IBM DATABASE 2 Version 2

SC26-4380-1

SQL Reference

Release 2

SECOND EDITION (SEPTEMBER 1989)

This edition replaces and makes obsolete the previous edition, SC26-4380-0. Also replaced and obsolete is the associated technical newsletter, SN26-8224.

This edition applies to Release 2 of IBM DATABASE 2 Version 2, Program Number 5665-DB2, and to any subsequent releases until otherwise indicated in new editions or technical newsletters.

The changes for this edition are summarized under "Summary of Changes" in the first section of this publication. Specific changes are indicated by a vertical bar to the left of the 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.

Changes are made periodically to this publication; before using this publication in connection with the operation of IBM systems, consult the latest *IBM System/370, 30xx, 4300, and 9370 Processors Bibliography*, GC20-0001, for the editions that are applicable and current.

Requests for IBM publications should be made to your IBM representative or to the IBM branch office serving your locality. If you request publications from the address given below, your order will be delayed because publications are not stocked there.

A Reader's Comment Form is provided at the back of this publication. If the form has been removed, comments may be addressed to IBM Corporation, Department J57, P. O. Box 49023, San Jose, California, U.S.A. 95161-9023. IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

Contents

Chapter 1. Introduction

Statement of Purpose

DB2 includes a number of programming interfaces, some intended for general use and some that are product sensitive. We define those terms as follows:

General-Use Programming Interface

General-use programming interfaces are provided to allow you to write programs that use the services of IBM DATABASE 2.

End of General-Use Programming Interface

Product-Sensitive Programming Interface

Installation exits and other product-sensitive interfaces are provided to allow you to perform tasks such as product tailoring, monitoring, modification, or diagnosis. They are dependent on the detailed design or implementation of the product. Such interfaces should be used only for those 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 porduct releases or versions, or as a result of maintenance.

End of Product-Sensitive Programming Interface

A definitive description of each interface is included in an appropriate place in the DB2 library, as shown in the table that follows. (All book titles begin with *IBM DATABASE 2 Version 2*.)

Interface	**Type**	**Description is in ...**
Structured Query Language (SQL), including SQLCA and SQLDA	general use	*SQL Reference*
Call attachment facility	general use	*Application Programming and SQL Guide*
SQL return codes	product-sensitive	*Messages and Codes*
Interfaces to user-written exit routines	product-sensitive	*Administration Guide*
Instrumentation facility interface (IFI)	product-sensitive	*Administration Guide*
Commands used with IFI	product-sensitive	*Command and Utility Reference*
DB2 catalog	product-sensitive	*SQL Reference*
DSNHDECP load module	product-sensitive	DSNDDECP mapping macro
RDI parameter list	product-sensitive	DSNXRDI mapping macro

This book is intended to help you to write programs that include SQL statements. It contains a general-use programming interface, which allows you to write programs that use the services of DB2.

However, this book also provides information on the DB2 catalog, which is a product-sensitive programming interface; that information is explicitly identified where it occurs.

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 references to an IBM licensed program or other IBM product in this publication is not intended to state or imply that only IBM's program or other product may be used. Any functionally equivalent program may be used instead.

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 the IBM Director of Commercial Relations, IBM Corporation, Purchase, NY 10577.

Trademarks and Service Marks

The following term, used in the DB2 library, is a registered trademark or service mark of the IBM Corporation in the United States or other countries:

IBM

The following terms, used in the DB2 library, have been adopted by the IBM Corporation as trademarks or service marks in the United States or other countries:

DB2	MVS/XA	SQL/DS
DFSMS	QBE	System/370
MVS/DFP	QMF	3090
MVS/ESA	SAA	

About This Book

This book serves as a reference for the DB2 Structured Query Language (SQL).

Who This Book Is For

This book is intended as a reference book for end users, application programmers, system and database administrators, and for persons involved in error detection and diagnosis. Those who use it should already be acquainted with SQL: the book is not an introduction to the subject. For an introduction, read Section 2 of *Application Programming and SQL Guide*.

How This Book is Organized

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, "Concepts" on page 11 discusses the basic concepts of relational databases and SQL.

Chapter 3, "Language Elements" on page 17 describes the basic syntax of SQL and the language elements that are common to many SQL statements.

Chapter 4, "Functions" on page 65 contains syntax diagrams, semantic descriptions, rules, and usage examples of SQL column and scalar functions.

Chapter 5, "Queries" on page 83 describes the various forms of a query, which is a component of various SQL statements.

Chapter 6, "Statements" on page 95 contains syntax diagrams, semantic descriptions, rules, and examples of all SQL statements.

The appendixes contain information about SQL limits, SQLCA, SQLDA, catalog tables, the communications database, and SQL reserved words.

How to Use This Book

When you first use this book, consider reading Chapters 2 through 5 sequentially. Chapters 2 through 4 supply the conceptual framework for DB2 SQL, along with its elements and vocabulary. Chapter 5 covers the core of SQL—the select statement, through which users retrieve relational data.

The rest of the book is designed for the quick location of answers to specific SQL questions. Suppose, for example, that you need to know what authority you need to use the UPDATE statement on some table. Then you would look up this statement in Chapter 6, in which these statements are alphabetized, and you would find the information you need under *Authorization*.

How to Use the DB2 Library

The most rewarding task associated with a database management system is asking questions of it and getting answers, the task we call *end use*. But other tasks are also necessary—defining the parameters of the system, putting the data in place, and so on. We group the tasks associated with DB2 into the following major categories:

End use: End users want to issue SQL statements to retrieve data. Possibly they also insert, update, or delete data, still by means of SQL statements. They may need an elementary introduction to SQL, detailed instructions for using SPUFI, and an alphabetized reference to the types of SQL statements. Those are found in *Application Programming and SQL Guide* and this book.

The same users may issue SQL statements through QMF or some other program, and the library for that program may provide all the instruction or reference material they need. For a list of 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. Since they write SQL statements, they need *Application Programming and SQL Guide* and this book, just as end users do.

They 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. The material you need for writing a host program containing SQL is in *Application Programming and SQL Guide*. And for those handling errors, we also recommend *Messages and Codes*.

System and Database Administration: *Administration* covers almost everything else. *Administration Guide* (Volumes 1, 2 and 3) divides those tasks among the following sections:

- Section 2 (Volume 1) of *Administration Guide* deals with defining the DB2 system, estimating storage needs, and running the jobs that install the DB2 program.
- Section 3 (Volume 1) of *Administration Guide* explains how to connect several DB2 subsystems for communication using distributed data.
- Section 4 (Volume 2) 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 5 (Volume 2) 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 6 (Volume 2) 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 7 (Volume 3) 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, and interpreting DB2 trace output. The information previously in the separate book *IBM DATABASE 2 Version 2 Instrumentation Facility Interface* is now an appendix.

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

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

Diagnosis: A diagnostician detects and describes errors in the DB2 program. He or she may also recommend or apply a remedy. The documentation needed for the task is in *Diagnosis Guide and Reference* and *Messages and Codes*.

Titles of books in the DB2 library begin with IBM DATABASE 2 Version 2. However, references from one DB2 book to another are shortened and do not include the product title. Instead, they point directly to the section that holds the information. For a complete list of books in the DB2 library, and the sections in each book, see the back cover of this book.

Summary of Changes to DB2

Release 2

Version 2 Release 2 offers distributed database support. A DB2 user or application program connected to one DB2 subsystem can access data stored at another DB2 subsystem.

DB2 distributed database supports all of the function available in Version 2 Release 1, plus the following new features:

- **Authorization** is done at the system where the tables reside. For example, a user at Branch 1 wanting to access data at Branch 2 must be authorized at Branch 2 to do this. The DB2 system at Branch 2 identifies the Branch 1 user by an authorization identifier. If the Branch 1 user wants to select data from a Branch 2 table, authority to do so must be granted at Branch 2.
- **Auditing** of data is done at the site where the tables reside. So, if a user at Branch 1 is authorized to perform queries on Branch 2 tables, the work done on those tables is audited at Branch 2.
- **DB2 Governor** is extended to provide a system to control the amount of resources used by remote users. When a user from Branch 1 attempts to run queries on Branch 2 tables, the queries can be terminated by Branch 2 if the time limit is exceeded.
- **Data Manipulation** SQL statements are supported among DB2 systems. You can update, insert and delete data at different sites if your connection is through TSO or the call attachment facility (CAF). You can select data at different sites if your connection is through TSO, CAF, IMS, or CICS. Data definition SQL statements — GRANT, REVOKE, CREATE, DROP, and ALTER — are supported at the system where the tables reside.
  - **Remote Update Capabilities** are provided for a single site per commit scope with TSO and CAF. An authorized user at Branch 1 can update, insert, or delete data at Branch 2 if TSO or CAF is used. However, an update at any site, remote or local, must be committed before additional updates at a different site may be performed.

    This remote update capability is provided for DB2 users connected through TSO or CAF only. Users connected through IMS or CICS may not perform remote updates; however, they may update local data and they may select data from any site.
  - **Remote Query Capabilities** are provided for multiple sites per commit scope. SELECT statements may be made to the local and to multiple remote sites without committing data after each one. However, each SELECT statement must reference only one DB2 system. All DB2 users, connected through TSO, CAF, IMS or CICS, have this remote query capability.
- **Portability** of applications and data is easy between DB2 systems. Because of the three-part naming convention available with DB2 Version 2 Release 2, the end user or application program can point directly to data at a specific location. In addition, by defining alias table names, you can access data that frequently moves from one site to another.
- **Install, Migration, and Fall Back** are similar to past releases of DB2. However, a new option, the distributed data facility, must be selected at install or migration in order to use DB2 data in a distributed environment. In addition, a communications database must be created at install or migration so that DB2 systems can communicate with each other through VTAM. If the communications database is not created at install or migration, it can be

created by an authorized user after DB2 is operational. The distributed data facility cannot be started until the communications database has been created. Migration to DB2 Version 2 Release 2 is supported only from DB2 Version 2 Release 1 and from DB2 Version 1 Release 3. Fall back is supported from DB2 Version 2 Release 2 to DB2 Version 2 Release 1 and to DB2 Version 1 Release 3. You may only fall back to the release from which you migrated.

Release 1

Version 2 Release 1 offers all of the functions available with Version 1, plus:

- **Referential integrity**, which provides the ability to have DB2 ensure that references from one table to another are valid. You can define referential constraints which are automatically enforced on the tables to which they apply.
- **Performance** enhancements that significantly improve transaction, query, data definition, and utility response time. Not only is response time improved, but the amount of machine resource used by DB2 is reduced, freeing it for other purposes.
- **MVS/ESA** support enhancements which allow MVS/ESA customers to take advantage of improvements in performance and recovery for multiple address space operations. These improvements provide greater efficiency in hardware resource use from which DB2 directly benefits.
- **Application development and tuning flexibility** enhancements. Before loading data, you do not need to create indexes as often as you did in past releases of DB2. If and when additional indexes are needed, they can be created more quickly and they are well-organized.
- **Authorization control** that is more flexible. A user can be represented not only by a single (primary) authorization identifier, but also by one or more secondary identifiers, which can serve as group identifiers. This is helpful when using a security system like RACF. Users can also create objects to be owned by their secondary identifiers.
- **Audit trace** that allows you to determine who has accessed data stored in DB2 tables. The trace can record events of several types, such as unauthorized access attempts, write accesses, and read accesses.
- **Resource limit facility**, or **governor**, to control the amount of resources used when certain SQL queries are run. The governor terminates dynamic queries that exceed a predetermined time limit.
- **Recovery of data** enhancements. The RECOVER utility allows multiple table spaces and partitions to be recovered at the same time.
- **Recovery point in time**, with the addition of the QUIESCE utility. This allows you to establish a point at which a list of table spaces is consistent.
- **LOAD utility** enhancements in detecting and processing unique index violations. When the LOAD utility detects duplicate values for unique indexes, duplicate data is not loaded.
- **Segmented table spaces**, which provide performance advantages for storing more than one table in a single table space.
- **Alter storage attributes** ability. You can reassign table spaces, index spaces, and partitions to different storage groups or to user-managed data sets.
- **DFHSM** (Data Facility Hierarchical Storage Manager) support enhancements. You can perform synchronous automatic recall controlled by a time value you set.
- **DL/I batch** support. This programming enhancement allows you to access IMS data and DB2 data in the IMS batch environment.
- **National Language Support** to display DB2I help and task panels in Kanji.
- **Serviceability improvements** to decrease service costs and problem resolution time.
- A subset of **American National Standards Institute** (ANSI) SQL support.

Throughout this book, the term *MVS* is used to represent both MVS/Enterprise Systems Architecture (MVS/ESA) and MVS/Extended Architecture (MVS/XA). When it is necessary to make a technical distinction between the two environments, the specific term is used. *CICS* is used to represent both CICS/OS/VS and CICS/MVS; *IMS* is used to represent both IMS/VS and IMS/ESA; *C* and *C language* are used to represent the C/370 programming language.

Summary of Changes to This Book

Listed below are changes to the current edition of the book and to the previous edition.

Release 2

Many changes have been made to the current edition of the book to reflect the enhancements of DB2 Version 2 Release 2. Chief among them are the following:

Distributed Data Changes: The fact that processes operating under one DB2 subsystem can access data managed at another has produced many small changes throughout the book, plus the following major ones:

- The addition of a general description of the topic ("Distributed Data" on page 15).
- Changes to the descriptions of the special registers ("Special Registers" on page 41).
- The addition of a description of the communications database (Appendix D, "The Communications Database" on page 295).

Aliases: These afford a way of referring to tables and views managed at other (remote) DB2 subsystems. Among the alias-related changes are:

- The addition of descriptions for the CREATE ALIAS and DROP ALIAS statements.
- Changes in the description of various DB2 catalog tables to accommodate alias information. (Appendix C, "DB2 Catalog Tables").

New Precompiler Options: These are the NOFOR and STDSQL options. When used, they align the syntax rules for embedded SQL statements more closely with the rules for American National Standards Institute (ANSI) SQL.
("Standard SQL Language" on page 39 and "The NOFOR Option: FOR UPDATE OF" on page 40).

Index Key Statistics: The catalog table SYSIBM.SYSFIELDS, once used exclusively for field procedures, now contains statistical information for index keys as well. (Appendix C, "DB2 Catalog Tables").

Release 1

Major changes to the previous edition include the following:

- Chapter 5, "Queries" on page 83 was added to the book. Queries—the subject of that chapter—are fundamental to the use of SQL. Through them, data can be retrieved from relational databases. The information in this chapter was originally in a section entitled *SELECT*, in Chapter 6, "Statements."
- The section "SELECT INTO" on page 236 replaced the section titled *SELECT*. The replacement confines its discussion to the SELECT INTO statement.

- The authorization enhancements introduced in DB2 Version 2 Release 1 are reflected in many areas of the book. Most affected are the authorization discussions in the descriptions of the SQL statements. In addition, a description of the SET CURRENT SQLID statement was added to the book.

How to Read the Syntax Diagrams

Throughout this book, syntax is described using the structure defined as follows:

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

  The ►►— symbol indicates the beginning of a statement.

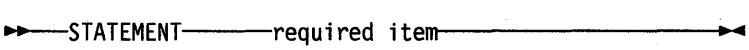
  The —► symbol indicates that the statement syntax is continued on the next line.

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

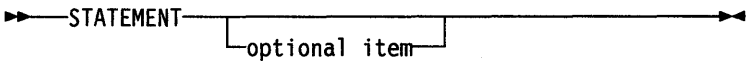
  The —►◄ symbol indicates the end of a statement.

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

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

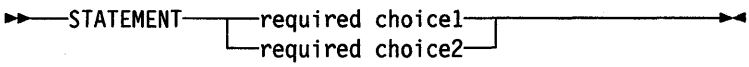


- Optional items appear below the main path.

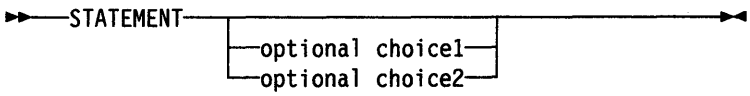


- If you can choose from two or more items, they appear in a stack.

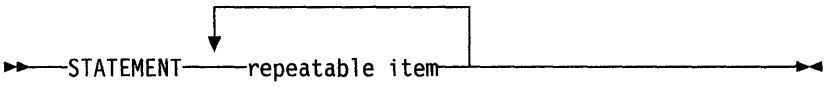
  If you *must* choose one of the items, one item of the stack appears on the main path.



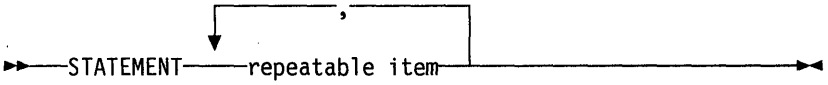
  If choosing none of the items is an option, the entire stack appears below the main path.



- An arrow returning to the left, above the main line, indicates an item that can be repeated. In this case, repeated items must be separated by one or more blanks.

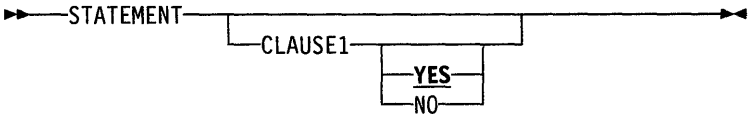


If the repeat arrow contains a comma, you must separate repeated items with a comma.



A repeat arrow above a stack indicates that you can make more than one choice from the stacked items or repeat a single choice.

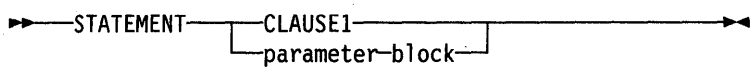
- Statements appear in uppercase (for example, CREATE TABLE).
- Keywords also appear in uppercase (for example, FROM). They must be spelled exactly as shown. Variables appear in lowercase (for example, column-name). They represent user-supplied names or values.
- A default parameter appears in boldface and is underscored. If you don't supply it in the statement, you will get the same result as if you had included it.



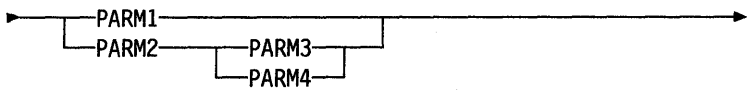
Defaults can be indicated only in the case of fixed default values. Variable default values are explained in the description of the statement. A case in point is the BUFFERPOOL parameter of the CREATE TABLESPACE statement. When this is omitted, the buffer pool used is the default for the corresponding database, and this was determined by the statement that created the database.

- If punctuation marks, parentheses, arithmetic operators, or other such symbols are shown, you must enter them as part of the syntax.

Sometimes a single variable represents a set of several parameters. For example, in the following diagram, the variable parameter-block can be replaced by any of the interpretations of the diagram that is headed **parameter-block**:



parameter-block




```
Example: STATEMENT CLAUSE1
     or  STATEMENT PARM1
     or  STATEMENT PARM2 PARM4
```

DB2 and Systems Application Architecture

System Application Architecture (SAA) is a definition — a set of software interfaces, conventions and protocols that provide a framework for designing and developing applications with cross-system consistency.

Systems Application Architecture:

- Defines a **common programming interface** that you can use to develop applications that can be integrated with each other and transported to run in multiple SAA environments
- Defines **common communications support** that you can use to connect applications, systems, networks, and devices
- Defines a **common user access** that you can use to achieve consistency in panel layout and user interaction techniques
- Offers some **common applications** written by IBM.

The SAA database interface is SQL. However, SAA SQL and DB2 SQL are not identical. SAA SQL is a subset of DB2 SQL. You should consult *SAA CPI Database Reference* if you intend to develop applications that fully adhere to the definition of SAA SQL.

Chapter 2. Concepts

Structured Query Language (SQL) is the language used to access data in a relational database. SQL is unlike many programming and data languages because you do not have to code a sequence of instructions explaining how to access the data. SQL allows you to select data by using a single statement directed to DB2. It is the function of DB2 to access and to maintain the data.

SQL provides full data definition and data manipulation capabilities. You can use it to define objects such as indexes, tables, and views. You can also retrieve, insert, update, and delete data, and control access authorization to data.

The SQL statements can be:

- Embedded inside application programs written in other languages, such as C, COBOL, FORTRAN, Assembler, and PL/I.

  This is called *static* SQL. The SQL statements are present in the program at the time it is precompiled.

- Typed in from a terminal or built by a program.

  This is termed *interactive* or *dynamic* SQL. The SQL statements are not provided to DB2 until the program runs.

Static SQL

SQL programmers can write source programs containing static SQL statements. Before a program containing static SQL statements is compiled, the DB2 precompiler flags the SQL statements as comments and includes the code necessary to invoke DB2. Then the compiler can process the program. The precompiler also checks the syntax of the SQL statements.

