46
LOWKEY column of SYSCOLSTATS catalog
 table 549
LUNAME
 column of LULIST catalog table 539
 column of LUMODES catalog table 540

LUNAME (*continued*)
column of LUNAMES catalog table 541
column of MODESELECT catalog table 543
column of SYSPROCEDURES table 590

M

MAX function 133
MAXROWS clause
ALTER TABLESPACE statement 241
CREATE TABLESPACE statement 338
MAXROWS column
SYSTABLESPACE catalog table 612
MEMBER CLUSTER clause
CREATE TABLESPACE statement 334
message
precompiler processing of DECLARE TABLE statement 356
MICROSECOND function 150
MIN function 134
MINUTE function 151
MIXED column
SYSDBRM catalog table 562
SYSPACKAGE catalog table 575
mixed data
convention 5
description 58
in string assignments 70
LIKE predicate 115
MIXED DATA
field of panel DSNTIPF 59, 123
MIXED_CCSID column
SYSDATABASE catalog table 558
SYSTABLESPACE catalog table 612
MODENAME column
LUMODES catalog table 540
MODESELECT catalog table 543
MODESELECT column of LUNAMES catalog table 542
MON1AUTH column of SYSUSERAUTH catalog table 615
MON2AUTH column of SYSUSERAUTH catalog table 615
MONITOR1 privilege
GRANT statement 410
REVOKE statement 454
MONITOR2 privilege
GRANT statement 410
REVOKE statement 454
MONTH function 152

N

NACTIVE column
SYSTABLESPACE catalog table
description 610

NACTIVE column (*continued*)
SYSTABSTATS catalog table 613

NAME

column of SYSCOLDIST catalog table 547
column of SYSCOLDISTSTATS catalog table 548
column of SYSCOLSTATS catalog table 549
column of SYSCOLUMNS catalog table 550
column of SYSDATABASE catalog table 557
column of SYSDBAUTH catalog table 559
column of SYSDBRM catalog table 562
column of SYSFIELDS catalog table 565
column of SYSINDEXES catalog table 567
column of SYSINDEXSTATS catalog table 572
column of SYSPACKAGE catalog table 574
column of SYSPACKAUTH catalog table 578
column of SYSPACKLIST catalog table 580
column of SYSPACKSTMT catalog table 581
column of SYSPKSYSTEM catalog table 583
column of SYSPLAN catalog table 584
column of SYSPLANAUTH catalog table 587
column of SYSPLSYSTEM catalog table 589
column of SYSRESAUTH catalog table 594
column of SYSSTMT catalog table 595
column of SYSSTOGROUP catalog table 597
column of SYSSYNONYMS catalog table 599
column of SYSTABLES catalog table 606
column of SYSTABLESPACE catalog table 610
column of SYSTABSTATS catalog table 613
column of SYSVIEWS catalog table 618

NAMES

USING clause of DESCRIBE statement 365
USING clause of PREPARE statement 434
names, prepared SQL statements 352
naming convention
SQL 48
NEARINDREF column of SYSTABLEPART catalog table 603
NEAROFFPOSF column
SYSINDEXPART catalog table 571
nested table expressions 174
NEWAUTHID column of USERNAMES catalog table 620
NLEAF column
SYSINDEXES catalog table
description 567
SYSINDEXSTATS catalog table 572
NLEVELS column
SYSINDEXES catalog table
description 567
SYSINDEXSTATS catalog table 572
NO ACTION
delete rule
CREATE TABLE statement 318
NO ACTION delete rule
description 24

NO WLM ENVIRONMENT clause
 CREATE PROCEDURE statement 299
 NOCOLLID clause
 CREATE PROCEDURE statement 298
 NOFOR option
 precompiler 126
 NOGRAPHIC option of precompiler 123
 nonexecutable statement 195, 196
 NOT FOUND clause of WHENEVER statement 483
 NOT NULL clause
 ALTER TABLE statement 220
 CREATE GLOBAL TEMPORARY TABLE statement
 description 277
 CREATE TABLE statement
 description 313
 NPAGES column
 SYSTABLES catalog table
 description 606
 SYSTABSTATS catalog table 613
 NTABLES column of SYSTABLESPACE catalog
 table 610
 NULL
 predicate 116
 VALIDPROC clause of ALTER TABLE
 statement 222
 null value
 assigned to host variable 462
 assignment 66
 description 57
 duplicate rows 171
 grouping columns 177
 result columns 173
 specified by indicator variable 89
 NULLIF function 153
 NULLS column of SYSCOLUMNS catalog table 550
 numbers in SQL 61
 NUMCOLUMNS column
 SYSCOLDIST catalog table 547
 SYSCOLDISTSTATS catalog table 548
 numeric
 assignments 66
 comparisons 72
 conversion errors 463
 data type 61
 NUMERIC data type
 CREATE TABLE statement 312
 DECLARE TABLE statement 354
 Numparts
 clause of CREATE TABLESPACE statement 334

O
 OBID
 clause of CREATE TABLE statement 322
 column of SYSCHECKS catalog table 545
 column of SYSINDEXES catalog table 567
 OBID (*continued*)
 column of SYSTABLES catalog table 606
 column of SYSTABLESPACE catalog table 610
 object table 84
 OBTYP column of SYSRESAUTH catalog table 594
 ON clause
 CREATE INDEX statement 284
 joining tables 175
 ON DELETE clause
 ALTER TABLE statement 226
 CREATE TABLE statement 318
 ON TABLE clause
 GRANT statement 413
 REVOKE statement 457
 online books 7
 OPEN
 statement
 description 428
 open cursor 398
 operands
 datetime 97
 decimal 94
 floating-point 97
 integer 94
 operation
 SQL
 assignment 65
 comparison 72
 description 65
 OPERATIVE column
 SYSPACKAGE catalog table 574
 SYSPLAN catalog table 584
 operator
 arithmetic 93
 OPTIMIZE FOR n ROWS
 clause of SELECT statement 190
 OR truth table 118
 ORDER BY clause
 select-statement 188
 order of evaluation, operators 102
 order of statements
 in a compound statement 494
 ORDERING column of SYSKEYS catalog table 573
 ordinary identifier in SQL 46
 OUT clause of CREATE PROCEDURE statement 297
 OUTCCSID column of SYSSTRINGS catalog
 table 598
 outer join 175
 See *also* join operation
 example 180, 181, 182
 FULL OUTER JOIN 180
 FROM clause of subselect 175
 LEFT OUTER JOIN 181
 FROM clause of subselect 175
 RIGHT OUTER JOIN 181
 FROM clause of subselect 175

output host variable 89

OWNER

column of SYSINDEXSTATS catalog table 572

column of SYSPACKAGE catalog table 574

column of SYSTABSTATS catalog table 613

P

PACKADM authority

GRANT statement 403

REVOKE statement 447

package

binding

remote 55

description 30

dropping 377

invalidated

ALTER TABLE statement 230

privileges

remote bind 55

PACKAGE

clause of DROP statement 377

clause of GRANT statement 406

clause of REVOKE statement 450

page set

description 26

PAGESAVE column of SYSTABLEPART catalog table

description 604

parallel processing

SET CURRENT DEGREE statement 468

parameter

assignment rules for CALL statement 251

passing to stored procedure 250

parameter marker

description 435

EXECUTE statement 382

EXPLAIN statement 389

obtaining information with DESCRIBE INPUT 370

OPEN statement 429

PREPARE statement 435

rules 435

parent key

description 24

parent row 24

parent table 24

PARENTS column of SYSTABLES catalog table 607

PARMLIST column

SYSFIELDS catalog table 565

SYS PROCEDURES catalog table 591

PART

clause of ALTER INDEX statement 207

clause of ALTER TABLESPACE statement 237

clause of LOCK TABLE statement 426

CLUSTER clause of CREATE INDEX

statement 289

NUMPARTS clause of CREATE TABLESPACE

statement 335

PARTITION column

SYSCOLDISTSTATS catalog table 548

SYSCOLSTATS catalog table 549

SYSINDEXPART catalog table 570

SYSINDEXSTATS catalog table 572

SYSTABLEPART catalog table 603

SYSTABLESPACE catalog table 610

SYSTABSTATS catalog table 613

PASSWORD

clause of ALTER STOGROUP statement 214

clause of CREATE STOGROUP statement 304

column of USERNAMES catalog table 620

pattern character

description 114

PCTFREE

clause of ALTER INDEX statement 207

clause of ALTER TABLESPACE statement 237

clause of CREATE INDEX statement 288

clause of CREATE TABLESPACE statement 333

column of SYSINDEXPART catalog table 570

column of SYSTABLEPART catalog table 604

PCTIMESTAMP column of SYSPACKAGE catalog

table 576

PCTPAGES column

SYSTABLES catalog table 606

SYSTABSTATS catalog table 613

PCTROWCOMP column

SYSTABLES catalog table

description 608

SYSTABSTATS catalog table 613

PDSNAME column

SYSDBRM catalog table 562

SYSPACKAGE catalog table 576

PERCACTIVE column of SYSTABLEPART catalog

table

description 603

PERCDROP column of SYSTABLEPART catalog table

description 603

PERIOD option

precompiler 121

PGM_TYPE column

SYS PROCEDURES catalog table 591

PGSIZE column

SYSINDEXES catalog table 567

SYSTABLESPACE catalog table 610

PIECESIZE clause

ALTER INDEX statement 211

CREATE INDEX statement 291

PIECESIZE column

SYSINDEXES catalog table 569

PIT_RBA column of SYSCOPY catalog table 555

PKSIZE column of SYSPACKAGE catalog table 574

PL/I application program

host structure 90

host variable

description 89

PL/I application program (*continued*)
 INCLUDE SQLCA 518
 INCLUDE SQLDA 525
 varying-length string 60
 PLAN
 clause of EXPLAIN statement 388
 clause of GRANT statement 408
 clause of REVOKE statement 452
 plan element 30, 470
 plan table 388
 PLAN_TABLE table
 EXPLAIN statement 389
 plan, application 230
 See also application plan
 PLANNAME column
 SYSIBM.MODESELECT catalog table 543
 SYSPACKLIST catalog table 580
 PLCREATOR column
 SYSDBRM catalog table 562
 SYSSTMT catalog table 595
 PLENTRIES column of SYSPLAN catalog table 585
 PLNAME column
 SYSDBRM catalog table 562
 SYSSTMT catalog table 595
 PLSIZE column of SYSPLAN catalog table 584
 point of consistency
 description 28
 PORT column of LOCATIONS catalog table 538
 PQTY column
 SYSINDEXPART catalog table 570
 SYSTABLEPART catalog table 603
 precedence of operators 102
 precision of numbers
 description 61
 determined by SQLLEN variable 523
 in assignments 66
 in comparisons 72
 results of arithmetic operations 93
 values for data types 61
 PRECOMPDATE column of SYSDBRM catalog
 table 562
 PRECOMPILE_OPTS
 column of SYSPSMOPTS table 626
 precompiler
 checks SQL statements 354
 DECLARE TABLE statement 354
 escape character 47
 options
 COBOL decimal point 121
 CONNECT 119
 date 124
 NOFOR 126
 STDSQL 124
 string delimiter 122
 time 124
 using INCLUDE statements 417
 PRECOMPTIME column of SYSDBRM catalog
 table 562
 PRECOMPTS column
 SYSDBRM catalog table 563
 predicate
 basic 106
 BETWEEN 109
 description 106
 EXISTS 109
 IN 111
 LIKE 112
 NULL 116
 quantified 107
 prefix operator 94
 PRELINK_OPTS
 column of SYSPSMOPTS table 626
 PREPARE
 statement
 description 433
 prepared SQL statement
 dynamically prepared by PREPARE 433
 executing 382
 identifying by DECLARE 352
 obtaining information with DESCRIBE 362
 obtaining information with DESCRIBE INPUT 370
 SQLDA provides information 519
 statements allowed 509
 primary index 23
 See also index, primary
 primary key 23
 See also key, primary
 PRIMARY KEY clause
 ALTER TABLE statement 223
 CREATE TABLE statement
 description 313, 316
 PRIQTY clause
 ALTER INDEX statement 209
 ALTER TABLESPACE statement 238
 CREATE INDEX statement 286
 CREATE TABLESPACE statement 330
 privilege
 revoking 456
 types 400
 See also GRANT statement
 PRIVILEGE column
 SYSCOLAUTH catalog table 546
 PROCEDURE column of SYSPROCEDURES catalog
 table 590
 PROCEDURENAME
 column of SYSPSM table 625
 column of SYSPSMOPTS table 626
 process
 description 28
 PROGRAM TYPE clause
 CREATE PROCEDURE statement 300