Application program preparation requires several steps including precompilation and compilation. For more on these steps, see Section 3 of *Application Programming and SQL Guide*.

Dynamic SQL

A capability for processing SQL statements entered from a terminal is part of the architecture of DB2. You can write programs that read SQL statements from terminals; in this sense, the process is similar to the SQL-handling processes used by DB2's interactive facilities. Dynamic SQL allows you to create your own query programs, tailored to your users and designed for your specific needs.

Tables

A relational database is a set of *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.

A *base table* is created with the CREATE TABLE statement and is used to hold persistent user data. A *result table* is a set of rows that DB2 selects or generates from one or more base 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 table columns. An *index* is an object that is separate from the data in the table. When you request an index DB2 builds this structure and maintains it automatically.

Indexes are used by DB2 to:

- Improve performance. In most cases, access to data is faster than without an index.
- Ensure uniqueness. A table with a unique index cannot have rows with identical keys. (A key is a column, or an ordered collection of columns, on which the index is created.)

Views

Views provide an alternative way of looking at the data in one or more tables.

Like tables, views have rows and columns with no inherent order of rows. You specify view names in the FROM clause of the SELECT statement just as you specify table names. You can create views and authorize their use by users who use them like tables. However, certain operations are not valid on views.

A table has a storage representation, but a view does not. When a view is created, its definition is stored in the catalog. No data is stored and, therefore, no index can be created for a view. However, an index created for a table on which a view is based may improve the performance of operations on the view.

Views may be used to:

- Control access to a table. Access to a view of a table can be granted on the view without granting access to the table itself. The view could show only portions of data in the table, thereby screening out sensitive data.
- Make data easier to use. For example, a view could show summary data for a given table, combine two or more tables in meaningful ways, or show only rows that are pertinent to the process using the view.

Catalog

The database manager maintains a set of tables containing information about the data the manager controls. These tables are collectively known as the *catalog*. The *catalog tables* contain information about DB2 objects such as tables, views, and indexes.

Tables in the catalog are like any other database tables. If you have authorization, you can use SQL statements to look at data in the catalog tables in the same way you retrieve data from any other table in the system. The database manager ensures that the catalog contains accurate descriptions of the databases at all times.

Note: In this book, the terms "the catalog" and "the local catalog" are reserved for the DB2 catalog that your local DB2 subsystem maintains. DB2 catalogs maintained by other DB2 subsystems are known as "remote catalogs," and the catalogs maintained by AMS (Access Method Services) are known as ICF catalogs.

For further information about the DB2 catalog, see Appendix C, "DB2 Catalog Tables" on page 257.

Locking and Recovery

DB2 provides locking and recovery facilities for the databases that it controls. An important aspect of these facilities are the commit and rollback operations described below. The interface used by an SQL program to explicitly specify these 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.

In every environment, an SQL program executes as part of a process. The word 'process' is a general term for a unit that is dependent on the environment, but which 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. Different processes may involve the execution of different programs or different executions of the same program. The means of initiating and terminating a process are dependent on the environment.

Multiple processes may request access to the same data at the same time. The mechanism used to maintain integrity under such conditions is called locking. For example, locks are used to prevent two processes from updating the same row at the same time.

An important aspect of locking is that DB2 implicitly acquires locks on behalf of processes such that the uncommitted changes of any process can only be perceived by that process. All locks acquired on behalf of a process are implicitly released when it terminates, but locks can also be released sooner at the explicit request of the process. This operation is called commit.

DB2 also provides a means by which the uncommitted changes of a process can be backed out. This is done in the event of a failure of the process or a deadlock situation, and can also be done at the explicit request of the process. This operation is called rollback.

A unit of recovery is a recoverable sequence of operations within a process. At any time, a process is a single unit of recovery, but the life of a process may involve many units of recovery as a result of commit or rollback operations.

A unit of recovery is initiated by the initiation of a process or by the termination of a previous unit of recovery. A unit of recovery is terminated by a commit operation, a rollback operation, or the termination of a process. A commit or rollback operation only affects the results of operations executed within the unit of recovery that it terminates. Uncommitted database changes made in a unit of recovery cannot be perceived by other processes and can be backed out. Committed database changes can be perceived by other processes and cannot be backed out.

A unit of recovery is sometimes called a *logical unit of work*. This reflects the assumption that the initiation and termination of a unit of recovery are points of consistency. For example, a banking transaction might involve the transfer of funds from account A to account B. The amount is subtracted from A and then added to B. After subtracting, the data is inconsistent. It is consistent again only after the amount is added to B. When both steps are complete, the commit operation can be used to terminate the unit of recovery and make the changes available to other processes. If a failure occurs before the unit of recovery is terminated, the uncommitted changes are backed out to restore the consistent state that (presumably) existed when the unit of recovery was initiated.

Authorization and Privileges

Before it can execute a specific SQL statement, a process must have appropriate DB2 authority. A process derives this authority from its *authorization IDs* These supply the needed authority in the following ways:

- By their ownership of objects referred to in the statement
- By their possession of various DB2 authorities and privileges.

For more on DB2 authority and its relationship to authorization IDs, see "Authorization IDs" on page 22.

Authority requirements vary with the type of statement. Requirements for a given type of statement are discussed in the description of that statement type in Chapter 6, under *Authorization*.

DB2 authorization plays a key role in establishing security and integrity for a DB2 subsystem. For details on this role, see Section 5 (Volume 2) of *Administration Guide*. DB2 authority is distributed by executing SQL GRANT and REVOKE statements. For a description of these statements, and of the authorities and privileges that can be granted and revoked, see Chapter 6, "Statements" on page 95.

Storage Structures

DB2 storage structures are managed by Access Method Services (AMS). Related DB2 objects include:

- *table spaces*, in which one or more tables, or a portion of a single table, are kept.
- *index spaces*, which contain indexes for the DB2 subsystem's tables.
- *databases*, from each of which a set of table spaces and indexes are managed.
- *Storage groups*, which list the DASD volumes that designated table spaces and index spaces can use.

For more on DB2 structures, see Section 4 (Volume 2) of *Administration Guide*.

Distributed Data

DB2 allows a group of DB2 subsystems to have access to each other's relational data. For example, a process connected to one subsystem could, with appropriate DB2 authority, retrieve data from a table at another subsystem, or insert new rows. The data at all the subsystems of a group is known as the *distributed data* for that group.

Distributed Data Management

The DB2 subsystems are part of a VTAM network. Managing this network for DB2 is the *distributed data facility* (DDF). For each participating subsystem, the DDF obtains needed information from the subsystem's *communication database* (CDB). For more on the CDB, see Appendix D, "The Communications Database" on page 295.

Terminology: "Local" and "Remote"

From the viewpoint of a process, the subsystem to which it is connected is the *local subsystem*. The rest are *remote subsystems*. "Local" and "remote" are also applied to DB2 objects. A *local table*, for instance, is managed by the local subsystem. A *remote view* is defined at a remote subsystem and is recorded in that subsystem's catalog.

Allowable SQL Statements

Access to remote objects is restricted to tables and views. With appropriate DB2 authority, a transaction can use static and dynamic SQL statements that retrieve, delete, and insert rows, update columns, and commit or rollback changes against remote objects. SQL statements of other types cannot refer to remote DB2 objects. A process cannot, for example, create remote objects or alter their structures, nor can it grant or revoke privileges that apply to remote subsystems.

DB2 Authority for Remote Access

DB2 authority to access a remote object must be recorded in the catalog of the subsystem that manages the object. All privileges in this catalog are for objects local to that subsystem. Only a process connected to this subsystem can execute the pertinent SQL GRANT statement.

The record for a privilege for a given authorization ID does not identify a DB2 subsystem. As a result, a privilege afforded a given authorization ID applies to any process operating with this authorization ID, regardless of its local subsystem.

Committing and Rolling Back Changes

All changes within a unit of recovery must be to objects managed by a single DB2 subsystem: Before a process can make changes for another subsystem, it must commit or roll back the changes it has made for the current subsystem. This rule applies to local changes as well as remote ones, with the following important implications:

- No changes can be made to remote objects until any local changes have been rolled back or committed.
- A number of SQL statements are restricted to changes to the local subsystem. None of these statements will execute if there are uncommitted changes to remote objects. The following are statements of this kind:

- All CREATE, DROP, and ALTER statements, which create or delete local objects or alter their structures
- The COMMENT ON and LABEL ON statements, which record information in the local catalog about tables, views, aliases, and columns.
- The GRANT and REVOKE statements, which record grants and revocations of DB2 privileges in the local catalog.

Additional Restrictions

Additional restrictions for statements that refer to remote objects are as follows:

- Processes connected to CICS or IMS cannot execute INSERT, UPDATE, or DELETE statements.
- All objects referenced in a single statement must be managed at the same subsystem. This precludes, for example, referring to both remote and local objects in the same SQL statement.
- Strings transmitted from one DB2 subsystem to another are not translated. This fact should be taken into account if the two subsystems have different internal representations for characters. For example, a language-dependent character at a French-Canadian subsystem may have a different representation at a Brazilian subsystem, or may have no representation at all. Under these or similar conditions, it is possible, for example, for one subsystem to introduce inaccurate or incoherent character data into tables at another subsystem. And it is possible to retrieve data from these tables that won't display properly on local terminals.

  Before you modify data at a remote subsystem, be sure to understand the differences between your character representations and theirs. Then avoid, if possible, the use of any characters that have different representations at the two subsystems.

Chapter 3. Language Elements

This chapter defines the basic syntax of SQL and language elements that are common to many SQL statements. Although examples are shown and most terms are defined before they are used, this chapter is not a tutorial. It is intended for those who require a definition of the following language elements:

Characters

The basic symbols of the language are EBCDIC characters. Characters are classified as letters, digits, or special characters. A *letter* is any one of the uppercase characters A through Z plus the three characters reserved as alphabetic extenders for national languages (#, @, and $ in the United States). A *digit* is any one of the characters 0 through 9. A *special character* is any character other than a letter or a digit.

The language may also include double-byte characters. If mixed data is not in effect, double-byte characters are only recognized in graphic string constants. If mixed data is in effect, double-byte characters are also recognized within character 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 in effect, 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 blanks and characters within a string constant or delimited identifier. (These terms are defined later.)

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

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

- 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 under "PREPARE" on page 218.

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 may 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, that ordinary token must be followed by a space.

Uppercase and Lowercase

Any token may 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 or the CHARACTER SET installation option is set to KATAKANA when the statement is parsed. Delimiter tokens are never folded to uppercase. Thus, the statement:

```
select * from DSN8220.EMP where lastname = 'Smith';
```

is equivalent, after folding, to:

```
SELECT * FROM DSN8220.EMP WHERE LASTNAME = 'Smith';
```

Identifiers

An *identifier* is a token that is used to form a name. An identifier in an SQL statement is an SQL identifier, a location identifier, or a host identifier.

SQL Identifiers

There are two types of SQL identifiers: ordinary identifiers and delimited 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 must not be identical to a reserved word. (For a list of reserved words, see Appendix E, "SQL Reserved Words" on page 299.)

  If the CHARACTER SET option is set to KATAKANA when a statement is parsed, an ordinary identifier may include Katakana characters.

- A *delimited identifier* is a sequence of one or more characters enclosed within SQL escape characters. A delimited identifier can be used when the sequence of characters doesn't qualify as an ordinary identifier. Such a sequence, for example, could be an SQL reserved word, or it could begin with a number. Two consecutive escape characters are used to represent one escape character within the delimited identifier. The escape character is the quotation mark (") except for:

  - Dynamic SQL when the SQL STRING DELIMITER installation option is set to the quotation mark. Here the SQL escape character is the apostrophe (').
  - COBOL application programs. A COBOL precompiler option specifies whether the escape character is the quotation mark (") or the apostrophe (').

Examples: WEEKLYSAL WEEKLY_SAL "WEEKLY SAL" $500

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.

Restrictions on what can appear in an ordinary identifier were discussed at the start of this section. They apply to both long and short ordinary identifiers. In contrast, any character can appear within a long delimited identifier. This is also true for a short delimited identifier, with an additional restriction: A short delimited identifier cannot begin with an underscore or a number. For delimited identifiers of either type, periods, leading blanks and trailing blanks should be avoided.

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. However, a *host-identifier* must not include DBCS characters.

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 can be local or remote, and the *alias-name* can be used wherever the *table-name* or *view-name* can be used to reference 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 a remote alias, but the table or view that the remote alias represents must then be local to the DB2 subsystem at which the remote alias is defined.

authorization-name A short identifier that designates a set of privileges. It may also designate a user or group of users, but this property is not controlled by DB2.

catalog-name A short identifier that designates an ICF catalog.

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.

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.

descriptor-name A *host-identifier* that designates an SQL descriptor area (SQLDA). The host identifier may be preceded by a colon.

host-variable A sequence of tokens that designates a host variable. A *host-variable* includes at least one *host-identifier*, as explained in "Host Variables" on page 46.

index-name A qualified or unqualified name that designates an index. The unqualified form of an *index-name* is a long identifier. An unqualified *index-name* in an SQL statement is implicitly qualified by the authorization ID of that statement. The qualified form is a short identifier followed by a period and a long identifier.

location-name A location identifier that designates a DB2 subsystem.

plan-name A short identifier that designates an application plan.

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.

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

table-name A qualified or unqualified name that designates a table. A fully qualified *table-name* is a three-part name. The first part is a *location-name* that designates the DB2 subsystem 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 be specified between each of the parts.

A two-part *table-name* is implicitly qualified by the *location-name* of the local DB2 subsystem. A period must be specified between the two parts. A one-part or unqualified *table-name* is a long identifier with two implicit qualifiers. The first implicit qualifier is the *location-name* of the local DB2 subsystem. The second is an authorization ID. In a static SQL statement, the authorization ID is the owner of the plan. In a dynamic SQL statement, the authorization ID is the SQL authorization ID.

tablespace-name A short identifier that designates a table space of an identified database. If a database is not identified, a *tablespace-name* designates a table space of database DSNDB04.

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 DB2 subsystem at which the view definition is stored. The second part is the authorization ID that designates the owner of the view. The third part is a long identifier. A period must be specified between each of the parts.

A two-part *view-name* is implicitly qualified by the *location-name* of the local DB2 subsystem. A period must be specified between 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 local DB2 subsystem. The second is an an authorization ID. In a static SQL statement, the authorization ID is the owner of the plan. In a dynamic SQL statement, the authorization ID is the SQL authorization ID.

Aliases and Synonyms

A table or view can be referenced 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 explictly 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 referenced in an SQL statement, with two exceptions: an 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 designate a local table or view, and the synonym is defined on the name of that table or view. If a synonym is used in CREATE ALIAS, the alias is defined on the name of the local table or view designated 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 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 local or remote table or view. A synonym can only be defined on the name of a local table or view.
- 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 a local alias. (Note, however, that the name can designate a remote alias if that alias represents a table or view at the same remote location.) When a synonym is used, this error cannot occur.

Authorization IDs

An *authorization ID* is a character string that designates a set of privileges. It may also designate a user or a group of users, but this property is not controlled by DB2.

Authorization IDs are used by DB2 to provide:

- authorization checking of SQL statements, and
- implicit qualifiers for the names of local 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. The use of the primary and secondary authorization IDs depends on whether the process creates a plan or uses a plan.

Authorization IDs and Bind

A process that creates a plan is called a bind process. The connection with DB2 is the result of the execution of a BIND or REBIND command. Both commands allow for the specification of the authorization ID of the owner of the plan. The specified authorization ID must be one of the authorization IDs of the process. The default owner for BIND is the primary authorization ID. The default owner for REBIND is the previous owner of the plan (ownership is unchanged if an owner is not explicitly specified).

The ID that is specified as owner must be one of the authorization IDs of the process, unless one of the authorization IDs of the process has SYSADM authority. In this case, the owner can be set to any value.

For static SQL statements that refer to local objects, it is the authorization ID of the owner of the plan that is used for authorization checking and implicit qualification. The plan owner must have all appropriate privileges as well as the authority to issue the BIND subcommand.

For static SQL statements that refer to remote objects, authorization checking is deferred until run time. For more information on this, see "Authorization IDs and Remote Execution" on page 24.

If a plan is bound with VALIDATE(BIND), all local objects referred to in the SQL statements of the plan, and all privileges required for these statements, must exist at bind time. If any of these objects or privileges does not exist, the bind operation is unsuccessful. This does not, however, apply to remote objects. For these, existence and authority checking is deferred until run time.

If a plan is bound with VALIDATE(RUN), authorization checking is still performed at bind time, but referenced objects and required privileges 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 recovery and all privileges required for the statement must exist at that time. If any privilege does not exist, execution of the statement is unsuccessful. Note that when the authorization check is performed at execute time, it is performed against the plan owner, not the executor. For the effect of this option on cursors, see "DECLARE CURSOR" on page 165.

Authorization IDs and Dynamic SQL

The discussion below applies to dynamic SQL statements that refer to local objects. For those that refer to remote objects, see "Authorization IDs and Remote Execution" on page 24.

A process that uses a plan is called an application process. As described below, the authorization IDs of an application process are used for dynamic SQL statements in much the same way that the authorization IDs of a bind process are used for static SQL statements. The authorization ID that is analogous to the authorization ID of the owner of the plan is called the *SQL authorization ID*.

At any time, the SQL authorization ID of a process is the value of CURRENT SQLID. This is an SQL special register that can be initialized by the authorization exit. If it is not initialized by the exit, its initial value is the primary authorization ID of the process. The value of CURRENT SQLID can be changed by the execution of a SET CURRENT SQLID statement. Unless some authorization ID of the process has SYSADM authority, the new value is constrained to 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 a CREATE, GRANT, or REVOKE statement, each privilege required for the statement can be a privilege that is designated by any authorization ID of the process. Thus the privilege set that applies to these

statements is the union of the privileges designated by each authorization ID of the process. This is called *composite privileges*.

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. Thus each privilege required for the statement must be a privilege that is designated by that single authorization ID. As explained in the description of CREATE INDEX, CREATE TABLE, and CREATE VIEW, another authorization ID may be used when a qualified name is specified in these statements. However, this exception is not the same as composite privileges.

Authorization IDs and Remote Execution

In an SQL statement that refers to a remote object, every referenced object must be remote and managed by the same remote subsystem. For execution, this statement is sent to the remote subsystem, along with two authorization IDs for the process. One of these is the primary authorization ID. The other is the owner of the application plan. The first is used for authority checking when the statement to be executed is dynamic. The second is used when this statement is static.

Before either authorization ID is used, it may be translated once or twice: once at the local subsystem before it is sent, and once by the remote subsystem after it is received. Suppose, for example, that a statement to be executed dynamically at a remote subsystem has JONES as its primary authorization ID. Then, before it is sent, JONES could be translated to, say, SJJONES. And, after it is received, SJJONES could be translated to USERCLX. The authorization ID to be used for authority checking is therefore USERCLX.

The authorization check is conducted against the system catalog for the remote subsystem. No other system catalog is involved. The requisite authority must come from GRANT statements executed by a process connected to the remote subsystem.

Controlling the translation of authorization IDs are tables in the remote and local *communications databases*. These are part of the security mechanism that DB2 provides for remotely executed SQL. The database is described in Appendix D, "The Communications Database" on page 295. The security mechanism is discussed in Section 5 (Volume 2) of *Administration Guide*.

Please note that the security mechanism could reject a remotely executed statement, regardless of the DB2 authorities recorded in the remote system catalog. For example, "come from" processing at a given subsystem could reject all SQL statements that originate at a specific subsystem if their authority-checking authorization IDs do not appear in a certain table of the receiving subsystem's communication database (CDB).

Examples

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 log-on and that you execute the following statement interactively:

```
GRANT SELECT ON TDEPT TO KEENE;
```

Also assume that your site has not replaced the default connection authorization exit and that you have not executed a SET CURRENT SQLID statement. Thus, when the GRANT statement is prepared and executed by SPUFI, the SQL

authorization ID of the process 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.

Here are two examples of names that could represent a table, view, index, or alias:

```
NAME1     SMITH.NAME1
```

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.

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, and special registers.

The basic data types are character string, graphic string, binary integer, floating-point, decimal, date, time, and timestamp. Character and graphic strings are further classified as fixed-length or varying-length, and varying-length strings are classified as short or long. Floating-point values are further classified as single precision and double precision, while integers are further classified as small integer and large integer.

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.

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.* This value should not be confused with the null value.

Fixed-Length 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 254 inclusive. Therefore every fixed-length string column is a *short string column.*

Varying-Length Strings

All values of a varying-length string column have the same maximum length, which is determined by the length attribute of the column. If the length attribute is greater than 254, the column is a *long string column.* Long string columns cannot be referenced in:

- A function other than SUBSTR or LENGTH
- 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 LIKE

String Variables

Fixed-length string variables can be defined in all host languages. (In C they 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, they are simulated as described in Section 3 of *Application Programming and SQL Guide*. In C they may also be represented in the form of null-terminated strings. String variables with values longer than 254 bytes are subject to the same restrictions as long string columns.

Mixed Data in Character Strings

Character strings may contain sequences of double-byte characters, each sequence preceded by a "shift-out" character and followed by a "shift-in" character. A string containing one or more such sequences is called "mixed." The principal use of mixed data is to represent national language texts.

SQL does not recognize subclasses of double-byte characters, and does not assign any specific meaning to particular double-byte codes. However, if you choose to use mixed data, then two single-byte EBCDIC codes are given special meanings:

- X'0E', the "shift-out" character, is used to mark the beginning of a sequence of double-byte codes.
- X'0F', the "shift-in" character, is used to mark the end of a sequence of double-byte codes.

In order for DB2 to recognize double-byte characters in a mixed string, three conditions must be present:

1. For dynamically prepared SQL statements, the MIXED DATA install option must have the value YES.
2. For static SQL statements, the GRAPHIC/NOGRAPHIC precompiler option must have been specified as GRAPHIC. If neither GRAPHIC nor NOGRAPHIC is specified, the installation option serves as default.
3. Within the string, the double-byte characters must be enclosed between paired shift-out and shift-in characters.

   The pairing is detected as the string is read from left to right. The code X'0E' is interpreted as a shift-out character if X'0F' occurs later; otherwise it is invalid. The first X'0F' following the X'0E' is the paired shift-in character.

   There must be an even number of bytes between the paired characters, and each pair of bytes is considered to be a double-byte character. There may be more than one set of paired shift-out and shift-in characters in the string.

If these conditions are present, we say that 'mixed data is in effect'.

The length of a mixed string is its total number of bytes, counting two bytes for each double-byte character and one byte for each shift-out or shift-in character.

Mixed Data in Delimited Identifiers

A long delimited identifier may also contain sequences of double-byte characters. Each sequence must begin with the single-byte shift-out character, and end with the single-byte shift-in character. The shift-out and shift-in characters are considered part of the identifier.

Graphic Strings

A *graphic string* is any sequence of double-byte characters (and does not include shift-out or shift-in characters). The length of the string is the number of its characters. Like character strings, graphic strings may be empty.

All values of a fixed-length graphic column have the same length, given by the length attribute of the column. The length attribute cannot be greater than 127. Therefore every fixed-length graphic string column is a short string column.

All values of a varying-length graphic column have the same maximum length, given by the length attribute of the column. If that is greater than 127, the column is a long string column. Long graphic-string columns are subject to the same limitations that apply to long character-string columns. Graphic variables with values longer than 127 characters are subject to the same restrictions as long string columns. In all cases, the length control field of a varying length graphic string indicates the number of characters, not bytes.

Graphic variables can be defined only in COBOL, PL/I, and assembler.

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.

All numbers have a sign and a precision. When a number in a column is 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.

Binary integer and floating-point variables can be defined in all host languages. Decimal variables can be defined in all host languages except C and FORTRAN. In COBOL, decimal numbers may be represented in the packed decimal format used for columns or in the format denoted by DISPLAY SIGN LEADING SEPARATE.

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 approximately 5.4E-79 to 7.2E+75.

Double Precision Floating-Point

A *double precision floating-point* number is 64 bits long. The range of magnitude is approximately 5.4E-79 to 7.2E+75.

Decimal

A *decimal* value 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 15 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 the absolute value of *n* is the largest number that can be represented with the applicable precision and scale. The maximum range is -999999999999999 to +999999999999999.

Date/Time Values

The date/time data types are described below. Although date/time values can be used in certain arithmetic and string operations and are compatible with certain strings, they are neither strings nor numbers. However, strings can represent date/time values; see "String Representations of Date/Time Values" on page 29.

Date

A *date* is a three part value (year, month, and day) designating a point in time under the Gregorian calendar, which is assumed to have been in effect from the year 1 A.D.[1] 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 *x*, where *x* depends on the month.

The internal representation of a date is a string of four bytes. Each byte consists of two packed decimal digits. The first two 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 four bytes, representing the internal length. The length of a DATE column as described in the SQLDA is ten bytes, unless your site specified a date installation exit when DB2 was installed. (The exit is discussed in an appendix of *Administration Guide*.) In this case, the string format of a date may be up to 254 bytes in length. Accordingly, DCLGEN defines fixed-length string variables for DATE columns with the length specified by the LOCAL DATE LENGTH install option, or a length of ten bytes if the install option was not specified.

---

[1] Note that historical dates may not 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.

Time

A *time* is a three part value (hour, minute, and second) designating a time of day under a 24 hour clock. The range of the hour part is 0 to 24, while the range of the other parts is 0 to 59. If the hour is 24, the minute and second specifications will both be zero.

The internal representation of a time is a string of three bytes. Each byte is 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 three bytes, representing the internal length. The length of a TIME column as described in the SQLDA is eight bytes, unless your site specified a time installation exit when DB2 was installed. (The exit is discussed in an appendix of *Administration Guide*.) In this case, the string format of a time may be up to 254 bytes in length. Accordingly, DCLGEN defines fixed-length string variables for TIME columns with the length specified by the LOCAL TIME LENGTH install option, or a length of eight bytes if the install option was not specified.

Timestamp

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

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

The length of a TIMESTAMP column as described in the catalog is ten bytes, representing the internal length. The length of a TIMESTAMP column as described in the SQLDA is twenty-six bytes. DCLGEN therefore defines twenty-six byte fixed-length string variables for TIMESTAMP columns.

String Representations of Date/Time Values

Values whose data types are DATE, TIME, or TIMESTAMP are represented in an internal form that is transparent to the user of SQL. Dates, times, and timestamps can, however, also be represented by character strings, and these representations directly concern the user of SQL because there are no special SQL constants for date/time values and no host variables with a data type of date, time, or timestamp.

For retrieval, date/time 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 or installation option. When a string representation of a date/time value is used in other operations, it is converted to a date/time value. However, this can only be done if the string representation is recognized by DB2 or an exit provided by the site and the other operand is a compatible date/time value. An input string representation of a date or time value with LOCAL specified can be any DB2 short character string. The sections that follow describe the string formats that are recognized by DB2.

Date Strings: A string representation of a date is a character string that starts with a digit and has a length of at least eight characters. Trailing blanks may be included. Leading zeros may be omitted from the month and day portions.

Valid string formats for dates are listed in Table 1 on page 30. Each format is identified by name, and includes an associated abbreviation (for use by the CHAR function) and an example of its use. For a site-defined date string format, the format and length must have been specified when DB2 was installed. They cannot be listed here.

Table 1. 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
Site-defined	LOCAL	Any site-defined form	—

Time Strings: A string representation of a time is a character string that starts with a digit, and has a length of at least four characters. Trailing blanks may be included; a leading zero may be omitted from the hour part of the time, and seconds may be omitted entirely. If you choose to omit seconds, an implicit specification of zero seconds is assumed. Thus 13.30 is equivalent to 13.30.00.

Valid string formats for times are listed in Table 2. 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 a site-defined time 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 Times

Format Name	Abbreviation	Time Format	Example
International Standards Organization	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
Site-defined	LOCAL	Any site-defined form	—

In the case of the 'IBM USA standard' time string format, the minutes specification may be omitted, indicating an implicit specification of 00 minutes. Thus 1 PM is equivalent to 1:00 PM. In the IBM USA standard, the letters A, M, and P may be lowercase.

In the USA time 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 corresponds to 00.01.00 through 00.59.00.

01:00 AM through 11:59 AM corresponds to 01.00.00 through 11.59.00.

12:00 PM (noon) through 11:59 PM corresponds to 12.00.00 through 23.59.00.

12:00 AM (midnight) corresponds to 24.00.00 and 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 sixteen characters. The complete string representation of a timestamp has the form *yyyy-mm-dd-hh.mm.ss.nnnnnn*. Trailing blanks may be included. Leading zeros may be omitted from the month, day, and hour part of the timestamp, and microseconds may be truncated or entirely omitted. If you choose to omit any digit of the microseconds portion, an implicit specification of zero is assumed. Thus, *1987-3-2-8.30.00* is equivalent to *1987-03-02-08.30.00.000000*.

Restrictions on the Use of LOCAL Date/Time Formats

When an SQL statement refers to a remote object, the following rules apply to the character string representation of dates and times.

The use of LOCAL in the CHAR function specifies the local format in effect at the remote DB2 subsystem. Otherwise, the character string representation of dates and times is limited to the standard formats. An error occurs if any local format is used in an input string. When a date or time is assigned to a host variable, the format used is determined by the option in effect at the local DB2 subsystem. If this format is LOCAL, ISO is used instead.

Basic Operations

The basic operations of SQL are assignment and comparison. Assignment operations are performed during the execution of 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 numbers and strings are not compatible. Thus, numbers and strings cannot be compared, numbers cannot be assigned to string columns or variables, and strings cannot be assigned to numeric columns or variables. Also, character and graphic strings are not compatible. However, all character strings are compatible, all graphic strings are compatible, and all numbers are compatible. 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.

Dates, times, and timestamps are not compatible. Date/time values can be assigned to character string columns and variables. A valid string representation of a date can be assigned to a date column or compared with a date. A valid string representation of a time can be assigned to a time column or compared with a time. A valid string representation of a timestamp can be assigned to a timestamp column or compared with a timestamp.

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. (See "Host Variables" on page 46 for a discussion of indicator variables.)

Numeric Assignments

The basic rule for numeric assignments is that the whole part of decimal or integer number is never truncated. If necessary, the fractional part of a decimal number is truncated.

Decimal or Integer to Floating-Point

Floating-point numbers are approximations of real numbers. Hence, when a decimal or integer number is assigned to a floating-point column or variable, the result may not be identical to the original number.

Because of the added length of double precision floating-point numbers (64 bits rather than the 32 bits of a single precision value), the approximation will be more accurate if the receiving column or variable is defined as double precision rather than single precision. Accuracy is lost if the precision of the target is less than that of the assigned value, as would be the case if a number greater than 16,777,216 were assigned to a single precision floating-point column.

Floating-Point or Decimal to Integer

When a floating-point or decimal number is assigned to an integer column or variable, the fractional part of the number is lost.

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 appended or eliminated, and, in the fractional part of the number, the necessary number of trailing zeros is appended, 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 converted to decimal, the number is first converted to a temporary decimal number of precision 6 by rounding on the seventh decimal digit. Nine zeros are then appended to the number to bring the precision to 15. Because of the rounding involved, a number less than $0.5*10^{-6}$ is reduced to 0.

When a double precision floating-point number is converted to decimal, 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, the number is rounded (using floating-point arithmetic) to a precision of 15 decimal digits. As a result, a number less than $0.5*10^{-15}$ is reduced to 0. If the decimal representation 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 example that follows shows the effect of rounding a double precision floating-point number to create the temporary decimal number:

```
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
```

To COBOL Integers

Assignment to COBOL binary integers is performed as if the NOTRUNC compiler option were specified. Thus the assigned value is not necessarily within the range of values specified by the PICTURE clause.

String Assignments

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.)

When a string is assigned to a fixed-length string column or 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 EBCDIC or double-byte blanks.

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 may or may 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. When a string is assigned to a column and the string is longer than the length attribute of that column, an error occurs. For a description of the SQLCA, see "SQL Communication Area (SQLCA)" on page 249.

If mixed data is in effect, special truncation rules apply when a string is assigned to a host variable that is not long enough to hold the shift-in character that ends the double-byte sequence. To prevent loss of the shift-in character, one additional character may be cut from the end of the string; then a shift-in character is appended before the assignment is made. In the truncated result, there is always an even number of bytes (and hence an integral number of double-byte characters) between each shift-out character and its matching shift-in character.

Date/Time Assignments

The basic rule for date/time assignments is that a DATE, TIME, or TIMESTAMP value may only be assigned to a column with a matching data type (whether DATE, TIME, or TIMESTAMP), or to a fixed- or varying-length character string variable or column for which no field procedure has been defined.

When a date/time value is assigned to a character string variable or column, conversion to a string representation is automatic. Leading zeros are not omitted from any part of the date, time, or timestamp. The required length of the target will vary depending on the format of the string representation. If the length of the target is greater than required, it will be padded on the right with blanks. If the length of the target is less than required, the result will depend on the type of date/time value involved, and on the type of target.

If the target is a column, truncation is not allowed. The length must be 10 for a date, 8 for a time, and 26 for a timestamp.

When the target is a host variable, the following rules apply:

For a DATE: If the variable is less than 10 bytes, an error will occur.

For a TIME: If the USA format is used, the length of the variable must not be less than 8; in other formats the length must not be less than 5.

If ISO or JIS formats are used, and if the length of the host variable is less than 8, the seconds part of the time is omitted from the result and assigned to the indicator variable, if provided. The SQLWARN1 field of the SQLCA is set to indicate the omission.

For a TIMESTAMP: If the host variable is less than 19 bytes, an error occurs. If the length is less than 26, but greater than or equal to nineteen bytes, trailing digits of the microseconds part of the value are omitted.

For further information on string lengths for date/time values, see "Date/Time Values" on page 28.

Numeric Comparisons

Numbers are compared algebraically; that is, with regard to sign. −2, for example, is less than +1, even though 2 is greater 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 floating-point and the other is integer or decimal, the comparison is made with a temporary copy of the other number which has been converted to double precision floating-point. Similarly, if one number is single precision floating-point and one is double precision floating-point, the comparison is made with a temporary copy of the single precision floating-point number that has been converted to double precision. An exception to these rules is this: If one number is single precision floating-point and the other is a floating point constant, the former is compared to a single precision floating-point number derived from the constant.

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

String Comparisons

The comparison of two strings is determined by the comparison of 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. Varying-length strings that differ only in the number of trailing blanks are considered equal. 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 EBCDIC collating sequence.

As just stated, two 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 may 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.

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 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.

Date/Time Comparisons

A DATE, TIME, or TIMESTAMP value may 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 farther 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 is implied.

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

```
TIMESTAMP('1985-02-23-00.00.00') > '1985-02-22-24.00.00'
```

Constants

A *constant* (sometimes 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 an integer as a signed or unsigned number of at most 10 digits that does not include a decimal point. The data type of an integer constant is large integer, and its value must be within the range of a 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 may include a sign and a decimal point; the second number may 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: `15E1` `2.E5` `2.2E-1` `+5.E+2`

Decimal Constants

A *decimal constant* specifies a decimal number as a signed or unsigned number that includes a decimal point and at most 15 digits. The precision is the total number of digits (including leading and trailing zeros); the scale is the number of digits to the right of the decimal point (including trailing zeros).

Examples: `25.5` `1000.` `-15.` `+37589.3333333333`

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 (either an apostrophe (') or a quotation mark ("), depending on the host language you are using and the character chosen by your site at the time DB2 was installed). This form of string constant specifies the character string contained between the string delimiters. The length of the character string must not be greater than 254. 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 (upper or lower case). 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'
```

Graphic String Constants

A *graphic string constant* specifies a varying-length graphic string. There are two forms of graphic string constants: one for statements embedded in PL/I, and one for all other statements, including those prepared dynamically, and those embedded in COBOL or assembler programs. In the description below of these two forms, the string of asterisks (*****) is the actual string, consisting of 0 to 124 double-byte characters, while < and > represent shift-out and shift-in characters, respectively. The character G and the string delimiter (here represented by the apostrophe (')) are required in the positions indicated.

The forms of graphic string constants are:

- In PL/I source programs: <'*****'G>
  ' is the *double-byte* string delimiter X'427D'. To use that character within the double-byte character sequence, it must be doubled. Also G is the double-byte character X'42C7'.
- In all other contexts: G'<****>' where the delimiter (') is the EBCDIC string delimiter, X'7D'.

The precompiler recognizes graphic string constants only in COBOL, PL/I, and assembler.

Alternative Syntax

A number of installation and precompiler options influence the way SQL statements can be composed. Options that apply to static statements may not apply to dynamic statements, and vice versa. These options are discussed below.

Decimal Point Representation

Decimal points in SQL statements are represented with either periods or commas. Three options control the requisite representation:

- An installation option whose value can be comma (,) or period (.). In the explanation below, this is termed "the install option."
- COMMA and PERIOD, which are mutually exclusive DB2 precompiler options for OS/VS COBOL and VS COBOL II.

These options relate as follows to individual 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 option is specified, the value of the install option at precompilation time determines the representation.
- In all programs, the value of the install option at the time a dynamic statement is prepared determines the decimal representation for that statement.
- In non-COBOL programs, the decimal representation for static SQL statements is always the period.

For remotely executed statements, the applicable option is the local one. For example, for a dynamically prepared statement, the local value of the install option, not the value at the remote subsystem, determines the decimal point representation.

If the comma is the decimal point, the following rules apply:

- A comma that is intended as a separator of numeric constants in a SELECT clause or a VALUES clause must be followed by a space.
- In any context, a comma intended as a decimal point must not be followed by a space.

Thus, to specify a decimal constant without a fractional part, the trailing comma must be followed by a nonblank character. The nonblank character may be a separator comma, as in:

```
VALUES(9999999999,, 111)
```

If `VALUES(9999999999, 111)` were specified, an error would occur because 9999999999 would be interpreted as an integer constant, and this value is not within the range of integers. This error can be avoided in all contexts by enclosing the decimal constant within parentheses, as in:

```
SELECT A FROM B WHERE A < (9999999999,)
```

String Delimiters

APOST and QUOTE are mutually exclusive DB2 precompiler options for OS/VS COBOL and VS COBOL II. Their meanings are exactly what they are for the COBOL compilers: APOST designates the apostrophe (') as the string delimiter in COBOL statements. QUOTE designates this delimiter as the quotation mark ("). Please note that neither option applies to SQL syntax, and neither should be confused with the APOSTSQL and QUOTESQL options about to be described.

APOSTSQL and QUOTESQL are mutually exclusive DB2 precompiler options for OS/VS COBOL and VS COBOL II. They determine what should be used for string delimiters and escape characters in static SQL statements. APOSTSQL designates the apostrophe (') as the string delimiter, and the quotation mark as the escape character. QUOTESQL designates the opposite. If neither of these options is specified for a COBOL precompilation, the SQL string delimiter installation option supplies a default.

For static SQL statements in non-COBOL programs, usage is fixed: the string delimiter is the apostrophe, and the escape character is the quotation mark. Usage for dynamically prepared statements is determined by the SQL string delimiter installation option. This is true for all source languages, not just COBOL.

The Character Set Option

This is an installation option that specifies whether ordinary identifiers can contain Katakana characters. There are no corresponding precompiler options. The option applies equally to static and dynamic statements. For a remotely executed statement, the local option is in effect, not the one at the remote subsystem.

The Mixed Data Option

This installation option specifies whether character strings can contain DBCS characters. A corresponding precompiler option (GRAPHIC or NOGRAPHIC) exists for every host language supported except C.

The 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 50. The installation option applies to dynamic statements, the precompiler option to static statements. An option value that applies to a statement with local references also applies to any statement with remote references. An exception is the LIKE predicate, for which the installation option at the remote subsystem applies.

The Date and Time Options

These two options affect the formatting of date/time strings. For each there is an installation option and DB2 precompiler options.

The formatting of date/time strings is described in "String Representations of Date/Time Values" on page 29. Unlike the options previously described, an option value in effect for a statement that references local objects may not be in effect for a statement that references remote objects. This is addressed under "Restrictions on the Use of LOCAL Date/Time Formats" on page 31.

Standard SQL Language

ISO-ANS SQL, described in *ANSI X3.135-1986*, differs from DB2 SQL in some of its syntax rules. The STDSQL option for the DB2 precompiler addresses some of these differences:

- STDSQL(NO) designates the regular DB2 syntax rules.
- STDSQL(86) aligns the syntax rules more closely with those of ISO-ANS SQL.

The STDSQL installation option supplies the default if neither STDSQL(NO) nor STDSQL(86) is specified for a precompilation. With STDSQL(86) in effect, the syntax rules are as follows:

Declaring Host Variables: All host variable declarations must lie between pairs of BEGIN DECLARE SECTION and END DECLARE SECTION statements:

```
BEGIN DECLARE SECTION
 one or more host variable declarations
END DECLARE SECTION
```

Separate pairs of these statements can bracket separate sets of host variable declarations.

Declaration for SQLCODE: SQLCODE cannot be part of any structure, including the one for SQLCA. It is up to the programmer, not DB2, to make the declaration, which defines SQLCODE as a full-word integer. The declaration can appear wherever variable declarations normally occur. For PL/I, an acceptable declaration could look like this:

```
DECLARE SQLCODE BIN FIXED(31);
```

SQLCODE cannot be declared within a list of variables with a common set of attributes. In a PL/I program, for example, the declaration below is invalid:

```
DECLARE (SQLCODE, ABC) BIN FIXED(31); /* INVALID DECLARATION */
```

Definitions for SQLCA: SQLCA cannot be defined, either by coding its definition manually or by using the statement INCLUDE SQLCA. The DB2 precompiler will add the appropriate definition. In this definition, a variable named SQLCADE occupies the position held by SQLCODE when STDSQL(86) is not in effect. Hence, when an SQL statement is executed, the return code is stored in SQLCADE. Precompiler-generated code then copies the value of SQLCADE into the variable SQLCODE, whose declaration was discussed above.

If the precompiler encounters an INCLUDE SQLCA statement, it ignores it and issues a warning message. But the precompiler does not recognize hand-coded definitions. Hence, a hand-coded definition produces a compile-time conflict with the precompiler-generated definition. A similar conflict arises if a definition of SQLCADE, other than the one generated by the DB2 precompiler, appears in the program.

Note that in FORTRAN programs, variables named SQLCOD and SQLCAD play the roles described above for SQLCODE and SQLCADE.

Column Functions Containing DISTINCT: DB2 SQL allows an operand of an arithmetic operator to be a column function containing the reserved word DISTINCT. ISO-ANS SQL, on the other hand, does not allow this. With SQLSTD(86) in effect, the precompiler allows this type of construct but issues a warning message whenever one is encountered.

The NOFOR Option: FOR UPDATE OF

The NOFOR precompiler option concerns the use of the FOR UPDATE OF clause when a cursor is declared for a static (embedded) query.

With NOFOR in effect, this clause is optional. When used, the clause restricts updates to the columns designated within it, and it causes the acquisition of update locks when the cursor is used to fetch a row. When not used, positioned updates can be made to any columns that the program has DB2 authority to update. With NOFOR not in effect, the clause is mandatory, and must identify every column involved in a positioned update.

NOFOR is in effect for a precompilation if one or both of the following is true:

- The NOFOR option is specified.
- STDSQL(86) is in effect.

NOFOR is otherwise not in effect. With NOFOR not in effect, the DB2 precompiler can still build DBRMs incrementally. But with NOFOR in effect, DBRMs must be built entirely in virtual storage. Hence, use of NOFOR may increase the virtual storage requirements for the DB2 precompiler. On the other hand, creating

DBRMs entirely in virtual storage may ease concurrency problems with DBRM libraries.

Special Registers

A *special register* is a storage area that is defined for a process by DB2 and is used to store information that can be referenced in SQL statements. For remotely executed statements (those that reference remote objects), the meanings of certain registers differ slightly from their meanings for locally executed statements. The six special registers are as follows:

Note: A commit or rollback operation has no effect on special register values.

USER

For locally executed statements, USER specifies the primary authorization ID of the process. For remotely executed statements, it specifies the authorization ID derived from the primary authorization ID through "translation." Translation is described in Section 5 (Volume 2) of *Administration Guide*. If necessary, the authorization ID is padded on the right with blanks so that the value of USER is always a fixed-length character string of length 8.

Example:

```
SELECT * FROM SYSIBM.SYSTABLES
  WHERE CREATOR = USER
```

CURRENT DATE

The CURRENT DATE special register specifies the current date. For a locally executed statement, this date is found as follows:

- A reading is taken from the local time-of-day clock at the time the statement is executed.
- To this reading is added the hours, minutes, and seconds specified in the TIMEZONE parameter of the local MVS system. (The TIMEZONE parameter is in SYS1.PARMLIB(CLOCKXX).) The date is then taken from the newly calculated time.

For a remotely executed statement, the calculations are the same, but they are based on the time-of-day clock and TIMEZONE parameter at the remote location.

When, in a single statement, CURRENT DATE is used more than once, or is used with CURRENT TIME or CURRENT TIMESTAMP, all their values are based on a common clock reading and TIMEZONE parameter value.

Example:

```
UPDATE DSN8220.PROJ SET PRSENDATE = CURRENT DATE
  WHERE PROJNO = 'MA2111'
```

CURRENT SQLID

The CURRENT SQLID special register specifies the SQL authorization ID of the process. If necessary, the authorization ID is padded on the right with blanks so that the value of CURRENT SQLID is always a fixed length character string of length eight.

CURRENT SQLID can be initialized by the authorization exit. If not, its initial value is the primary authorization ID of the process. CURRENT SQLID is the only special register that you can change. See "SET CURRENT SQLID" on page 238 for an explanation of how to do this.

Note: A statement that refers to remote objects cannot contain an occurrence of CURRENT SQLID.

CURRENT TIME

The CURRENT TIME special register specifies the current time. For a locally executed statement, this time is found as follows:

- A reading is taken from the local time-of-day clock at the time the statement is executed.
- To this reading is added the hours, minutes, and seconds specified in the TIMEZONE parameter of the local MVS system. (The TIMEZONE parameter is in SYS1.PARMLIB(CLOCKXX).) The time is then taken from these calculations.

For a remotely executed statement, the calculations are the same, but they are based on the time-of-day clock and TIMEZONE parameter at the remote location.

When, in a single statement, CURRENT TIME is used more than once, or is used with CURRENT DATE or CURRENT TIMESTAMP, all their values are based on a common clock reading and TIMEZONE parameter value.

CURRENT TIMESTAMP

The CURRENT TIMESTAMP special register specifies the current timestamp. For a locally executed statement, the timestamp is found as follows:

- A reading is taken from the local time-of-day clock at the time the statement is executed.
- To this reading is added the hours, minutes, and seconds specified in the TIMEZONE parameter of the local MVS system. (The TIMEZONE parameter is in SYS1.PARMLIB(CLOCKXX).) The timestamp is then taken from these calculations.

For a remotely executed statement, the calculations are the same, but they are based on the time-of-day clock and TIMEZONE parameter at the remote location.

When, in a single statement, CURRENT TIMESTAMP is used more than once, or is used with CURRENT DATE or CURRENT TIME, all their values are based on a common clock reading and TIMEZONE parameter value.

CURRENT TIMEZONE

The CURRENT TIMEZONE special register specifies, as a time duration, the value of the MVS TIMEZONE parameter of SYS1.PARMLIB(CLOCKXX) at the appropriate subsystem: the local subsystem for a locally executed SQL statement; the pertinent remote subsystem for a remotely executed statement. The data type is DECIMAL(6,0). The intended use of CURRENT TIMEZONE is to allow easy conversion of the time kept at a subsystem into Greenwich Mean Time (GMT) by subtracting CURRENT TIMEZONE from a local TIME value.

TIMEZONE must represent a time duration between -24 and 24 hours. Any values outside of this range will cause an error when CURRENT TIMEZONE is referenced. The seconds part of CURRENT TIMEZONE is always zero. Any seconds in TIMEZONE are rounded to the nearest minute.

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 under "SELECT INTO" on page 236.) 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 may be a table name, a view name, an alias name, a synonym, or a correlation name.

Whether a column name may 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, it *may* 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. They are described under:

- "Column Name Qualifiers to Avoid Ambiguity"; and
- "Column Name Qualifiers in Correlated References" on page 45.

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 designate the table from which the column comes.

Table Designators: A qualifier that designates 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 partial statement:

```
SELECT Z.CODE, Y.MYTABLE.CODE
  FROM X.MYTABLE Z, Y.MYTABLE
  WHERE ...
```

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

1. 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.
2. A table name, view name, alias, or synonym that is *not* followed by a correlation name is a table designator. Thus, Y.MYTABLE (a qualified table name) is a table designator, and qualifies the second column name after SELECT.

Avoiding undefined or ambiguous references: 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 designated 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. In this case, distinct correlation names must be used to unambiguously designate the particular instances of the table.

In the following FROM clause, for example, X and Y are defined to refer, respectively, to the first and second instances of the table EMP.

```
FROM DSN8220.EMP X, DSN8220.EMP Y
```

A situation to avoid: Do NOT write an example like this one:

```
SELECT MYTABLE.CODE
  FROM MYTABLE, MYTABLE
```

In this instance, MYTABLE.CODE is ambiguous. DB2 arbitrarily resolves such a reference wherever it occurs, and not always in the same way. It does not produce an error message. In the example, the ambiguity should be avoided by defining a correlation name for each instance of MYTABLE.

Column Name Qualifiers in Correlated References

A *subselect* is a form of a query that may be used as a component of various SQL statements. Refer to Chapter 5, "Queries" on page 83 for more information on subselects. A subselect used within a search condition of any statement is called a *subquery*.

A subquery may include search conditions of its own, and these search conditions may, in turn, include subqueries. Thus an SQL statement may 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 may 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*.

An unqualified column name is never a correlated reference. A qualified column name, Q.C, is a correlated reference if, and only if, these conditions are met:

1. Q.C is used in a search condition of a subquery.
2. Q does not designate a table used in the FROM clause of that subquery.
3. Q does designate 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. Since the same table or view can be identified at many levels, unique correlation names are recommended as table

designators. However, Q need not be a correlation name. If Q is used to designate a table at more than one level, Q.C refers to the level that most directly contains the subquery that includes Q.C.

The correlated reference Q.C identifies a value of C in a row or group of Q to which two search conditions are being applied: condition 1 in the subquery, and condition 2 at some higher level. If condition 2 is used in a WHERE clause, the subquery is evaluated for each *row* to which condition 2 is applied. If condition 2 is used in a HAVING clause, the subquery is evaluated for each *group* to which condition 2 is applied. (For another discussion of the evaluation of subqueries, see the descriptions of the WHERE and HAVING clauses under Chapter 5, "Queries" on page 83.)

For example, in the following statement, the correlated reference X.WORKDEPT (in the last line) refers to the value of WORKDEPT in table DSN8220.EMP at the level of the first FROM clause. (That clause establishes X as a correlation name for DSN8220.EMP.) The statement lists employees who make less than the average salary for their department.

```
SELECT EMPNO, LASTNAME, WORKDEPT
  FROM DSN8220.EMP X
  WHERE SALARY < (SELECT AVG(SALARY)
                  FROM DSN8220.EMP
                  WHERE WORKDEPT = X.WORKDEPT)
```

Host Variables

A *host variable* is a PL/I variable, C variable, FORTRAN variable, COBOL data item, or Assembler language storage area that is referenced 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 referenced in ways that do not apply to FORTRAN and Assembler language. This is explained under "Host Structures in PL/I, C, and COBOL" on page 48. The following applies to all host languages.

The term *host-variable*, as used in the syntax diagrams, shows a reference to a host variable. A *host-variable* in the INTO clause of a FETCH or a SELECT INTO statement identifies a host variable to which a value from a column of a row is assigned. In all other contexts a *host-variable* specifies a value to be passed to DB2 from the application program.

The general form of a *host-variable* reference is:

```
:V1 INDICATOR :V2
```

where V1 and V2 are the names of host variables. The word INDICATOR is optional. It is ***not*** an SQL reserved word. The first colon is also optional. If it is omitted, the word INDICATOR must also be omitted. Conditions under which the first colon can be omitted are discussed later in this section.

The variable V1 is the *main variable*. Depending on the operation, it either furnishes a value to DB2 or is furnished one. It could, for example, specify a comparand in a WHERE clause or a replacement for a column value. Or it could receive a column value when a row is fetched from a table.

The variable V2 is the *indicator variable*. A negative value for V2 designates a null value for the variable V1. A negative value for the indicator variable designates a null value.

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 a negative value, either to -1 if the value selected was the null value, or to -2 if the null value was returned because of numeric conversion errors or arithmetic expression errors met in the SELECT list of an outer SELECT statement. 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.

Another form of *host-variable* reference is:

```
:V1
```

If this form is used, V1 has no indicator variable and hence cannot be assigned null values. Thus this form should not be used in an INTO clause unless the corresponding column cannot contain null values.

If a null value is returned, and you have not provided an indicator variable, a negative value is returned in the SQLCODE field of the SQLCA. If your data is truncated and there is no indicator variable, no error condition results, but a warning indicator is set.

A host variable can also be referenced without specifying the leading colon. The colon is always required if the host identifier is identical to an SQL reserved word. The colon can be omitted if the host identifier is not identical to a reserved word, or if it is referenced in any of the following contexts:

- The INTO clause of a SELECT statement
- The FROM clause of a PREPARE statement
- The USING clause of an EXECUTE or OPEN statement
- The DESCRIPTOR clause of an EXECUTE, FETCH, or OPEN statement
- Following LIKE or NOT LIKE
- Following VALUES (in an INSERT statement)
- Following IN (in a SELECT statement)
- With indicator variables
- Qualified by a valid host structure.

Note: 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 reference to a column. 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 should avoid declaring host structures with a name that is the same as any possible qualifiers of a *column-name* specified in your program.

In PL/I and C, an SQL statement that references host variables must be within the scope of the declaration of those host variables. For host variables referenced in the SELECT statement of a cursor, this rule applies to the OPEN statement rather than to the DECLARE CURSOR statement.

Also note that a COBOL host variable beginning with a digit will be treated as a number if it could be so interpreted. To prevent this, always precede the variable with a colon. 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 referenced 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 designates a host structure. If S2 designates a host structure, it must be defined as a vector of small integer variables. S1 is the main structure and S2 is its indicator structure.

A host structure may be referenced in any context where a list of host variables may 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 structure is the indicator variable for the *n*th variable of the main structure.

In PL/I, for example, if V1, V2, and V3 are declared as variables within the structure S1, the statement:

```
EXEC SQL FETCH CURSOR1 INTO :S1;
```

is equivalent to:

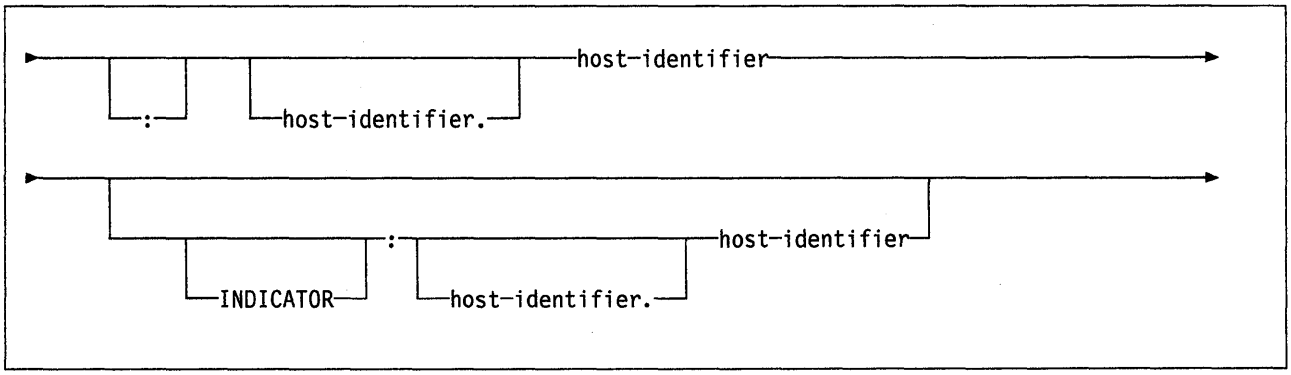
```
EXEC SQL FETCH CURSOR1 INTO :V1, :V2, :V3;
```

If the main structure has *m* more variables than the indicator structure, the last *m* variables of the main structure do not have indicator variables. If the main structure has *m* less variables than the indicator structure, the last *m* variables of the indicator structure are ignored. These rules also apply if a reference to a host structure includes an indicator variable or if a reference to a host variable includes an indicator structure. If an indicator structure or variable is not specified, no variable of the main structure has an indicator variable.

In addition to structure references, individual host variables or indicator variables in PL/I, and COBOL may be referenced by qualified names. The qualified form is a host identifier followed by a period and another host identifier. The first host identifier must designate a structure, and the second host identifier must designate a host variable within that structure.

If a host variable is referenced by a qualified name, the leading colon may always be omitted. So, you should avoid declaring host structures with a name that is the same as any possible qualifier of a column-name specified in your program.

The following diagram specifies the syntax of references to host variables and host structures. When the first colon shown in the diagram is omitted, the word INDICATOR must also be omitted.

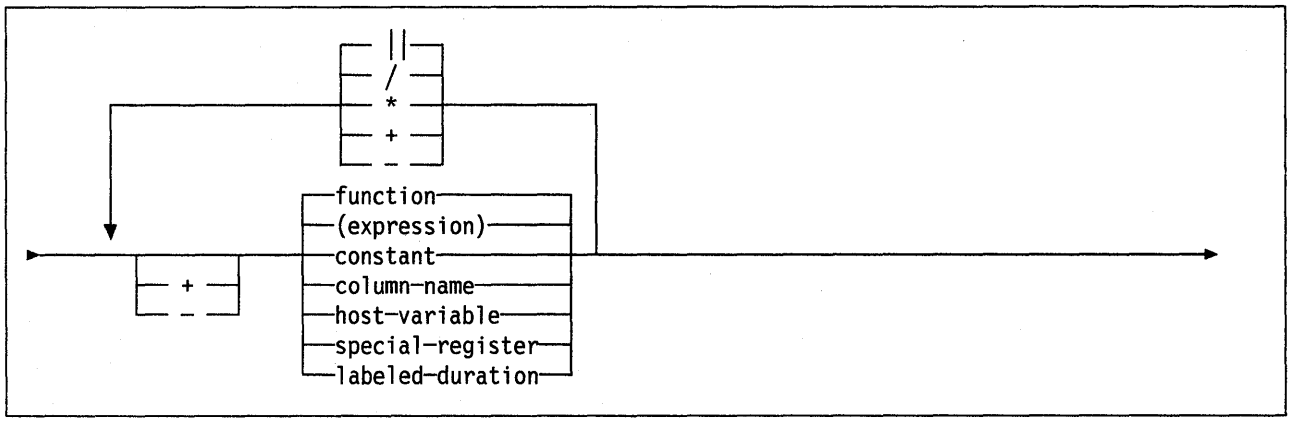


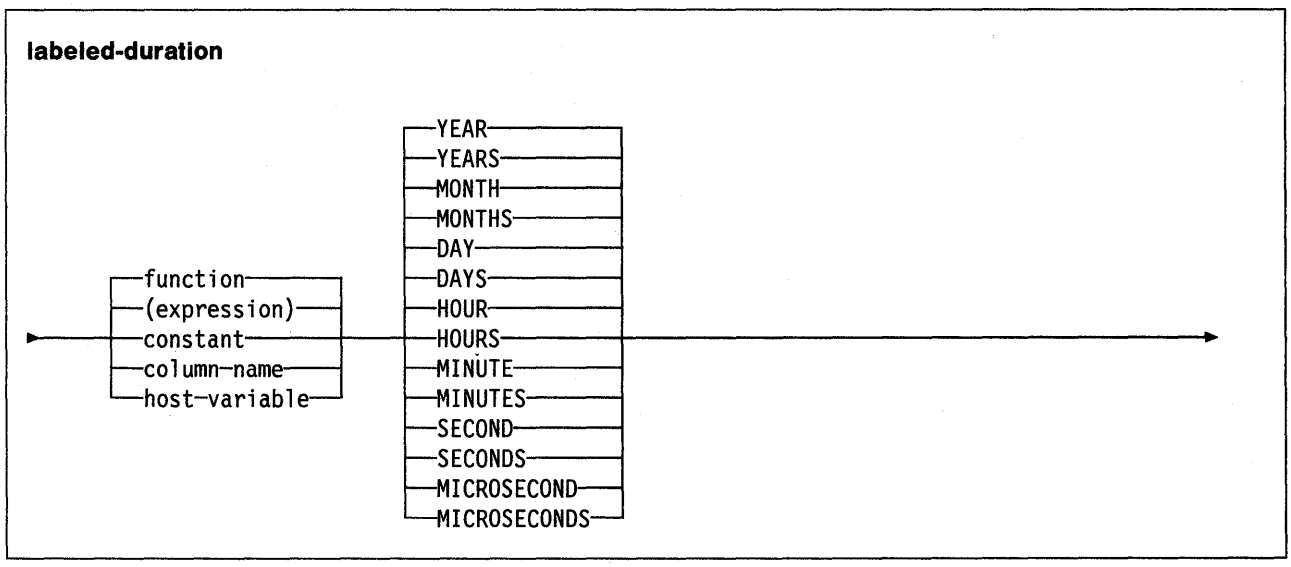
Examples: :V1 :S1.V1 S1.V1:V2 S1.V2:S2.V4

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.

Expressions

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





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

If the concatenation operator (||) is used, the result of the expression is a string. The operands of concatenation must be compatible strings. If both operands are character strings, the sum of their lengths must not exceed 32,754; if both are graphic strings, the sum of their lengths must not exceed 16,382.

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. With mixed data this result will not have redundant shift codes "at the seam." Thus, if the first operand is a string ending with a "shift-in" character (X'0F'), while the second operand is a character string beginning with a "shift-out" character (X'0E'), these two bytes are eliminated from the result.

The length of the result is the sum of the lengths of the operands, unless redundant shift codes are eliminated, in which case the length is two less than the lengths of the operands.

If both operands are fixed-length graphic strings, and the length of the result is less than 128, the result is a fixed-length graphic string. Otherwise, the result is a varying-length graphic string whose maximum length is the sum of the maximum lengths of the operands, or 16,382, whichever is less. If the maximum length is greater than 127, the result is subject to the restrictions that apply to long strings.

If both operands are fixed-length character strings, and the length of the result is less than 255 (and mixed data is not in effect), the result is a fixed-length character string. Otherwise, the result is a varying-length character string whose maximum length is the sum of the maximum lengths of the operands, or 32,764, whichever is less. If the maximum length is greater than 254, the result is subject to the restrictions that apply to long strings.

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.

Example: `FIRSTNME||' '||LASTNAME`

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 to character strings. For example, USER+2 is invalid. Multiplication and division operators must not be applied to date/time values; these 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; and if the data type of A is *small integer*, then 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.

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.

Integer and Decimal Operands

If one operand is an integer and the other is decimal, the operation is performed in decimal using a temporary copy of the integer which 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 5 digits (including leading zeros)	same as the number of digits in the constant
Constant: 5 digits or fewer	5

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 are dependent on the operation and the precision and scale of the operands. 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 result of a decimal operation must not have a precision greater than 15. The result of decimal addition, subtraction, and multiplication is derived from a temporary result which may have a precision greater than 15. If the precision of the temporary result is not greater than 15, the final result is the same as the temporary result. If the precision of the temporary result is greater than 15, the final result is derived from the temporary result by the elimination of leading zeros so the final result has a precision of 15. An error occurs if the excess digits are not zeros.

Decimal Arithmetic in SQL

The following formulas define the precision and scale of the result of decimal operations in SQL. The symbols p and s denote the precision and scale of the first operand and the symbols p' and s' denote the precision and scale of the second operand.

The precision of the result of addition and subtraction is $\min(15, \max(p\text{-}s, p'\text{-}s') + \max(s, s') + 1)$ and the scale is $\max(s, s')$.

The precision of the result of multiplication is $\min(15, p + p')$ and the scale is $\min(15, s + s')$.

The precision of the result of division is 15 and the scale is $15\text{-}p + s\text{-}s'$. The scale must not be negative. You can specify with an install option that the scale of the result must never be less than 3.

Floating-Point Operands

If either operand of an arithmetic operator is floating-point, the operation is performed in floating-point, the operands having first been converted to double precision floating-point numbers, if necessary. 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 which 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 which has been converted to double precision floating-point. The result of a floating-point operation must be within the range of floating-point numbers.

Date/Time Operands

Date/time values can be incremented, decremented, and subtracted. The discussions that follow clarify the process by which these operations are carried out, and introduce the concept of durations.

Durations

A *duration* is a number representing an interval of time. The number may be a constant, a column name, a host variable, a function, or an expression. There are three types of durations:

Labeled Durations

A *labeled duration* represents a specific unit of time as expressed by a number followed by one of the seven duration keywords: YEARS, MONTHS, DAYS, HOURS, MINUTES, SECONDS, and MICROSECONDS.[2] 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 so the other operand is a value of data type 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 several 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 - BRTHDATE, is a date duration.

Time Duration

A *time duration* represents several 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.

Date/Time Arithmetic in SQL

The only arithmetic operations that can be performed on date/time values are addition and subtraction. If a date/time value is the operand of addition, the other operand must be a duration. The specific rules governing the use of the addition operator with date/time 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.

---

[2] Note that the singular form of these keywords is also acceptable: YEAR, MONTH, DAY, HOUR, MINUTE, SECOND, and MICROSECOND.

- 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.

The rules for the use of the subtraction operator on date/time values are not the same as those for addition because a date/time value cannot be subtracted from a duration, and because the operation of subtracting two date/time values is not the same as the operation of subtracting a duration from a date/time value. The specific rules governing the use of the subtraction operator with date/time 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 duration.
- The second operand must not be a timestamp.
- Neither operand of the subtraction operator can be a parameter marker.

Date Arithmetic

Dates can be subtracted, incremented, or decremented. The discussions that follow will clarify how these operations are carried out.

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.

- If DAY(DATE2) <= DAY(DATE1)
  then DAY(RESULT) = DAY(DATE1) - DAY(DATE2).
- If DAY(DATE2) > DAY(DATE1)
  then DAY(RESULT) = N + DAY(DATE1) - DAY(DATE2)
  where N = the last day of MONTH(DATE2).
  MONTH(DATE2) is then incremented by 1.
- If MONTH(DATE2) <= MONTH(DATE1)
  then MONTH(RESULT) = MONTH(DATE1) - MONTH(DATE2).
- If MONTH(DATE2) > MONTH(DATE1)
  then MONTH(RESULT) = 12 + MONTH(DATE1) - MONTH(DATE2).
  YEAR(DATE2) is then 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.

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', indicating 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, may also be added to and subtracted from dates. The result is a date that has been incremented or decremented by the specified number of years, months, and days, in this 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
```