PSID column of SYSTABLESPACE catalog table 610
 PSMDATE
 column of SYSPSM table 625
 PSMTIME
 column of SYSPSM table 625
 PUBLIC AT ALL LOCATIONS clause
 GRANT statement 401
 REVOKE statement 444
 PUBLIC clause
 GRANT statement 401
 REVOKE statement 444

Q

qualification of column names 84
 QUALIFIER
 column of SYSPACKAGE catalog table 574
 column of SYSPLAN catalog table 585
 column of SYSRESAUTH catalog table 594
 unqualified option names 51
 quantified predicate 107
 query 169
 question mark (?) 382
 See also parameter marker
 quotation mark 47, 122
 QUOTE
 column of SYSDBRM catalog table 562
 column of SYSPACKAGE catalog table 575
 option of precompiler 122
 QUOTESQL option
 precompiler 122

R

RACF (Resource Access Control Facility)
 security for remote execution 56
 RBA column of SYSCHECKS catalog table 545
 RBA1 column of SYSTABLES catalog table 608
 RBA2 column of SYSTABLES catalog table 608
 read-only
 FOR FETCH ONLY clause 190
 FOR READ ONLY clause 190
 result table 349
 view 345
 REAL data type
 CREATE TABLE statement 311
 DECLARE TABLE statement 354
 RECLENGTH column of SYSTABLES catalog
 table 607
 RECOVER privilege
 GRANT statement 410
 REVOKE statement 454
 RECOVERAUTH column of SYSUSERAUTH catalog
 table 615
 RECOVERDB privilege
 GRANT statement 405

RECOVERDB privilege (*continued*)
 REVOKE statement 449
 RECOVERDBAUTH column of SYSDBAUTH catalog
 table 560
 recovery 257
 See also unit of recovery
 description
 restoring data consistency 28
 REFCOLS column
 SYSTABAUTH catalog table 601
 REFERENCES clause
 ALTER TABLE statement 225
 CREATE TABLE statement 317
 REFERENCES privilege
 GRANT statement 412
 REVOKE statement 456
 REFERENCESAUTH column of SYSTABAUTH catalog
 table 601
 referential constraint
 ALTER TABLE statement 224
 CREATE TABLE statement 317
 description 24
 referential cycle 24
 referential integrity
 description 24
 REFTBCREATOR column of SYSRELS catalog
 table 593
 REFTBNAME column of SYSRELS catalog table 593
 RELEASE
 column of SYSPACKAGE catalog table 575
 column of SYSPLAN catalog table 585
 statement 437
 release level identification, current server 263, 268
 release pending connection state 34
 RELNAME column
 SYSCOLUMNS catalog table 566
 SYSRELS catalog table 593
 RELOBID1 column of SYSRELS catalog table 593
 RELOBID2 column of SYSRELS catalog table 593
 REMARKS column
 SYSCOLUMNS catalog table 551
 SYSTABLES catalog table 607
 REMOTE
 column of SYSPACKAGE catalog table 576
 Remote Recovery Data Facility (RRDF) 227, 322
 See also RRDF (Remote Recovery Data Facility)
 remote unit of work 31
 See also &ruw.
 REMOVE VOLUMES clause of ALTER STOGROUP
 statement 215
 RENAME
 statement 440
 REOPTVAR
 column of SYSPACKAGE catalog table 577
 column of SYSPLAN catalog table 586

REORG privilege
 GRANT statement 405
 REVOKE statement 449

REORGAUTH column of SYSDBAUTH catalog table 560

REPAIR privilege
 GRANT statement 405
 REVOKE statement 449

REPAIRAUTH column of SYSDBAUTH catalog table 560

REPEAT statement
 example 502
 SQL procedure 502

reserved keywords 621

RESET
 clause of CONNECT statement 268

RESTRICT
 delete rule
 ALTER TABLE statement 226
 CREATE TABLE statement 318
 description 24

result column
 data type 173
 names 173

RESULT SET clause
 CREATE PROCEDURE statement 298

result table
 description 22

RESULT_SETS column
 SYSPROCEDURES catalog table 591

REVOKE statement
 cascading effect 444
 collection privileges 447
 database privileges 448
 description 443
 package privileges 450
 plan privileges 452
 system privileges 453
 table privileges 456
 use privileges 458
 view privileges 456