If an end-of-month adjustment is performed to correct an invalid date, the SQLWARN6 field of the SQLCA is set to 'W'.

When adding durations to dates, it is assumed that adding one month to a given date gives the same date one month later *unless* that date does not exist in the later month. In that case, the date is set to that of the last day of the later month. For example, January 28 plus one month gives February 28; and one month added to January 29, 30, or 31 results in either February 28 or, for a leap year, February 29. Note that 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.

Time Arithmetic

Times can be subtracted, incremented, or decremented. The discussions that follow will clarify how these operations are carried out.

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.

- 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).
  MINUTE(DATE2) is then 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).
  HOUR(TIME2) is then incremented by 1.
- HOUR(RESULT) = HOUR(TIME1) - HOUR(TIME2).

For example, the result of TIME('11:02:26') - '00:32:56' is 2930 (a duration of 0 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. 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 will affect the seconds portion of the time and may affect the minutes and hours.

Time durations, whether positive or negative, may also be added to and subtracted from times. The result is a time that has been incremented or decremented by the specified number of hours, minutes, and seconds, in that order. Thus, TIME1 + X, where X is a positive DECIMAL(6,0) number, is equivalent to the expression

```
TIME1 + HOUR(X) HOURS + MINUTE(X) MINUTES + SECONDS(X) SECONDS
```

Timestamp Arithmetic

Timestamps can be incremented or decremented. 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.

Precedence of Operations

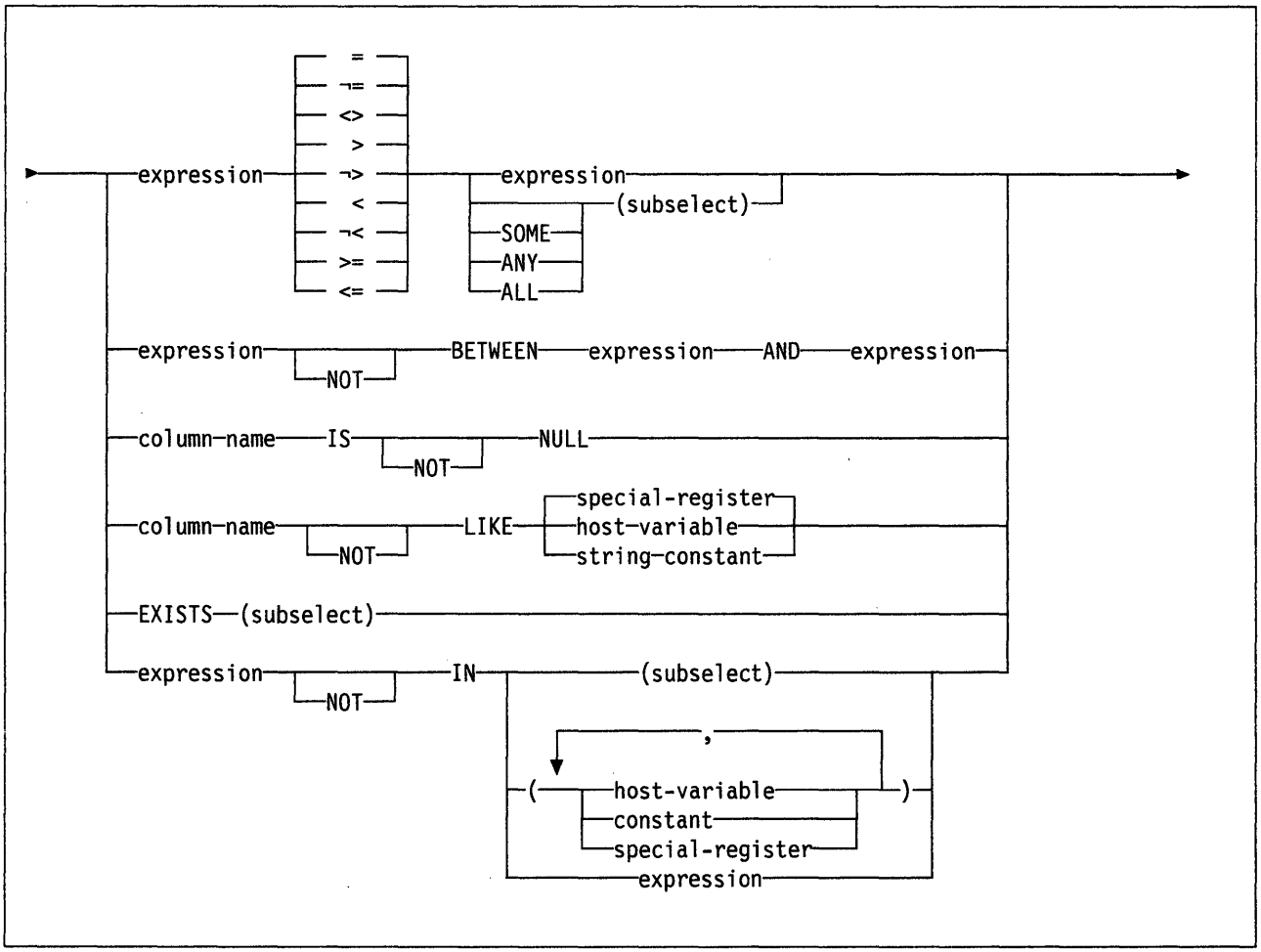
Expressions within parentheses are evaluated first. When the order of evaluation is not specified by parentheses, prefix operators are applied before multiplication and division, and multiplication and division are applied before addition and subtraction. Operators at the same precedence level are applied from left to right.

Example: `1.10 * (SALARY + BONUS)`

Predicates

A *predicate* specifies a condition that is "true," "false," or "unknown" about a given row or group.

The general form of a predicate is as follows:



All values specified in a predicate must be compatible. A long string column must not be referenced (except in a LIKE predicate), and the value of a host variable must not be a string longer than 254 bytes. The value of a host variable must not be null (that is, the variable may not have a negative indicator variable).

A view column referenced in a predicate must not be derived from a column function unless the predicate is part of a statement that meets certain special criteria. These criteria are discussed under "Use of Views: Special Criteria" on page 89.

Except for EXISTS, a subselect in a predicate must specify a single column.

Basic Predicate

A *basic predicate* compares two values. The format of a basic predicate is:

- An expression followed by a comparison operator and another expression, or
- An expression followed by a comparison operator and a subselect (without SOME, ANY or ALL). A subselect in a basic predicate must not return more than one value.

If the value of either operand is null or the subselect returns no value, the result of the predicate is unknown. Otherwise the result is either true or false.

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 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¬< *y*	*x* is not less than *y*
x¬> *y*	*x* is not greater than *y*

Examples:

```
EMPNO = '528671'
SALARY < 20000
PRSTAFF<>:VAR1
SALARY > (SELECT AVG(SALARY) FROM DSN8220.EMP)
```

Quantified Predicate

A *quantified predicate* compares a value with a collection of values. If the value of the first operand is null, the result is unknown.

A quantified predicate has the same form as a basic predicate except that the second operand is a subselect preceded by SOME, ANY, or ALL. The subselect may return any number of values, whether null or not null.

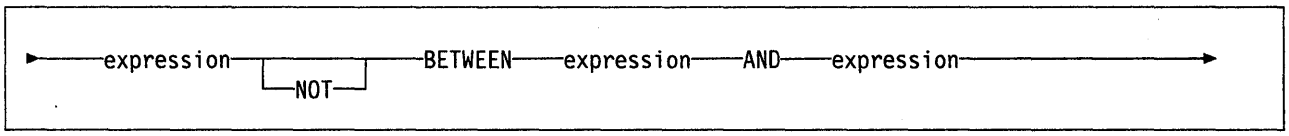
When ALL is specified, the result of the predicate is true if the subselect returns no values or if the specified relationship is true for every value returned by the subselect. The result is false if the specified relationship is false for at least one value returned by the subselect. The result is unknown if all values returned by the subselect are null or if the specified relationship is true for all non-null values and the subselect returns at least one 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. The result is false if the subselect returns no values or if the specified relationship is false for every value returned by the subselect. The result is unknown if all values returned by the subselect are null or if the specified relationship is false for all non-null values and the subselect returns at least one null value.

Example: `SALARY >= ALL(SELECT SALARY FROM DSN8220.EMP)`

BETWEEN Predicate

The BETWEEN predicate compares a value with a range of values. The format of a BETWEEN predicate is as follows:



The BETWEEN predicate:

```
value1 BETWEEN value2 AND value3
```

is equivalent to the search condition:

```
value1 >= value2 AND value1 <= value3
```

The BETWEEN predicate:

```
value1 NOT BETWEEN value2 AND value3
```

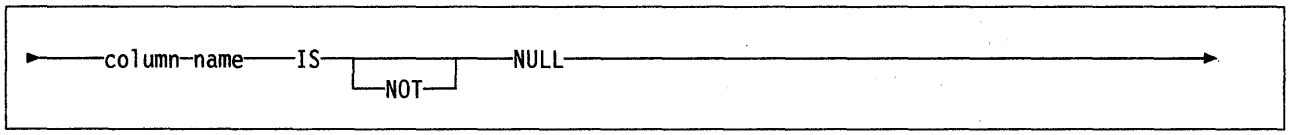
is equivalent to the search condition:

```
NOT(value1 BETWEEN value2 AND value3); that is,
value1 < value2 or value1 > value3.
```

Example: `SALARY BETWEEN 20000 AND 40000`

NULL Predicate

The NULL predicate tests for null values. The format of the NULL predicate is as follows:



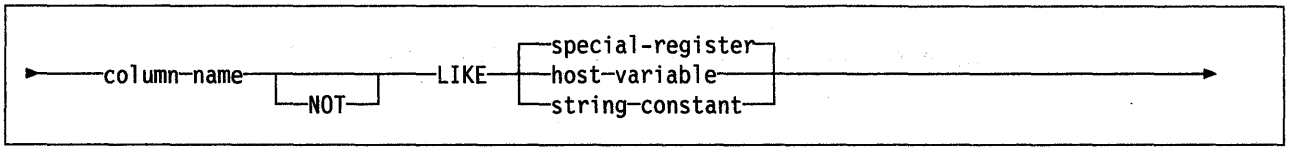
The result of a NULL predicate cannot be unknown. If the value of the column is null, the result is true. If the value is not null, the result is false. If NOT is specified, the result is reversed.

The *column-name* must not identify a view column derived from an expression, function, or constant, unless the predicate is part of a statement that meets certain special criteria. These criteria are discussed under "Use of Views: Special Criteria" on page 89.

Example: `PHONENO IS NULL`

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 have special meanings. The format of the LIKE predicate is as follows:



The *column-name* must identify a string column.

It must not identify a view column derived from an expression, function, or constant, unless the predicate is part of a statement that meets certain special criteria. These criteria are discussed under "Use of Views: Special Criteria" on page 89.

If a host variable is specified, it must identify a variable (not a structure) that is described in the program under the rules for declaring string host variables; it cannot have an indicator variable. For more on the use of host variables with specific programming languages, see Section 3 of *Application Programming and SQL Guide*.

The column may be a long string column, and may contain either character or graphic data; the host variable, special register, or string constant must contain data of the same type. If the column contains character data, the terms 'character', 'percent sign', and 'underscore' in the following discussion refer to single-byte characters; if the column contains graphic data, those terms refer to double-byte characters. The following description is intended for those who require a rigorous definition. The description uses *x* to denote a value of the column and *y* to denote the string specified by the second operand.

The string *y* is interpreted as a sequence of the minimum number of substring specifiers so each character of *y* is part of exactly one substring specifier. A substring specifier is an underscore, a percent sign, or any nonempty sequence of characters other than an underscore or a percent sign.

The result of the predicate is unknown if *x* is the null value. Otherwise, the result is either true or false. The result is true if *x* and *y* are both empty strings or 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.

The predicate x NOT LIKE y is equivalent to the search condition NOT(x LIKE y).

With Mixed Data: If the column identified by *column-name* allows mixed data, the column may contain double-byte characters, as may the host variable or string constant. In that case the special characters in *y* 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.

With a Field Procedure

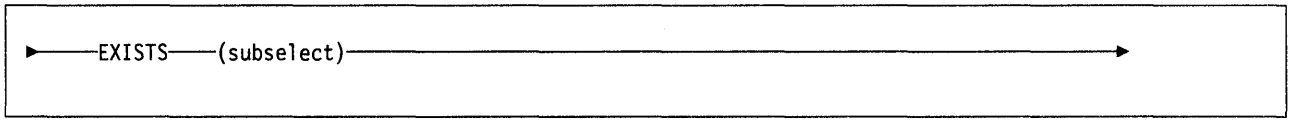
If the column has a field procedure, the procedure is invoked to decode the values of the column, and the comparisons are made with the decoded values.

Examples: `NAME LIKE '%SMITH%'` `STATUS LIKE 'N_'`

The first example is true if 'SMITH' appears anywhere within NAME. The second example is true if the value of STATUS has a length of two and the first character is 'N'.

EXISTS Predicate

The EXISTS predicate tests for the existence of certain rows. The format of the EXISTS predicate is:

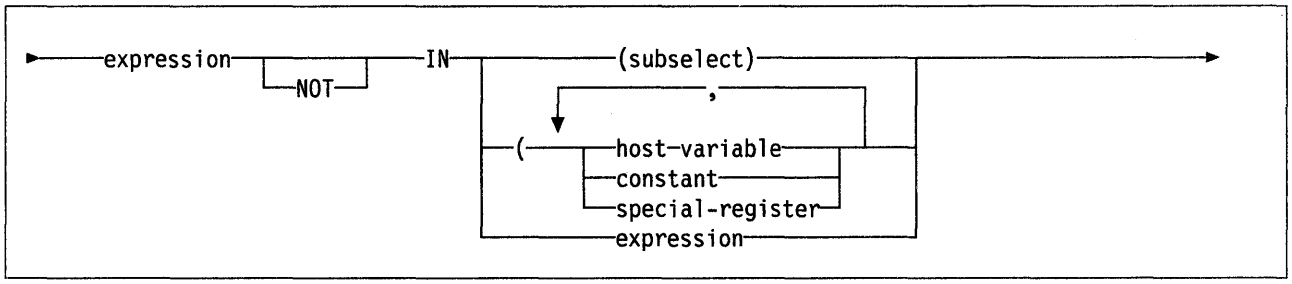


The subselect does not return values, and the result of the predicate cannot be unknown. The result is true only if the number of rows specified by the subselect is not zero.

Example: `EXISTS (SELECT * FROM TEMPL WHERE SALARY < 10000)`

IN Predicate

The IN predicate compares a value with a collection of values. The format of the IN predicate is as follows:



Each host variable specified must identify a structure or variable that is described in the program under the rules for declaring host structures and variables. An indicator variable must not be specified.

An IN predicate of the form:

```
expression IN expression
```

is equivalent to a basic predicate of the form:

```
expression = expression
```

An IN predicate of the form:

```
expression IN (subselect)
```

is equivalent to a quantified predicate of the form:

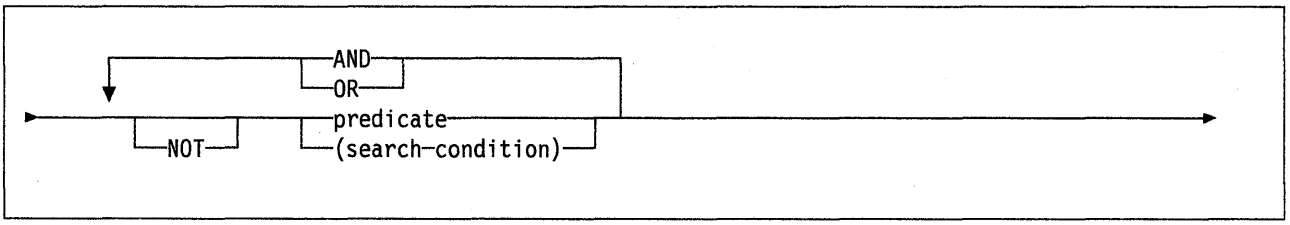
```
expression = ANY (subselect)
```

In the other form of the IN predicate, the second operand is a collection of one or more values specified by any combination of host variables, constants, or special register. This form of the IN predicate is equivalent to the form specified above, except that the second operand consists of the specified values rather than the values returned by a subselect.