RIGHT OUTER JOIN 175
 See *also* join operation
 example 181
 FROM clause of subselect 175

rollback
 description 28

ROLLBACK statement
 description 460

ROSHARE
 clause of ALTER DATABASE statement 203
 clause of CREATE DATABASE statement 273
 column of SYSDATABASE catalog table 557

row
 deleting 357
 description 22

row (*continued*)
 inserting 419
 selecting single row 462
 updating 477

RRDF (Remote Recovery Data Facility)
 altering a table for 227
 creating a table for 322

RUN OPTIONS clause
 CREATE PROCEDURE statement 300

RUNOPTS column of SYSPROCEDURES catalog table 591

S

sample table
 description 22

SBCS data
 description 57

SBCS site 59

SBCS_CCSID column
 SYSDATABASE catalog table 557
 SYSTABLESPACE catalog table 612

scalar function 136
 See *also* function

SCALE column
 SYSCOLUMNS catalog table 550
 SYSFIELDS catalog table 565

scale of numbers
 description 61
 in assignments 68
 in comparisons 72
 results of arithmetic operations 95

SCHEMA column
 SYSPSM table 625
 SYSPSMOPTS table 626

SCREATOR column of SYSTABAUTH catalog table 600

search condition
 DELETE statement 358
 description 118
 HAVING 178
 order of evaluation 118
 UPDATE statement 479
 WHERE clause 177

SECOND function 154

SECQTY
 clause of ALTER INDEX statement 209
 clause of ALTER TABLESPACE statement 239
 clause of CREATE INDEX statement 286
 clause of CREATE TABLESPACE statement 331

SECTNO column
 SYSPACKSTMT catalog table 581
 SYSSTMT catalog table 595

SECURITY clause
 CREATE PROCEDURE statement 300

SECURITY_IN column of LUNAMES catalog table 541

SECURITY_OUT column
 IPNAMES catalog table 537
 LUNAMES catalog table 541

SEGSIZE
 clause of CREATE TABLESPACE statement 337
 column of SYSTABLESPACE catalog table 611

SELECT
 clause as syntax component 171

SELECT INTO statement 462

SELECT privilege
 GRANT statement 413
 REVOKE statement 456

SELECT statement
 description 188
 dynamic invocation 198
 example 191
 fullselect 183
 list
 application 172
 description 171
 maximum number of elements 506
 notation 171
 static invocation 197
 subselect 170

SELECTAUTH column of SYSTABAUTH catalog table 601

selecting
 single row 462

self-referencing constraint 24

self-referencing row 24

self-referencing table 24

SEQNO
 column of SYSPSM table 625

SEQNO column
 SYSPACKLIST catalog table 580
 SYSPACKSTMT catalog table 581
 SYSSTMT catalog table 595
 SYSVIEWS catalog table 618

server
 accessible 32
 current 32
 establishing with CONNECT 261
 remote 31

SET clause of UPDATE statement 479

SET CONNECTION statement 465

SET CURRENT DEGREE statement 468

SET CURRENT PACKAGESET statement 470

SET CURRENT PRECISION statement 472

SET CURRENT RULES statement 473

SET CURRENT SQLID statement 474

SET host-variable statement 476

SET NULL delete rule
 ALTER TABLE statement 226
 CREATE TABLE statement 318
 description 24

SET QUERYNO clause of EXPLAIN statement 388

SGCREATOR column of SYSVOLUMES catalog table 619

SGNAME column of SYSVOLUMES catalog table 619

SHARE
 option of LOCK TABLE statement 426

shift-in character
 convention 5
 LABEL ON statement 425
 not truncated by assignments 70

shift-out character
 convention 5
 LABEL ON statement 425

short identifier in SQL 47

short string columns 59

SHRLEVEL
 column of SYSCOPY catalog table 555

sign-on exit routine
 CURRENT SQLID special register 54, 82

single precision floating-point number 61

single-byte character in LIKE predicate 115

SMALLINT
 data type
 CREATE TABLE statement 311
 DECLARE TABLE statement 354

softcopy publications 7

SOME quantified predicate 107

SOURCEDSN
 column of SYSPSMOPTS table 626

space character 46

SPACE column
 SYSINDEXES catalog table 568
 SYSINDEXPART catalog table 570
 SYSSTOGROUP catalog table 597
 SYSTABLEPART catalog table 604
 SYSTABLESPACE catalog table 611

SPCDATE column of SYSSTOGROUP catalog table 597

special character 45

special register
 CURRENT DATE 79
 CURRENT DEGREE 79
 CURRENT PACKAGESET 80
 CURRENT PRECISION 80
 CURRENT RULES 81
 CURRENT SERVER 82
 CURRENT SQLID 82
 CURRENT TIME 82
 CURRENT TIMESTAMP 83
 CURRENT TIMEZONE 83
 CURRENT_DATE 79
 CURRENT_TIME 82
 CURRENT_TIMESTAMP 83
 description 78
 USER 83

SQL (Structured Query Language)

- assignment operation 65
 - call level interface 22
 - character 45
 - comparison operation 65
 - constants 74
 - data types
 - character strings 57
 - datetime 62
 - description 57
 - graphic strings 60
 - numbers 61
 - deferred embedded 21
 - delimited identifier 47
 - description 21
 - dynamic
 - description 21
 - statements allowed 509
 - identifier 46
 - interactive 22
 - keywords, reserved 621
 - limits 505
 - naming conventions 48
 - null value 57
 - ordinary identifier 45
 - rules 81
 - standard 3, 124
 - static
 - description 21
 - token 45
 - value 57
 - variable names 48
- SQL procedure
- statements allowed 511
- SQL procedure statement
- assignment statement 486
 - CASE statement 489
 - compound statement 491
 - CONTINUE handler 493
 - EXIT handler 494
 - GET DIAGNOSTICS statement 497
 - GOTO statement 498
 - handler 493
 - handling errors 493
 - IF statement 496
 - LEAVE statement 500
 - LOOP statement 501
 - order of statements 494
 - REPEAT statement 502
 - WHILE statement 503
- SQL return code
- See SQLCODE
- SQL statements
- ALLOCATE CURSOR
 - description 200
 - ALTER DATABASE 202