Example: `DEPTNO IN ('D01', 'B01', 'C01')`

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 form of a search condition is as follows:



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.

Examples: `SALARY > 2000` `NAME LIKE :VAR4` `AVG(SALARY) < 30000`

AND and OR are defined in Table 3, in which P and Q are any predicates:

Table 3. Truth Tables 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: `MAJPROJ = 'MA2100' AND (DEPTNO = 'D11' OR DEPTNO = 'B03')`

Chapter 4. Functions

A *function* is an operation denoted by a function name followed by a pair of parentheses enclosing the specification of one or more operands. 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 the application of the function to the result of the expression.

Functions are classified as *scalar functions* or *column functions*. The argument of a column function is a collection of values. An argument of a scalar function is a single value. If multiple arguments are allowed, each argument is a single value.

In the syntax of SQL, the only use of the term "function" is 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 below and in Chapter 5, "Queries" on page 83.

Column Functions

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

The argument of a column function is a collection of values derived from one or more columns. The scope of the collection is a group or an intermediate result table as explained in Chapter 5, "Queries" on page 83. For example, the result of the following SELECT statement is the number of distinct values of JOBCODE for employees in department D01:

```
SELECT COUNT(DISTINCT JOBCODE)
  FROM DSN8220.EMP
  WHERE WORKDEPT = 'D01'
```

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. The operation, which is the elimination of duplicate values, is not performed if ALL is specified, or if neither ALL nor DISTINCT are specified.

The DISTINCT operation can only be applied to values of a column. Thus, *column-name* must not identify a view column derived from a constant or expression unless the function is part of a statement that meets certain special criteria. These are discussed in "Use of Views: Special Criteria" on page 89. If DISTINCT is omitted, the values of the arguments are specified by an expression. This expression must not include a column function, and must include at least one column name, a requirement that is not satisfied by a reference to a view column derived from a constant or expression without a column name, unless the function is part of a statement that meets the same special criteria mentioned above. If a *column-name* is a correlated reference (which is allowed in a subquery of a HAVING clause) the expression must not include operators.

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

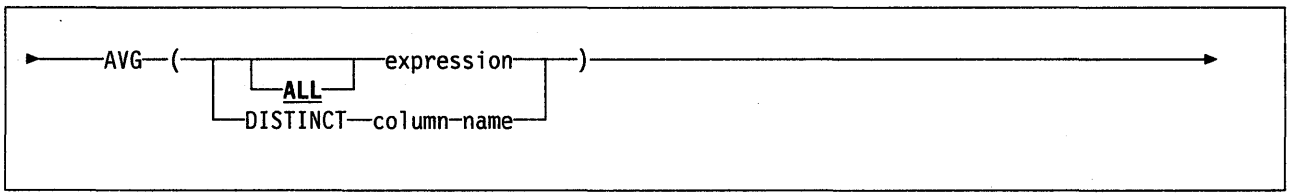
arithmetic expression, and any evaluation of the expression causes an arithmetic exception (such as division by zero).

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 alphabetical order, is a definition of each of the column functions.

AVG

The AVG function returns the average of a collection of numbers. The form of the function is:



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. If the data type of the argument values is decimal with precision *p* and scale *s*, the precision of the result is 15 and the scale is 15-*p*+*s*. The result can be null.

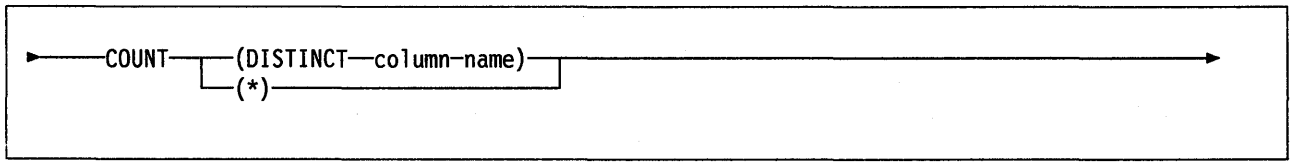
The function is applied to the collection of values derived from the argument values by the elimination of null values. If DISTINCT is specified, duplicate values are also eliminated.

If this collection of values is empty, the result of the function is the null value. Otherwise, the result is the average value of the collection. If the type of the result is INTEGER, the fractional part of the average is lost.

Example: `AVG(SALARY)`

COUNT

The COUNT function returns the number of rows or values in a collection of rows or values. The form of the function is:



The *column-name* must not identify a long string column. The result of the function must be within the range of large integers and cannot be null.

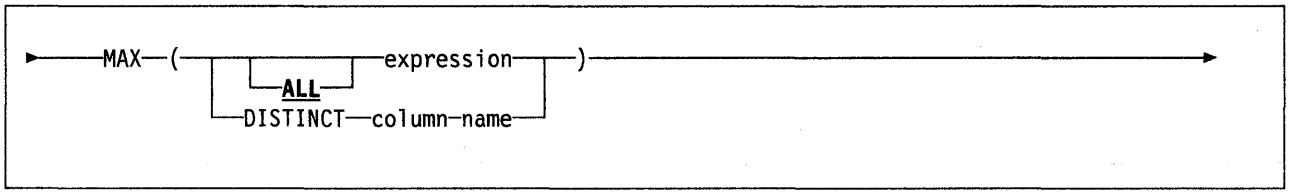
The argument of COUNT(*) is a collection of rows. The result is the number of rows in the collection.

The argument of COUNT(DISTINCT *column-name*) is a collection of values. The function is applied to the collection of values derived from the argument values by the elimination of null and duplicate values. The result is the number of values in the collection.

Example: `COUNT(DISTINCT JOBCODE)`

MAX

The MAX function returns the maximum value in a collection of values. The form of the function is:



The argument values can be any values other than character strings whose maximum length is greater than 254, or graphic strings whose maximum length is greater than 127.

The data type and length attribute of the result are the same as the data type and length attribute of the argument values. The result can be null.

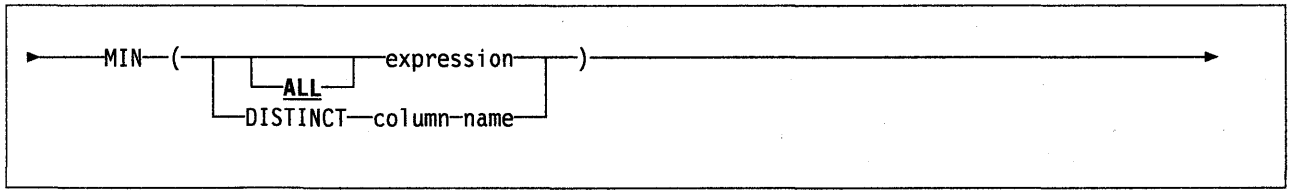
The function is applied to the collection of values derived from the argument values by the elimination of null values. If DISTINCT is specified, duplicate values are also eliminated. If this collection is empty, the result of the function is the null value. Otherwise, the result is the maximum value in the collection.

The specification of DISTINCT has no effect on the result and therefore is not recommended.

Example: `MAX(SALARY)`

MIN

The MIN function returns the minimum value in a collection of values. The form of the function is:



The argument values can be any values other than character strings whose maximum length is greater than 254, or graphic strings whose maximum length is greater than 127.

The data type and length attribute of the result are the same as the data type and length attribute of the argument values. The result can be null.

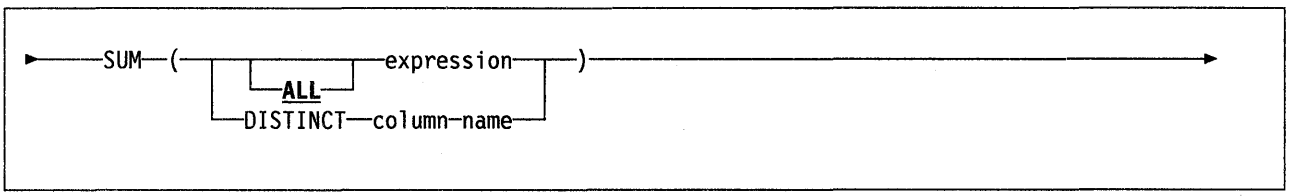
The function is applied to the collection of values derived from the argument values by the elimination of null values. If DISTINCT is specified, duplicate values are also eliminated. If this collection is empty, the result of the function is the null value. Otherwise, the result is the minimum value in the collection.

The specification of DISTINCT has no effect on the result and therefore is not recommended.

Example: `MIN(SALARY)`

SUM

The SUM function returns the sum of a collection of numbers. The form of the function is:



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 double precision floating-point if the argument values are single precision floating-point. If the data type of the argument values is decimal, the precision of the result is 15 and the scale is the same as the scale of the argument values. The result can be null.

The function is applied to the collection of values derived from the argument values by the elimination of null values. If DISTINCT is specified, duplicate values are also eliminated. If this collection is empty, the result of the function is the null value. Otherwise, the result is the sum of the values in the collection.

Example: `SUM(SALARY)`

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. For example, 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.

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 collection of

values. For example, the result of the following SELECT statement has as many rows as there are employees in department D01:

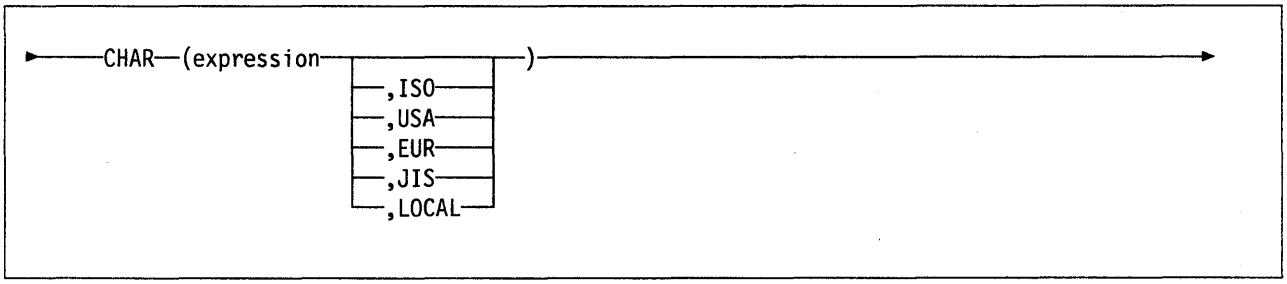
```
SELECT EMPNO, LASTNAME, YEAR(CURRENT DATE - BRTHDATE)
  FROM DSN8220.EMP
  WHERE WORKDEPT = 'D01'
```

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.

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

CHAR

The CHAR function returns a string representation of a date/time value. The form of the function is:



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

The result of the function is a fixed-length character 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. The other rules depend on the data type of the first argument.

> **Note:** When LOCAL is used with CHAR in a statement that refers to remote objects, the formatting is controlled by the option in effect at the remote subsystem, not the local subsystem.

If the first argument is a date:

Omission of the second argument is an implicit specification of the string format specified either by the DATE precompiler option, or by the DATE FORMAT install option. If LOCAL is implicitly or explicitly specified, a date installation exit must be installed.

The result is the character string representation of the date in the format specified by the second argument. If LOCAL is specified, the length of the result is the length specified by the LOCAL DATE install option. Otherwise, the length of the result is 10.

If the first argument is a time:

Omission of the second argument is an implicit specification of the string format specified by the TIME precompiler option, if provided, or, if not, by TIME FORMAT install option. If LOCAL is implicitly or explicitly specified, a time installation exit must be installed.

The result is the character string representation of the time in the format specified by the second argument. If LOCAL is specified, the

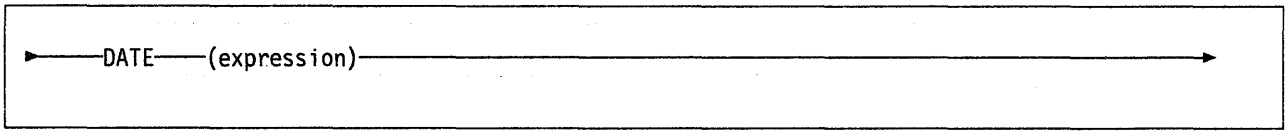
length of the result is the length specified by the LOCAL TIME install option. Otherwise, the length of the result is 8.

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

Example: `CHAR(HIREDATE,USA)`

DATE

The DATE function returns a date from a value. The form of the function is:



The argument must be a timestamp, a date, a positive number less than 3,652,059, a valid string representation of a date, or a character string of length 7.

If the argument is a character string of length 7, it must represent a valid date in the form *yyyynnn*, 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.

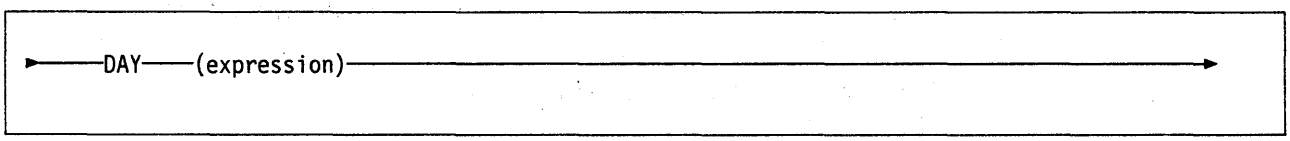
If the argument is a number:
The result is the date that is *n*-1 days after January 1, 0001, where *n* is the number that would occur if the INTEGER function were applied to the argument.

If the argument is a character string:
The result is the date represented by the character string.

Example: `DATE(STRTDATE)`

DAY

The DAY function returns the day part of a value. The form of the function is:



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

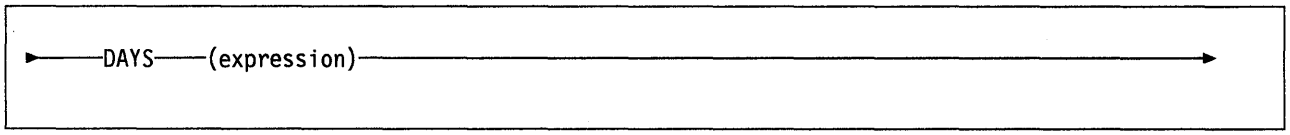
If the argument is a date or a timestamp, the result is the day part of the value, which is an integer between 1 and 31.

If the argument is a date duration, 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: `DAY(HIREDATE) = 1`

DAYS

The DAYS function returns an integer representation of a date. The form of the function is:



The argument must be a date, a timestamp, or a valid string representation of a date.

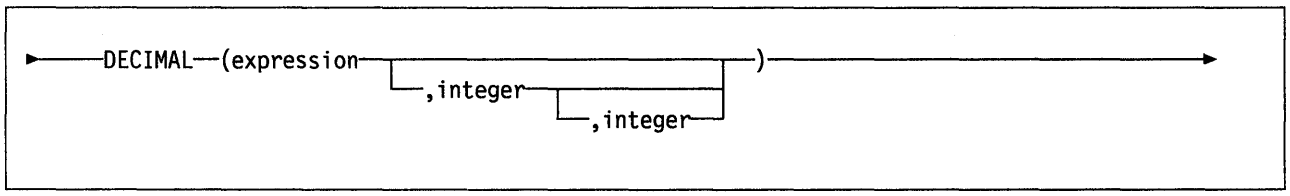
The result of the function is a large integer. If the argument can be null, the result can be null; if the argument is null, the result is the null value.

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: `DAYS(CURRDATE) - DAYS(HIREDATE)`

DECIMAL

The DECIMAL function returns a decimal representation of a numeric value. The form of the function is:



The first argument must be a number. The second argument, if specified, must be in the range of 1 to 15. The third argument, if specified, must be in the range of 0 to *p*, where p is the second argument. Omission of the third argument is an implicit specification of zero.

The default for the second argument depends on the data type of the first argument:

- 15 for floating-point and decimal
- 11 for large integer
- 5 for small integer.

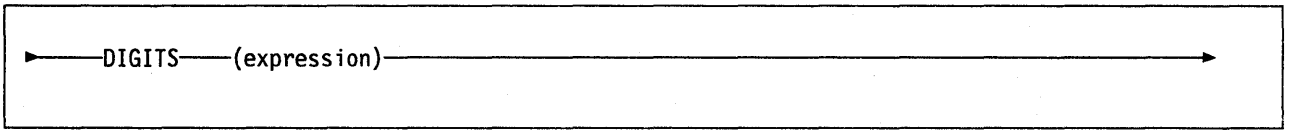
The result of the function is a decimal number with precision of *p* and scale of *s*, where *p* and *s* are the second and third arguments. If the first argument can be null, the result can be null; if the first argument is null, the result is the null value.

The result is the same number that would occur if the first argument were assigned to a decimal column or variable with a precision of *p* and a scale of *s*. An error occurs if the number of significant decimal digits required to represent the whole part of the number is greater than *p-s*.

Example: `DECIMAL(AVG(SALARY),8,2)`

DIGITS

The DIGITS function returns a character string representation of a number. The form of the function is:



The argument must be an integer or a decimal number.

The result of the function is a fixed-length character string. If the argument can be null, the result can be null; if the argument is null, the result is the null value.

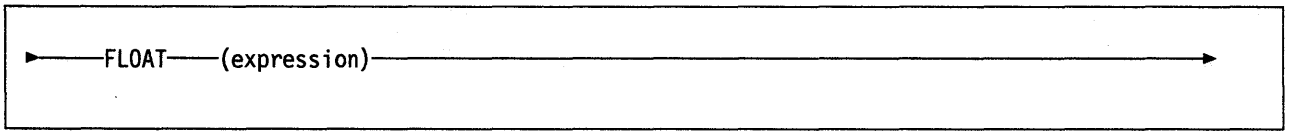
The result is a string of digits that represents the absolute value of the argument without regard to its scale. Thus, the result does not include a sign or a decimal point. The result includes any necessary leading zeros so that the length of the string is:

- 5 if the argument is a small integer
- 10 if the argument is a large integer
- *p* if the argument is a decimal number with a precision of *p*.

Example: Suppose that COLUMNX has the data type DECIMAL(6,2). Then, if COLUMNX has the value -6.28, `DIGITS(COLUMNX) = '000628'`

FLOAT

The FLOAT function returns a floating-point representation of a number. The form of the function is:



The argument must be a number.

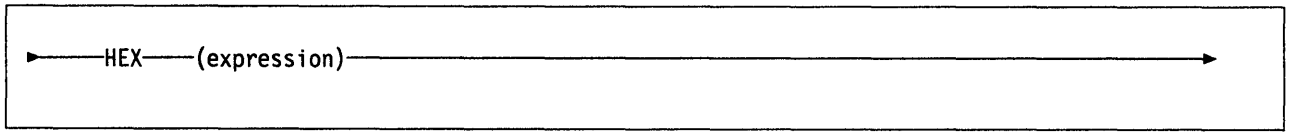
The result of the function is a double precision floating-point number. If the argument can be null, the result can be null; if the argument is null, the result is the null value.

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

Example: `FLOAT(ACSTAFF)/2`

HEX

The HEX function returns a hexadecimal representation of a value. The form of the function is:



The argument can be any value other than long strings.

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 date/time value, the result is the hexadecimal representation of the internal form of the argument.

If the argument is not a graphic string, the length of the result is twice the defined (maximum) length of the argument. If the argument is a graphic string, the length of the result is four times the defined 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.

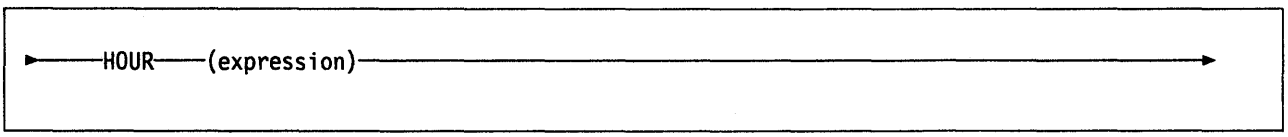
- If the argument is not a varying-length string, the maximum length of the result string is the same as the length of the result.
- If the argument is a varying-length character string, the maximum length of the result string is twice the maximum length of the argument.
- If the argument is a varying-length 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.

Example: `HEX(SYSIBM.SYSCOPY.START_RBA)`

HOUR

The HOUR function returns the hour part of a value. The form of the function is:



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

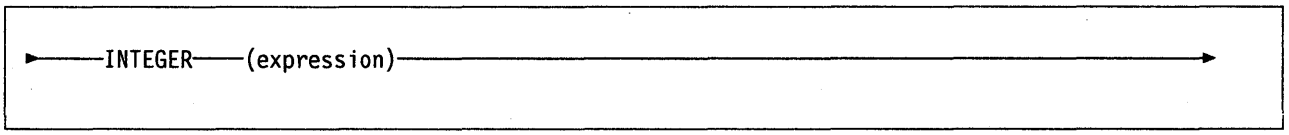
If the argument is a time or a timestamp, the result is the hour part of the value, which is an integer between 0 and 24.

If the argument is a time duration, 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: `HOUR(ARRIVAL) BETWEEN 12 AND 24`

INTEGER

The INTEGER function returns an integer representation of a number. The form of the function is:



The argument must be a number.

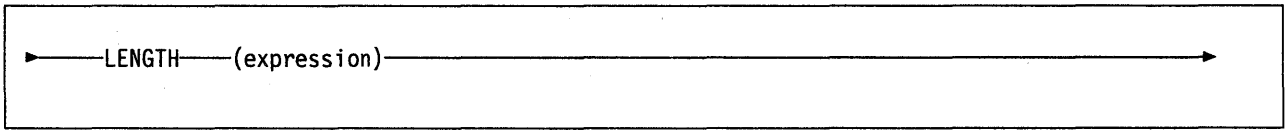
The result of the function is a large integer. If the argument can be null, the result can be null; if the argument is null, the result is the null value.

The result is the same number that would occur if the argument were assigned to a large integer column or variable. If the whole part of the argument is not within the range of integers, an error occurs.

Example: `INTEGER(SUM(EMPTIME)+.5)`

LENGTH

The LENGTH function returns the length of a value. The form of the function is:



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 strings includes blanks. 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:

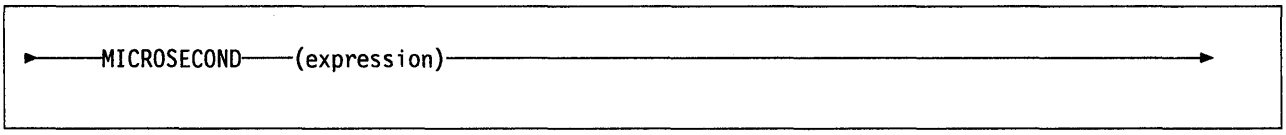
- the length of the string for character strings
- 2 for small integer
- 4 for large integer
- 4 for single precision floating-point
- 8 for double precision floating-point
- INTEGER(*p*/2) + 1 for decimal numbers with precision *p*.
- 4 for date

- 3 for time
- 10 for timestamp.

Example: `SUBSTR(STRING,LENGTH(STRING)-:N)`

MICROSECOND

The MICROSECOND function returns the microsecond part of a value. The form of the function is:



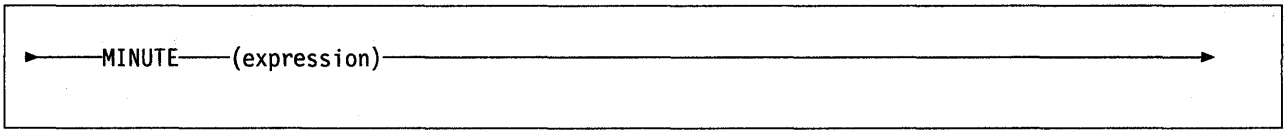
The argument must be a timestamp.

The result of the function is a large integer representing the microsecond part of the timestamp, which is an integer between 0 and 999999. If the argument can be null, the result can be null; if the argument is null, the result is the null value.

Example: `MICROSECOND(TS1) <> 0 AND SECOND(TS1) = SECOND(TS2)`

MINUTE

The MINUTE function returns the minute part of a value. The form of the function is:



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

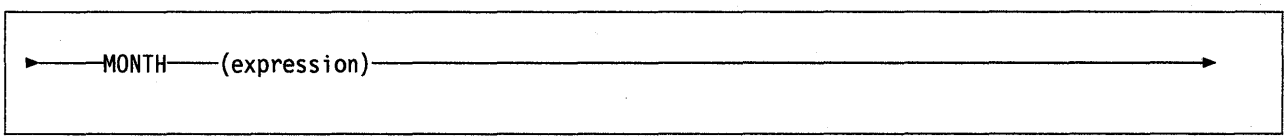
If the argument is a time or a timestamp, the result is the minute part of the value, which is an integer between 0 and 59.

If the argument is a time duration, 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: `MINUTE(ARRIVAL) = 0`

MONTH

The MONTH function returns the month part of a value. The form of the function is:



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

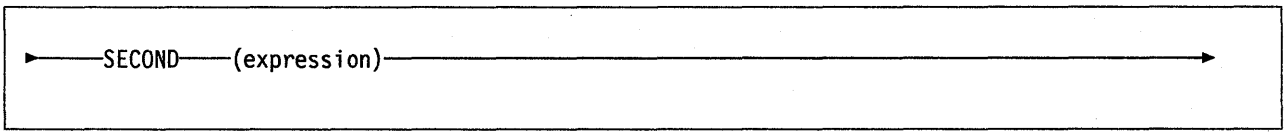
If the argument is a date or a timestamp, the result is the month part of the value, which is an integer between 1 and 12.

If the argument is a date duration, 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: `MONTH(HIREDATE) < 4`

SECOND

The SECOND function returns the seconds part of a value. The form of the function is:



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

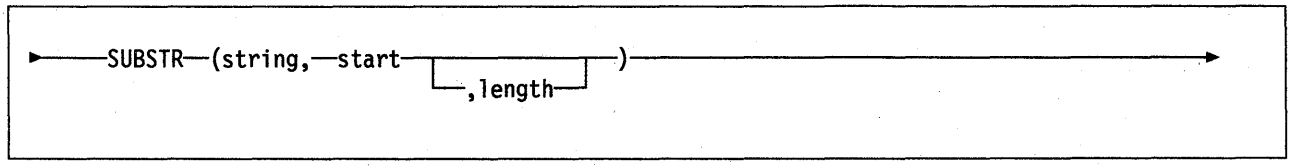
If the argument is a time or timestamp, the result is the seconds part of the value, which is an integer between 0 and 59.

If the argument is a time duration, 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: `SECOND(TS1) = SECOND(TS2) AND MICROSECOND(TS1) <> 0`

SUBSTR

The SUBSTR function returns a substring of a string. The form of the function is:



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 does not recognize mixed data, so if *string* contains mixed data, the result may not be a well-formed mixed data string.

start

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.)

length

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 the length attribute of *string* - *start* + 1. It must not, however, be the integer constant 0.

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. The default for *length* is the number of characters from the character specified by the *start* to the last character of *string*. However, if *string* is a varying-length string with a length less than *start*, the default is zero and the result is the empty string.

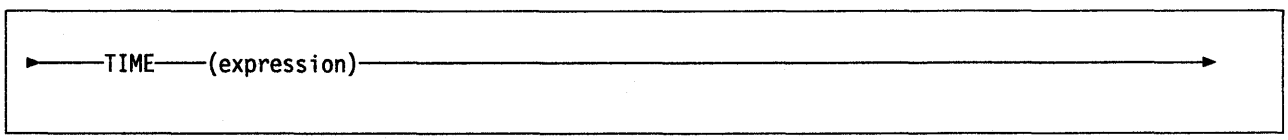
If *length* is explicitly specified by an integer constant less than 255, the result is a fixed-length string. If *length* is not explicitly specified, but *string* is a fixed-length string and *start* is an integer constant, the result is a fixed-length string. In all other cases, the result is a varying-length 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 254. These restrictions also apply if it is a graphic string whose maximum length exceeds 127.

If *string* is a fixed-length string, omission of *length* is an implicit specification of LENGTH(*string*) - *start* + 1. 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 any argument of the SUBSTR function can be null, the result can be null. If any argument is null, the result is the null value.

Example: `SUBSTR(FIRSTNAME,1,1)`

TIME

The TIME function obtains a time from a value. The form of the function is:



The argument must be a timestamp, a time, or a valid string representation of a time.

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.

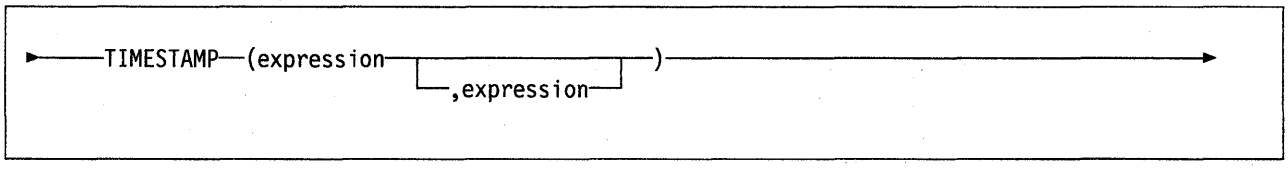
If the argument is a time:
The result is that time.

If the argument is a character string:
The result is the time represented by the character string.

Example: `TIME(TIMESTMP) > '5:00'`

TIMESTAMP

The TIMESTAMP function obtains a timestamp from a value or a pair of values. The form of the function is:



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.

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.

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.

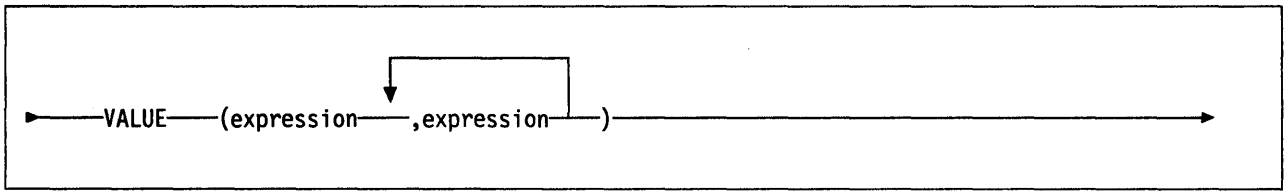
If both arguments are specified, the result is a timestamp with the date specified by the first argument and the time specified by the second argument. The microsecond part of the timestamp is zero.

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 Principles of Operation*.

Example: TIMESTAMP(DATEFRNK,TIMEFRNK)

VALUE

The VALUE function substitutes a value for the null value. The form of the function is:



The data types of the arguments must be compatible. Character strings are not converted to date/time values. Therefore, if any argument is a date, all arguments must be dates; if any argument is a time, all arguments must be times; if any argument is a timestamp, all arguments must be timestamps; and if any argument is a character string, all arguments must be character strings.

The arguments are evaluated in the order in which they are specified, and the result of the function is equal to the first argument that is not null. The result can be null only if all arguments can be null; the result is the null value only if all arguments are null.

The result is defined as 'equal to' an argument because that argument is converted or extended, if necessary, to conform to the data type of the function. The data type of the result is derived from the data types of the specified arguments as follows:

Strings:

If any argument is a varying-length string, the result is a varying length string whose maximum length is equal to the longest string that can result from the application of the function. The result is subject to the restrictions that apply to long strings if the arguments are character strings, and the maximum length of the result is greater than 254, or if the arguments are graphic strings and the maximum length of the result is greater than 127.

If all arguments are fixed-length strings, the result is a fixed length string whose length is equal to the longest string that can result from the application of the function.

Date/time Values:

If the arguments are dates, the result is a date. If the arguments are times, the result is a time. If the arguments are timestamps, the result is a timestamp.

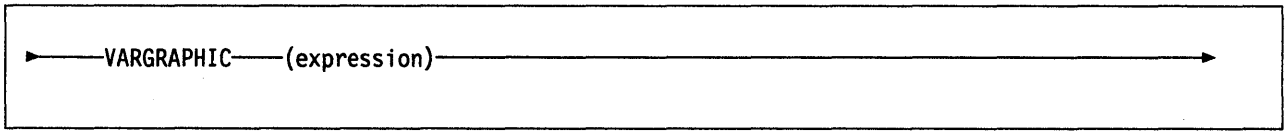
Numbers

If the arguments are numbers, the result is the numeric data type that would occur if all arguments were part of a single arithmetic expression. If that data type is decimal, it has precision of *p* and scale of *s* so *s* is the largest result scale of any argument, and *p* is the minimum of 15 and $s + n$, where *n* is the largest integral part result of any argument. Conversion errors are possible if $s + n$ is greater than 15.

Example: SALARY + VALUE(COMMISSION,0)

VARGRAPHIC

The VARGRAPHIC function obtains a graphic string representation of a character string. The form of the function is:



The argument must be a character string. If varying-length, the maximum length must not be greater than 254. If the string contains X'0E' or X'0F', they must be properly paired under the rules for mixed data.

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.

Regardless of whether the MIXED DATA install option is specified, the argument is interpreted as a mixed data string. The result includes all DBCS characters of the argument and the DBCS equivalent of all single-byte characters of the argument, the first character of the result is the first logical character of the argument, the second character of the result is the second logical character of the argument, and so on. The result does not include X'0E' or X'0F'.

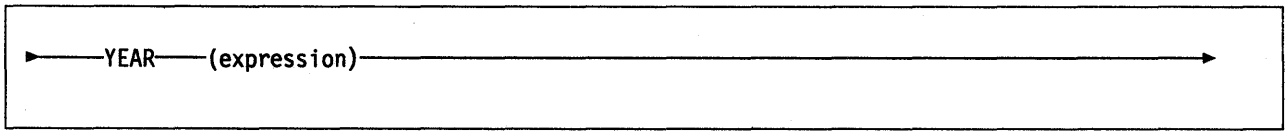
The DBCS equivalent of X'40' is X'4040'. The DBCS equivalent of other single-byte characters depends on the CHARACTER SET install option. If Katakana is specified and X'nn' is a Katakana code point, its DBCS equivalent is X'43nn'. If X'nn' is not a Katakana code point, its DBCS equivalent is X'42nn'. If Alphanumeric is specified, the DBCS equivalent of every single-byte character other than X'40' is X'42nn'.

The length of the result depends on the number of logical characters in the argument. If the length or maximum length of the argument is *n* bytes, the maximum length of the result is *n* (DBCS characters). If *n* is greater than 127, the result is subject to the restrictions that apply to long strings.

Example: `SUBSTR(VARGRAPHIC(MIXEDSTRING),:N)`

YEAR

The YEAR function obtains the year part of a value. The form of the function is:



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

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, 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: `YEAR(CURRDATE - BRTHDATE)`

Chapter 5. Queries

A *query* specifies a result table or intermediate result table.

In a program, a query is a component of other SQL statements. The three forms of a query described in this chapter are:

- The subselect,
- The fullselect, and
- The *select-statement*.

Note that there is another form of select, described under "SELECT INTO" on page 236.

Note: Where the syntax outlined in these descriptions is specifically limited to a *column-name*, (rather than to an *expression*), the column you identify must not be a column of a view derived from an expression, function, or constant.

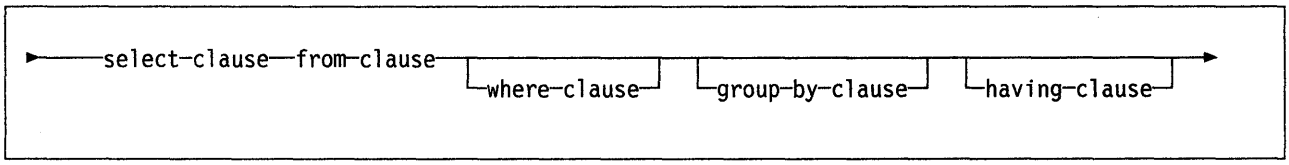
Authorization

When a select-statement is dynamically prepared, the union of the privileges designated by each authorization ID of the process must include at least one of the following privileges for every table or view identified in the select-statement:

- Ownership of the table or view
- The SELECT privilege on the table or view
- DBADM authority for the database (tables only)
- SYSADM authority.

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 the description of "CREATE VIEW" on page 161 for the authorization rules that apply to the subselect component of CREATE VIEW.

subselect



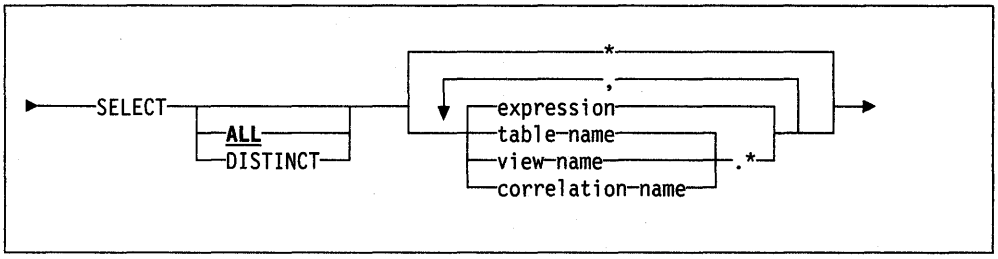
The *subselect* is a component of the fullselect, the CREATE VIEW statement, and the INSERT statement. It is also a component of certain predicates which, in turn, are components of a subselect. A subselect that is a component of another subselect is called a *subquery*.

A subselect specifies a result table derived from the tables or views identified in the 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



Produces a final result table by selecting only the columns indicated by the *select list* from R, where R is the result of the previous operation. For example, if the GROUP BY clause and HAVING clause are not specified, R is the result of the 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. This restriction includes SELECT DISTINCT and the use of DISTINCT in a column function of the select list or HAVING clause, but does not include subqueries of the subselect.

Two rows are duplicates of one another only if each value in the first is equal to the corresponding value of the second. (For determining duplicates, 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 at bind time. Hence * does not identify any columns that have been added to a table after bind time.

expression
May be any expression of the type described in Chapter 3, but commonly the expressions used include column names. Each column name used in the select list must unambiguously identify a column of R.

*name.**
Represents a list of names that identify the columns of *name*. *name* must also be used to designate a table or view 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 at bind time. Hence *name.** does not identify any columns that have been added to a table after bind time.

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 bind time), and may not exceed 300. The result of a subquery must be a single column unless the subquery is used in the EXISTS predicate.

Limitation on Long String Columns: No column in the list may be a long string column if:

- SELECT DISTINCT is used,
- the subselect is a subquery, or
- 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 or not GROUP BY or HAVING is used. Those results are described separately.

If neither GROUP BY nor HAVING is used:

- The select list must not include any column functions, or it must be entirely a list of column functions.
- If the select does not include column functions, then the select list 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, then R is the source of the arguments of the functions and the result of applying the select list is one row.

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.

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 do not allow null values if they are derived from:

- A scalar function or string expression that does not allow null values.
- A column that does not allow null values
- A constant
- The COUNT function
- A host variable that does not have an indicator variable.

Result columns allow null values if they are derived from:

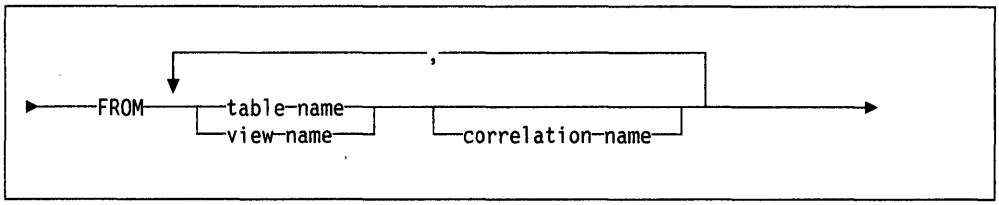
- Any column function but COUNT
- A column that allows null values
- An arithmetic expression in an outer select list
- An arithmetic expression that allows nulls
- A scalar function or string expression that allows null values
- A host variable that has an indicator variable.

Names of result columns: A result column derived from a column name acquires the unqualified name of that column. All other result columns have no names.

Data types of result columns: Each column of the result of SELECT acquires a data type from the expression from which it is derived.

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 variable	the same as the data type of the variable, with the same precision and scale for DECIMAL variables.
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 under "Expressions" on page 49.
any function	(see Chapter 4 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.
the name of any string variable	the same as the data type of the variable, with a length attribute equal to the length of the variable.
a character string constant of length *n*	VARCHAR(*n*)
a graphic string constant of length *n*	VARGRAPHIC(*n*)
the name of a date/time column	the same as the data type of the column.

from-clause



Names a single table or view, or produces an intermediate result table. The intermediate result table contains all possible combinations of the rows of the named tables or views. Each row of the result is a row from the first table or view concatenated with a row from the second table or view, concatenated in turn with a row from the third, and so on. The number of rows in the result is the product of the number of rows in all the named tables or views.