SQL statements (*continued*)

- ALTER INDEX 205
- ALTER STOGROUP 214
- ALTER TABLE 217
- ALTER TABLESPACE
 - description 233
- ASSOCIATE LOCATORS
 - description 243
- BEGIN DECLARE SECTION 246
- CALL
 - description 248
- catalog table restrictions 532
- CLOSE 253
- COMMENT ON 255
- COMMIT 257
- CONNECT (Type 1) 262
- CONNECT (Type 2) 267
- CONNECT differences 259
- CONTINUE 483
- CREATE ALIAS 270
- CREATE DATABASE 272
- CREATE GLOBAL TEMPORARY TABLE 275
- CREATE INDEX 280
- CREATE PROCEDURE 295
- CREATE STOGROUP 303
- CREATE SYNONYM 306
- CREATE TABLE 308
- CREATE TABLESPACE
 - description 327
- CREATE VIEW 341
- DECLARE CURSOR
 - description 347
- DECLARE STATEMENT 352
- DECLARE TABLE 354
- DELETE
 - description 357
- DESCRIBE 362
- DESCRIBE CURSOR
 - description 368
- DESCRIBE INPUT 370
- DESCRIBE PROCEDURE
 - description 372
- DROP
 - description 375
- END DECLARE SECTION 380
- EXECUTE 382
- EXECUTE IMMEDIATE 386
- EXPLAIN
 - description 388
- FETCH
 - description 397
- FOR 389
- GRANT 400
- INCLUDE
 - description 417
 - SQLCA 517
 - SQLDA 524

SQL statements (*continued*)

- INSERT
 - description 419
 - invocation 195
- LABEL ON 424
- LOCK TABLE 426
- OPEN
 - description 428
- PREPARE 433
- RELEASE 437
- remote execution
 - description 55
 - dynamic execution 56
 - static execution 56
- RENAME 440
- REVOKE 443
- ROLLBACK 460
- SELECT INTO 462
- SET CONNECTION 465
- SET CURRENT DEGREE 468
- SET CURRENT PRECISION 472
- SET CURRENT RULES 473
- SET CURRENT SQLID 474
- UPDATE
 - description 477
- WHENEVER 483
- SQLCA (SQL communication area)
 - contents 513
 - entry changed by UPDATE 480
 - INCLUDE statement 417
- SQLCABC field of SQLCA 513
- SQLCAID field of SQLCA 513
- SQLCODE 206
 - 626 206
 - 752 264
 - 900 265
 - 918 264
 - +100 199, 397, 421, 428, 462, 483
 - description 199
 - field of SQLCA 513
- SQLD field of SQLDA 363, 521
- SQLDA (SQL descriptor area)
 - clause of INCLUDE statement 417
 - contents 519, 520
- SQLDABC field of SQLDA 363, 521
- SQLDAID field of SQLDA 363, 521
- SQLDATA field of SQLDA 522
- SQLERRD(3) field of SQLCA 361
- SQLERRD(n) field of SQLCA 513
- SQLERRMC field of SQLCA 513
- SQLERRML field of SQLCA 513
- SQLERROR
 - clause of WHENEVER statement 483
 - column of SYSPACKAGE catalog table 576
- SQLERRP field of SQLCA 513
- SQLIND field of SQLDA 522
- SQLLEN field of SQLDA 363, 522
- SQLN field of SQLDA
 - description 363, 521
- SQLNAME field of SQLDA 363, 522
- SQLRULES
 - column of SYSPLAN catalog table 585
- SQLSTATE 206
 - '02000' 397, 421, 428, 462, 483
 - '55015' 206
 - description 199
 - field of SQLCA 513
- SQLTYPE field of SQLDA
 - description 521
 - values 363, 523
- SQLVAR field of SQLDA 363
- SQLWARN6 field of SQLCA 99
- SQLWARNING clause
 - WHENEVER statement 483
- SQLWARNn field of SQLCA 513
- SQTY column
 - SYSINDEXPART catalog table 570
 - SYSTABLEPART catalog table 603
- standard, SQL (ANSI/ISO)
 - description 3
 - SET CONNECTION statement 465
 - SQL-style comments 125
 - STDSQL precompiler option 124
- START_RBA column of SYSCOPY catalog table 554
- STARTDB privilege
 - GRANT statement 405
 - REVOKE statement 449
- STARTDBAUTH column of SYSDBAUTH catalog table 560
- state
 - application process 262
 - SQL connection 34
- statement
 - descriptions 21
 - See *also* SQL statements
 - operational form 21
 - preparation 21
 - source form 21
- STATEMENT clause of DECLARE STATEMENT statement 352
- static SQL
 - description 21, 195
 - invocation of SELECT statement 197
- STATS privilege
 - GRANT statement 405
 - REVOKE statement 449
- STATSAUTH column of SYSDBAUTH catalog table 560
- STATSTIME column
 - SYSOLDIST catalog table 547
 - SYSOLDISTSTATS catalog table 548

STATSTIME column (*continued*)
 SYSCOLSTATS catalog table 549
 SYSCOLUMNS catalog table 553
 SYSINDEXES catalog table 568
 SYSINDEXPART catalog table 570
 SYSINDEXSTATS catalog table 572
 SYSSTOGROUP catalog table 597
 SYSTABLEPART catalog table 604
 SYSTABLES catalog table 608
 SYSTABLESPACE catalog table 611
 SYSTABSTATS catalog table 613
 STATUS
 column of SYSPACKSTMT catalog table 581
 column of SYSSTMT catalog table 595
 STATUS
 column of SYSTABLES catalog table 607
 column of SYSTABLESPACE catalog table 610
 STAY RESIDENT clause
 CREATE PROCEDURE statement 300
 STAYRESIDENT column of SYSPROCEDURES
 catalog table 591
 STD SQL LANGUAGE field of panel DSNTIP4 124
 STDSQL option
 precompiler 124
 STGROUP column of SYSDATABASE catalog
 table 557
 STMT column of SYSPACKSTMT catalog table 581
 STMTNO column
 SYSPACKSTMT catalog table 581
 SYSSTMT catalog table 595
 STNAME column of SYSTABAUTH catalog table 600
 STOGROUP
 clause of ALTER DATABASE statement 203
 clause of ALTER INDEX statement 208
 clause of ALTER STOGROUP statement 214
 clause of ALTER TABLESPACE statement 238
 clause of CREATE DATABASE statement 273
 clause of CREATE INDEX statement 285, 287
 clause of CREATE STOGROUP statement 303
 clause of CREATE TABLESPACE statement 330,
 332
 clause of DROP statement 376
 STOGROUP privilege
 GRANT statement 415
 REVOKE statement 458
 STOPALL privilege
 GRANT statement 410
 REVOKE statement 454
 STOPALLAUTH column of SYSUSERAUTH catalog
 table 615
 STOPAUTH column of SYSDBAUTH catalog
 table 560
 STOPDB privilege
 GRANT statement 405
 REVOKE statement 449
 storage group, DB2
 altering 214
 creating 303
 defining 303
 description 27
 dropping 376
 storage structure 26
 stored procedure
 CALL statement 248
 creating
 CREATE PROCEDURE statement 295
 CURRENT PACKAGESET special register 471
 invoking 248
 STORNAME column
 SYSINDEXPART catalog table 570
 SYSTABLEPART catalog table 603
 STORTYPE column
 SYSINDEXPART catalog table 570
 SYSTABLEPART catalog table 603
 STOSPACE privilege
 GRANT statement 410
 REVOKE statement 454
 STOSPACEAUTH column of SYSUSERAUTH catalog
 table 615
 string
 columns 57
 comparison 72
 constant 75
 conversion 37
 delimiter
 COBOL 122
 controlling representation 122
 SQL 122
 description 37
 long
 column limitations 172
 columns 59
 use restrictions 59
 varying-length
 description 59
 string delimiter precompiler option 122
 STRIP function 155
 STYPE column of SYSCOPY catalog table 555
 SUBBYTE column of SYSSTRINGS catalog table 598
 SUBPAGES clause
 ALTER INDEX statement 211
 SUBPAGES clause of CREATE INDEX statement 290
 subquery
 description 86
 HAVING clause 178
 WHERE clause 177
 subselect
 CREATE VIEW statement 170, 343
 description 170
 example 178
 INSERT statement 170, 421

- substitution byte 38
- substitution character 71
- SUBSTR function 157
- SUM function 135
- synonym
 - defining 306
 - description 51
 - dropping 377
 - naming convention 50
 - qualifying a column name 84
- SYNONYM clause
 - CREATE SYNONYM statement 306
 - DROP statement 377
- syntax diagrams, how to read 4
- SYSADM authority
 - GRANT statement 410
 - REVOKE statement 454
- SYSADMAUTH column of SYSUSERAUTH catalog table 615
- SYSCTRL authority
 - GRANT statement 410
 - REVOKE statement 454
- SYSCTRLAUTH column of SYSUSERAUTH catalog table 616
- SYSENTRIES column
 - SYSPACKAGE catalog table 574
 - SYSPLAN catalog table 585
- SYSIBM.... catalog tables
 - See catalog tables
- SYSMODENAME column of LUNAMES catalog table 541
- SYSOPR authority
 - GRANT statement 410
 - REVOKE statement 454
- SYSOPRAUTH column of SYSUSERAUTH catalog table 615
- system
 - limits 505
- SYSTEM
 - column of SYSPKSYSTEM catalog table 583
 - column of SYSPLSYSTEM catalog table 589

T

- table
 - altering
 - ALTER TABLE statement 217
 - base table 22
 - creating
 - CREATE GLOBAL TEMPORARY TABLE statement 275
 - CREATE TABLE statement 308
 - description 22
 - designator 85
 - dropping
 - DROP statement 377

- table (*continued*)
 - empty table 22
 - joining 175
 - obtaining information with DESCRIBE 362
 - renaming
 - RENAME statement 440
 - restricting column values 26
 - result table 22, 430
 - temporary copy 430
 - temporary table 22
- TABLE
 - clause of COMMENT ON statement 255
 - clause of DECLARE TABLE statement 354
 - clause of DROP statement 377
 - clause of LABEL ON statement 424
- table check constraint 223
 - defining
 - ALTER TABLE statement 223
 - CREATE TABLE statement 319
 - deleting rows 360
 - description 26
 - inserting rows 422
 - SYSCHECKDEP catalog table 544
 - updating rows 481
- table name
 - naming convention 50
 - qualifying a column name 84
- table space
 - altering
 - ALTER TABLESPACE statement 233
 - catalog table 529
 - creating
 - CREATE TABLESPACE statement 327
 - implicitly 320
 - description 26
 - dropping 377
 - naming convention 50
- TABLESPACE
 - clause of ALTER TABLESPACE statement 233
 - clause of DROP statement 377
 - clause of GRANT statement 415
 - clause of REVOKE statement 458
- TABLESPACE privilege
 - GRANT statement 415
 - REVOKE statement 458
- TBCREATOR column
 - SYSCOLUMNS catalog table 550
 - SYSFIELDS catalog table 565
 - SYSINDEXES catalog table 567
 - SYSSYNONYMS catalog table 599
 - SYSTABLES catalog table 608
- TBNAME column
 - SYSCHECKDEP catalog table 544
 - SYSCHECKS catalog table 545
 - SYSCOLDIST catalog table 547
 - SYSCOLDISTSTATS catalog table 548

TBNAME column (*continued*)

- SYSCOLSTATS catalog table 549
- SYSCOLUMNS catalog table 550
- SYSFIELDS catalog table 565
- SYSFOREIGNKEYS catalog table 566
- SYSINDEXES catalog table 567
- SYSRELS catalog table 593
- SYSSYNONYMS catalog table 599
- SYSTABLES catalog table 608

TBOWNER column

- SYSCHECKDEP catalog table 544
- SYSCHECKS catalog table 545
- SYSCOLDIST catalog table 547
- SYSCOLDISTSTATS catalog table 548
- SYSCOLSTATS catalog table 549

TCREATOR column of SYSTABAUTH catalog table 600

temporary

- table copy 430

temporary table

- description 22

TEXT column

- SYSSTMT catalog table 595
- SYSVIEWS catalog table 618

three-part name

- description 32

TIME

- arithmetic 100
- data type
 - CREATE TABLE statement 313
 - DECLARE TABLE statement 354
 - description 62
- duration 97
- function 159
- strings 64

TIME FORMAT field of panel DSNTIP4 124

TIMEGRANTED column

- SYSCOLAUTH catalog table 546
- SYSDBAUTH catalog table 559
- SYSPLANAUTH catalog table 587
- SYSRESAUTH catalog table 594
- SYSTABAUTH catalog table 600
- SYSUSERAUTH catalog table 614

timestamp

- arithmetic 101
- data type 63
- duration 97
- strings 65

TIMESTAMP

- column of SYSCHECKS catalog table 545
- column of SYSCOPY catalog table 555
- column of SYSDATABASE catalog table 557
- column of SYSDBRM catalog table 562
- column of SYSPACKAGE catalog table 574
- column of SYSPACKAUTH catalog table 578
- column of SYSPACKLIST catalog table 580

TIMESTAMP (*continued*)

- column of SYSRELS catalog table 593
- data type
 - CREATE TABLE statement 313
 - DECLARE TABLE statement 354
 - description 63
 - function 160
- TNAME column of SYSCOLAUTH catalog table 546

TO

- clause of CONNECT (Type 1) statement 262
- clause of CONNECT (Type 2) statement 267
- clause of GRANT statement 401

token in SQL 45

TPN column of LOCATIONS catalog table 538

TRACE privilege

- GRANT statement 411
- REVOKE statement 454

TRACEAUTH column of SYSUSERAUTH catalog table 615

TRANSPROC column of SYSSTRINGS catalog table 598

TRANSTAB column of SYSSTRINGS catalog table 598

TRANSTYPE column of SYSSTRINGS catalog table 598

truncation

- numbers 66

truth table 118

truth valued logic 118

TSNAME column

- SYSCOPY catalog table 554
- SYSTABLEPART catalog table 603
- SYSTABLES catalog table 606
- SYSTABSTATS catalog table 613

TTNAME column of SYSTABAUTH catalog table 600

TYPE

- clause of CREATE INDEX statement 282
- column of SYSCOLDIST catalog table 547
- column of SYSCOLDISTSTATS catalog table 548
- column of SYSDATABASE catalog table 557
- column of SYSTABLES catalog table 606
- column of USERNAMES catalog table 620

TYPE 1 clause

- ALTER INDEX statement 211

TYPE 2 clause

- ALTER INDEX statement 211

TYPE column

- SYSTABLESPACE catalog table 611

U

- unary operation 94
- unconnected state 35
- UNION clause
 - duplicate rows 183
 - fullselect 183

- UNIQUE clause
 - CREATE INDEX statement 282
 - CREATE TABLE statement 313, 316
- unique index
 - description 23
- unique key
 - description 23
- UNIQUERULE column of SYSINDEXES catalog table 567
- unit of recovery
 - COMMIT statement 257
 - description 28
 - ROLLBACK statement 460
- unit of work
 - closes cursors 430
 - description 29
 - dynamic caching 436
 - ending 29, 257, 460
 - initiating 29
 - persistence of prepared statements 436
 - referring to prepared statements 433
- universal time, coordinated (UTC) 79
- unqualified object names 51
- UPDATE
 - statement
 - description 477
- UPDATE privilege
 - GRANT statement 413
 - REVOKE statement 457
- update rule 24, 480
- UPDATEAUTH column of SYSTABAUTH catalog table 601
- UPDATECOLS column of SYSTABAUTH catalog table 600
- UPDATES column of SYSCOLUMNS catalog table 551
- updating
 - rows in a table 477
- USA 63
 - See *also* *datetime*
- USEAUTH column of SYSRESAUTH catalog table 594
- USER
 - special register 83
- USERNAMES column
 - IPNAMES catalog table 537
 - LUNAMES catalog table 542
- USING clause
 - ALTER INDEX statement 208
 - ALTER TABLESPACE statement 237
 - CREATE INDEX statement 285, 287
 - CREATE TABLESPACE statement 329, 332
 - DESCRIBE statement 365
 - EXECUTE statement 382
 - OPEN statement 428
 - PREPARE statement 434

- USING DESCRIPTOR clause
 - EXECUTE statement 382
 - FETCH statement 398
 - OPEN statement 429
- UTC (universal time, coordinated) 79

V

- VALID column
 - SYSPACKAGE catalog table 574
 - SYSPLAN catalog table 584
- VALIDATE
 - column of SYSPACKAGE catalog table 574
 - column of SYSPLAN catalog table 584
- validation procedure 222
 - See *also* *validation routine*
- validation routine
 - VALIDPROC clause 222, 321
- VALIDPROC clause
 - ALTER TABLE statement 222
 - CREATE TABLE statement 321
- VALPROC column of SYSTABLES catalog table 606
- value
 - composite 23
 - SQL 57
- VALUE function 162
- VALUES
 - clause of CREATE INDEX statement 289
 - clause of INSERT statement 421
- VARCHAR
 - data type
 - CREATE TABLE statement 312
 - DECLARE TABLE statement 354
- VARGRAPHIC
 - data type
 - CREATE TABLE statement 312
 - DECLARE TABLE statement 354
 - function 164
- variable
 - host
 - referencing 89
 - SQL syntax 89
- VCAT
 - clause of CREATE STOGROUP statement 304
- USING clause
 - ALTER INDEX statement 208
 - ALTER TABLESPACE statement 237
 - CREATE INDEX statement 285, 287
 - CREATE TABLESPACE statement 330, 332
- VCATNAME column
 - SYSINDEXPART catalog table 570
 - SYSSTOGROUP catalog table 597
 - SYSTABLEPART catalog table 603
- VERSION
 - clause of DROP statement 377
 - column of SYSDBRM catalog table 563

VERSION (*continued*)
 column of SYSPACKAGE catalog table 576
 column of SYSPACKSTMT catalog table 581
 version identificaton, current server 263, 268
 version-id naming convention 50
 view
 creating
 CREATE VIEW statement 341
 description 27
 dropping
 description 377
 read-only 345
 using 345
 obtaining information with DESCRIBE 362
 VIEW clause
 CREATE VIEW statement 341
 DROP statement 377
 VOLID column of SYSVOLUMES catalog table 619
 VOLUMES clause of CREATE STOGROUP
 statement 303
 VPASSWORD column of SYSSTOGROUP catalog
 table 597
 VSAM (virtual storage access method)
 catalog 287
 See *also* integrated catalog facility
 password
 ALTER STOGROUP statement 214
 CREATE INDEX statement 291

W

WHENEVER statement
 description 483
 WHERE clause
 DELETE statement 358
 description 177
 search condition 177
 UPDATE statement 479
 WHILE statement
 example 503
 SQL procedure 503
 WITH clause
 DELETE statement 360
 INSERT statement 421
 SELECT INTO statement 463
 select-statement 191
 WITH GRANT OPTION clause of GRANT
 statement 401
 WITH HOLD clause of DECLARE CURSOR
 statement 348
 WITH PROCEDURE clause of ASSOCIATE
 LOCATORS statement 243
 WITH RETURN clause of DECLARE CURSOR
 statement 349
 WITH RR|RS|CS|UR 463
 See *also* WITH clause

WLM ENVIRONMENT clause
 CREATE PROCEDURE statement 299
 WLM_ENV column
 SYSPROCEDURES catalog table 591
 work file database
 creating 273
 WORKAREA column of SYSFIELDS catalog table 565

Y

YEAR function 166

We'd Like to Hear from You

DB2 for OS/390
Version 5
SQL Reference
Publication No. SC26-8966-03

Please use one of the following ways to send us your comments about this book:

- Mail—Use the Readers' Comments form on the next page. If you are sending the form from a country other than the United States, give it to your local IBM branch office or IBM representative for mailing.
- Fax—Use the Readers' Comments form on the next page and fax it to this U.S. number: 800-426-7773 or (408) 463-4393.
- Electronic mail—Use one of the following network IDs:
 - IBMMail: USIBMXFC @ IBMMAIL
 - IBMLink: DB2PUBS @ STLVM27
 - Internet: DB2PUBS@VNET.IBM.COM

Be sure to include the following with your comments:

- Title and publication number of this book
- Your name, address, and telephone number or your name and electronic address if you would like a reply

Your comments should pertain only to the information in this book and the way the information is presented. To request additional publications, or to comment on other IBM information or the function of IBM products, please give your comments to your IBM representative or to your IBM authorized remarketer.

IBM may use or distribute your comments without obligation.

Readers' Comments

DB2 for OS/390

Version 5

SQL Reference

Publication No. SC26-8966-03

How satisfied are you with the information in this book?

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Technically accurate	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Complete	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Easy to find	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Easy to understand	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Well organized	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Applicable to your tasks	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Grammatically correct and consistent	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Graphically well designed	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Overall satisfaction	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Please tell us how we can improve this book:

May we contact you to discuss your comments? Yes No

Name

Address

Company or Organization

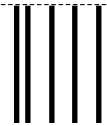
Phone No.



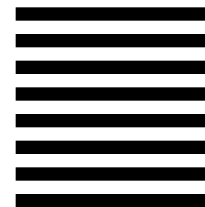
Fold and Tape

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
Department BWE/H3
PO Box 49023
San Jose, CA 95161-9945



Fold and Tape

Please do not staple

Fold and Tape



Program Number: 5655-DB2



Printed in the United States of America
on recycled paper containing 10%
recovered post-consumer fiber.

**DB2 for OS/390
Version 5**

SC26-8957 Administration Guide
SC26-8958 Application Programming and SQL Guide
SC26-8959 Call Level Interface Guide and Reference
SC26-8960 Command Reference
SC26-8961 Data Sharing: Planning and Administration
SX26-3841 Data Sharing Quick Reference
LY27-9659 Diagnosis Guide and Reference
LY27-9660 Diagnosis Quick Reference
GC26-8970 Installation Guide
GC26-8979 Master Index
SC26-8962 Messages and Codes
SC26-8964 Reference for Remote DRDA Requesters and Servers
SX26-3842 Reference Summary
SC26-8965 Release Guide
SC26-8966 SQL Reference
SC26-8967 Utility Guide and Reference
GC26-8971 What's New?

SC26-8966-03