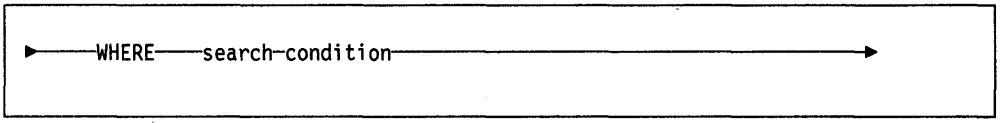
The list of names in the FROM clause must conform to these rules:

- A table-name or view-name must identify a table or view described in the catalog of the DB2 subsystem identified by the implicitly or explicitly specified location-name. If more than one table-name or view-name is specified, each location-name must identify the same DB2 subsystem.
- If the FROM clause is in a subquery of a basic predicate, no view named may use either GROUP BY or HAVING.

The FROM clause also defines the meaning of correlation names. A *correlation-name* applies to the table or view named by the immediately preceding *table-name* or *view-name*. If a correlation name is specified, then that correlation name must be used elsewhere in the subselect statement to designate that table or view. For rules governing the use of correlation names, see "Qualified Column Names" on page 43.

Each correlation name specified in the same FROM clause must be unique and must not be the same as a table name or view name specified in the clause. When the same table name or view name is specified more than once in a FROM clause, a correlation name must be specified after each occurrence of the replicated name. If a correlation name is specified for a table or view, any qualified reference to a column of that table or view in the subselect must use that correlation name.

where-clause



Produces an intermediate result table by applying *search-condition* to each row of R, where R is the result of the FROM clause. The result table contains the rows of R for which the search condition is true.

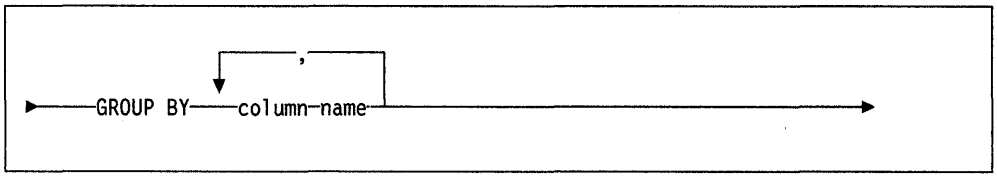
search-condition describes a search condition that conforms to these rules:

- The condition is formed as described in Chapter 3.

- Each *column-name* in the search condition either unambiguously identifies a column of R, or is a correlated reference. (A correlated reference is possible only in a subquery.)
- The search condition does not include a column function unless the argument of the function is a correlated reference to a group. (This is only possible in a subquery of a HAVING clause.)

Any subquery in the *search-condition* is effectively executed for each row of R and the results used in the application of the *search-condition* to the given row of R. 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



Produces an intermediate result table by grouping the rows of R, where R is the result of the previous clause.

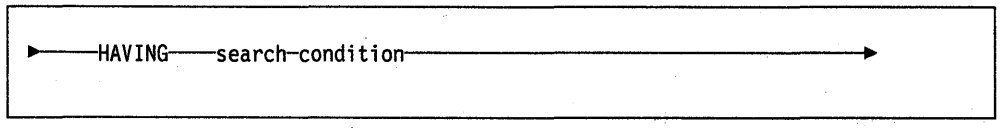
column-name unambiguously names a column of R. It must not specify a long string column. Each column named is called a *grouping column*. A grouping column must not be a view column derived from an expression, function, or constant, unless that column is part of a statement that meets certain special criteria. These criteria are discussed under "Use of Views: Special Criteria" on page 89.

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.

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

having-clause



Produces an intermediate result table by applying *search-condition* to each group of R where R is the result of the previous clause. If that clause is not GROUP BY, all rows of R are considered as one group. The result table contains those groups of R for which the search condition is true.

The *search-condition* describes a search condition that conforms to these rules:

- The condition is formed as described in Chapter 3.
- Each *column-name* in the *search-condition* must:
  - Unambiguously identify a grouping column of R, or
  - Be a correlated reference, or
  - Be specified within a column function.[3]

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 examples 4 and 5 under "Examples of a subselect" on page 90.

A correlated reference to a group of R must either identify a grouping column or be contained within a function.

The HAVING clause must not be used in a subquery of a basic predicate.

Use of Views: Special Criteria

As indicated in the descriptions of predicates, column functions, and the subselect, several restrictions on references to a column of a view do not apply in all cases. The instances in which these restrictions may be relaxed are determined by the language elements in the statement referring to the view and in the definition of the view itself. Those language elements pertaining to the view are not necessarily those of the referenced view, but also may be language elements that are used in a view on which the referenced view is dependent. References to views in the CREATE VIEW statement are included among the instances in which these restrictions are relaxed.

These restrictions do not apply if:

- The view includes DISTINCT, GROUP BY, HAVING or column functions in its SELECT list, and the subselect or embedded SELECT statement that references the view includes GROUP BY, HAVING, a join, or column functions in its SELECT list.
- The view includes DISTINCT, and the SELECT list of the subselect or embedded SELECT statement that references the view does not include a simple reference to every column of the view. That is, the SELECT list does not reference some columns of the view, or it references some columns only in an expression or scalar function.

---

[3] See Chapter 4, "Functions" on page 65 for restrictions that apply to the use of column functions.

- The view includes DISTINCT and the subselect or embedded SELECT statement that references the view includes DISTINCT.

Examples of a subselect

Example 1: Show all rows of DSN8220.EMP

```
SELECT * FROM DSN8220.EMP
```

Example 2: Show the job code, maximum salary, and minimum salary for each group of rows of DSN8220.EMP with the same job code, but only for groups with more than one row and with a maximum salary greater than $50,000.

```
SELECT JOBCODE, MAX(SALARY), MIN(SALARY)
  FROM DSN8220.EMP
  GROUP BY JOBCODE
  HAVING COUNT(*) > 1 AND MAX(SALARY) > 50000
```

Example 3: Show all rows of the Employee-to-Project-Activity table, for employees in Department E11.

```
SELECT * FROM DSN8220.EMP.PROJA
  WHERE EMPNO IN (SELECT EMPNO FROM DSN8220.EMP
                  WHERE WORKDEPT = 'E11')
```

Note that if you had wanted to choose a specific format for the two date columns returned by this example (EMSTDATE and EMENDATE), you could have coded the example using the CHAR function. If, for example, you had wanted the USA format for both columns, you might have coded the example like this:

```
SELECT EMPNO,PROJNO,ACTNO,EMPTIME,CHAR(EMSTDATE,USA),CHAR(EMENDATE,USA)
  FROM DSN8220.EMP.PROJA
    WHERE EMPNO IN (SELECT EMPNO FROM DSN8220.EMP
                    WHERE WORKDEPT = 'E11');
```

For information on the date/time string formats, see "Scalar Functions" on page 68.

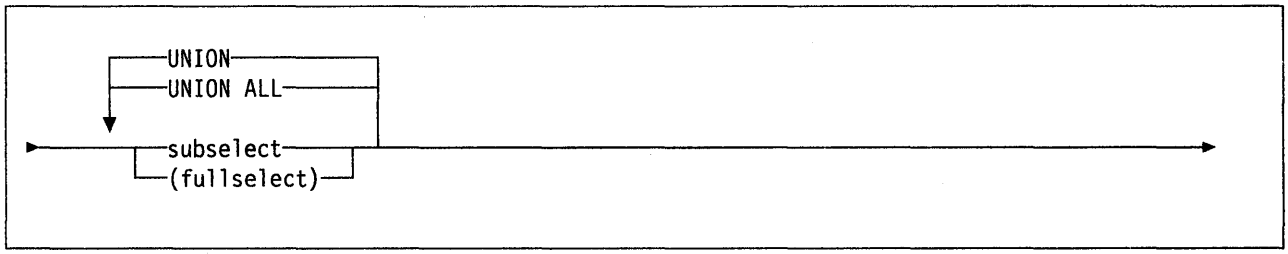
Example 4: Show the department number and maximum departmental salary for all departments whose maximum salary is less than the average salary for all employees. (In this example, the subselect would be executed only once.)

```
SELECT WORKDEPT, MAX(SALARY)
  FROM DSN8220.EMP
  GROUP BY WORKDEPT
  HAVING MAX(SALARY) < (SELECT AVG(SALARY)
                        FROM DSN8220.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 DSN8220.EMP Q
  GROUP BY WORKDEPT
  HAVING MAX(SALARY) < (SELECT AVG(SALARY)
                        FROM DSN8220.EMP
                        WHERE NOT WORKDEPT = Q.WORKDEPT)
```

fullselect



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

UNION or **UNION ALL**

Derives a result table by combining two other result tables (R1 and R2.). If UNION ALL is specified, the result consists of all rows in R1 and R2. If UNION is specified without the ALL option, the result is the set of all rows in either R1 or R2, with duplicate rows eliminated. In either case, however, each row of the UNION table is either a row from R1 or a row from R2. The columns of the result are not named.

Two rows are duplicates of one another only if each value in the first is equal to the corresponding value of the second. (For determining duplicates, two null values are considered equal.)

Note that the UNION ALL operation is associative, and that

```
(SELECT PROJNO FROM DSN8220.PROJ
UNION ALL
SELECT PROJNO FROM DSN8220.TPROJEC)
UNION ALL
SELECT PROJNO FROM DSN8220.EMPPROJA
```

will return the same results as

```
SELECT PROJNO FROM DSN8220.PROJ
UNION ALL
(SELECT PROJNO FROM DSN8220.TPROJEC
UNION ALL
SELECT PROJNO FROM DSN8220.EMPPROJA)
```

When you include the UNION ALL operator in the same SQL statement as a UNION operator, however, the result of the operation depends on the order of evaluation. Where there are no parentheses, evaluation is from left to right. Where parentheses are included, the parenthesized subselect is evaluated first, followed, from left to right, by the other components of the statement.

Rules for columns: R1 and R2 must have the same number of columns.

In the following explanations, let *Column1* denote the *n*th column of R1, *Column2* the *n*th column of R2, and *Column3* the *n*th column of the result of a UNION or UNION ALL.

- ***String Columns:*** *Column1* and *Column2* must be strings of the same type, either character or graphic. *Column3* will be a matching string. If UNION, rather than UNION ALL, is involved, neither *Column1* nor *Column2* can be a long string column. If both *Column1* and *Column2* are fixed-length, *Column3*

will be fixed-length. Otherwise, *Column3* will be varying-length. In either case, the length attribute of *Column3* will be the greater of the length attributes of *Column1* and *Column2*.

- ***Numeric Columns:*** *Column1* and *Column2* must both be numeric. The following rules govern the data type of *Column3*:
  - If *Column1* or *Column2* is floating-point, then, regardless of the data type of the other column, the result (*Column3*) is floating-point.
  - If *Column1* and *Column2* are decimal, *Column3* is decimal. If *p* and *s* are the precision and scale of *Column1*, and *p'* and *s'* are the precision and scale of *Column2*, the precision of *Column3* is MAX(*s*,*s'*)+MAX(*p*-*s*,*p'*-*s'*) and the scale of *Column3* is MAX(*s*,*s'*). The precision of *Column3* must not be greater than 15.
  - If *Column1* or *Column2* is decimal, and the other is integer, *Column3* is decimal. The integer is converted to decimal according to the rules for integer-to-decimal conversion. The precision and scale of *Column3* can be calculated, using the formulas above. For *p* and *s*, use the precision and scale of the decimal column. For *p'* and *s'*, use the precision and scale of the decimal representation of the integer column.
  - If *Column1* and *Column2* are large integer, *Column3* is large integer.
  - If *Column1* or *Column2* is large integer, and the other is small integer, *Column3* is large integer.
  - If *Column1* and *Column2* are small integer, *Column3* is small integer.
- ***Date/Time Columns:*** *Column1* and *Column2* must both be dates, both be times, or both be timestamps. *Column3* will have a matching data type.

In all cases, if *Column1* and *Column2* do not allow null values, *Column3* will not allow null values. Otherwise, *Column3* will permit null values. If the values of *Column1* or *Column2* must be converted to conform to *Column3*, the conversion operation is exactly the same as if the values were assigned to *Column3*. For example, if *Column1* is CHAR(10) and *Column2* is CHAR(5), *Column3* is CHAR(10) and values of *Column3* derived from *Column2* are padded on the right with five blanks.

Examples of a fullselect

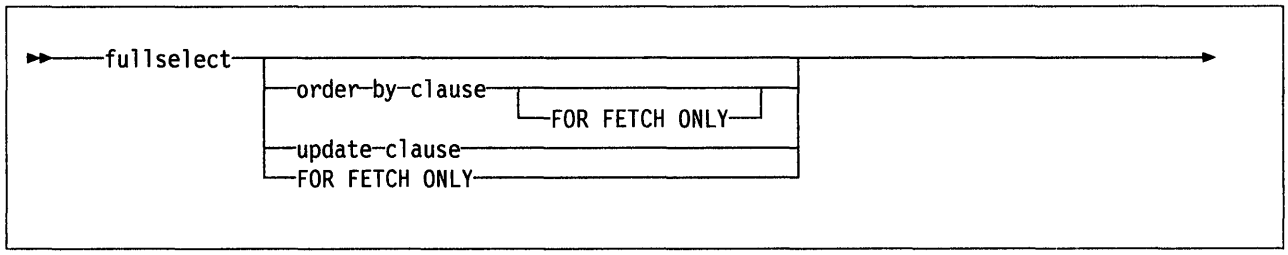
Example 1: Show all the rows from DSN8220.EMP.

```
SELECT * FROM DSN8220.EMP
```

Example 2: List the employee numbers of all employees whose department number begins with D (as determined from the employee table) OR who are assigned to projects whose project number begins with AD (as determined from the Employee-to-Project-Activity table).

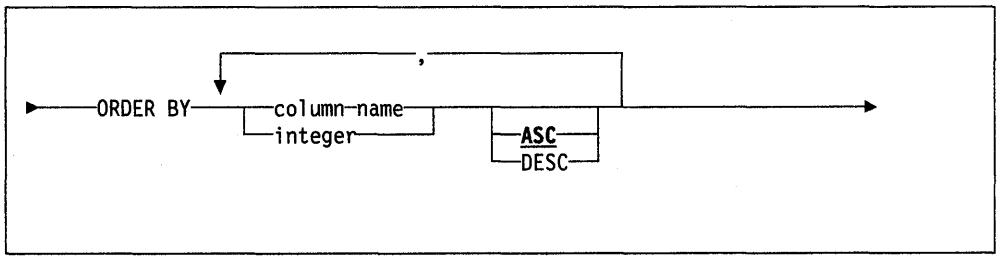
```
SELECT EMPNO FROM DSN8220.EMP
  WHERE WORKDEPT LIKE 'D%'
  UNION
  SELECT EMPNO FROM DSN8220.EMP.PROJA
  WHERE PROJNO LIKE 'AD%'
```

select-statement



The *select-statement* is the form of a query that can be prepared and subsequently executed by the use of an OPEN statement. It can also be issued interactively, by SPUFI, causing a result table to be displayed at your terminal. In either case, the table specified by a *select-statement* is the result of the fullselect.

order-by-clause



Puts the rows of the result table in order by the values of the columns you identify. If you identify more than one column, the rows are ordered by the values of the first column you identify, then by the values of the second column, and so on.

column-name
Must unambiguously identify a column of the result table. A long string column must not be identified.

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.

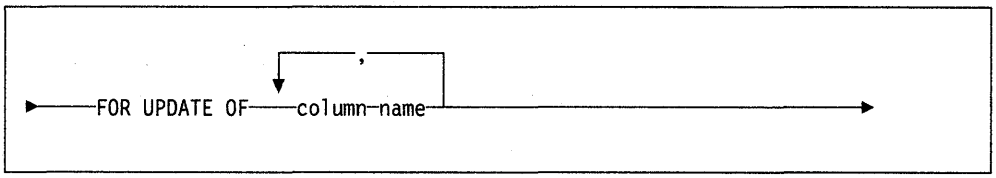
A named column may be identified by an *integer* or a *column-name*. An unnamed column must be identified by an integer. A column is unnamed if it is derived from a constant, an expression, or a function. If the fullselect includes a UNION operator, every column of the result table is unnamed.

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. 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



Used when the cursor will be referred to in a positioned UPDATE statement. Restricts the columns that can be updated to those specified in the *column-name* list. These columns must belong to the table or view named in the FROM clause of the fullselect. The column names must not be qualified.

An update clause is mandatory when a cursor is used for updates, with one exception: Consider a query appearing explicitly in a DECLARE CURSOR statement, and assume that its program is processed by the DB2 precompiler. Then the query needs no update clause if one or both of the following options is in effect for the precompilation: STDSQL(86) or NOFOR. For more on the subject, see "The NOFOR Option: FOR UPDATE OF" on page 40.

When FOR UPDATE OF is used and the plan is bound with cursor stability, FETCH operations referencing the cursor acquire U locks, rather than S locks. (For a discussion of U locks and S locks, see Section 6 of *System and Database Administration Guide*.)

The FOR UPDATE OF clause cannot be used if the result is read-only. Nor can it be used in a statement containing FOR FETCH ONLY. For a discussion of read-only result tables, see "DECLARE CURSOR" on page 165.

FOR FETCH ONLY

Indicates that the result table is read-only; that is, that its cursor will not be used for positioned deletes or updates.

Some result tables may be read-only because their nature precludes the use of delete or update operations. (Such tables may, for example, be based on read-only views.) For all such tables, FOR FETCH ONLY is redundant. But for result tables for which updates and deletes are possible, specifying FOR FETCH ONLY may improve the performance of FETCH operations.

A read-only result table must not be referenced in an UPDATE or DELETE statement. A violation of this rule is indicated by SQLCODE -510. The rule embraces result tables that are read-only by their nature, and those for which FOR FETCH ONLY is specified.

FOR FETCH ONLY cannot appear in a statement containing an update-clause.

Examples of a select-statement

Example 1: Select all the rows from DSN8220.EMP.

```
SELECT * FROM DSN8220.EMP
```

Example 2: Select all the rows from DSN8220.EMP in order by date of hiring.

```
SELECT * FROM DSN8220.EMP ORDER BY HIREDATE
```

Chapter 6. Statements

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

Table 4 (Page 1 of 2). SQL Statements

SQL Statement	Function	Refer to
ALTER INDEX	Changes the description of an index.	p. 100
ALTER STOGROUP	Changes the description of a storage group.	p. 106
ALTER TABLE	Changes the description of a table.	p. 108
ALTER TABLESPACE	Changes the description of a table space.	p. 115
BEGIN DECLARE SECTION	Marks the beginning of a host variable declaration section.	p. 122
CLOSE	Closes a cursor.	p. 123
COMMENT ON	Replaces or adds a comment to the description of a table, view, alias, or column.	p. 124
COMMIT	Terminates a unit of recovery and commits the database changes made by that unit of recovery.	p. 126
CREATE ALIAS	Defines an alias.	p. 127
CREATE DATABASE	Defines a database.	p. 129
CREATE INDEX	Defines an index on a table.	p. 131
CREATE STOGROUP	Defines a storage group.	p. 140
CREATE SYNONYM	Defines an alternate name for a table or view.	p. 142
CREATE TABLE	Defines a table.	p. 144
CREATE TABLESPACE	Allocates and formats a table space.	p. 154
CREATE VIEW	Defines a view of one or more tables or views.	p. 161
DECLARE CURSOR	Defines an SQL cursor.	p. 165
DECLARE STATEMENT	Declares names used to identify prepared SQL statements.	p. 168
DECLARE TABLE	Provides the programmer and the precompiler with a description of a table or view.	p. 169
DELETE	Deletes one or more rows from a table.	p. 172
DESCRIBE	Describes the result columns of a prepared statement.	p. 176
DROP	Deletes an alias, database, index, storage group, synonym, table, tablespace, or view.	p. 179
END DECLARE SECTION	Marks the end of a host variable declaration section.	p. 183
EXECUTE	Executes a prepared SQL statement.	p. 184
EXECUTE IMMEDIATE	Prepares and executes an SQL statement.	p. 186

Table 4 (Page 2 of 2). SQL Statements		
SQL Statement	**Function**	**Refer to**
EXPLAIN	Obtains information about how an SQL statement would be executed.	p. 188
FETCH	Assigns values of a row to host variables.	p. 192
GRANT (DATABASE PRIVILEGES)	Grants privileges on databases.	p. 196
GRANT (PLAN PRIVILEGES)	Grants authority to bind or execute an application plan.	p. 198
GRANT (SYSTEM PRIVILEGES)	Grants system privileges.	p. 199
GRANT (TABLE or VIEW PRIVILEGES)	Grants privileges on a table or view.	p. 201
GRANT (USE PRIVILEGES)	Grants authority to use specified buffer pools, storage groups, or table spaces.	p. 203
INCLUDE	Inserts declarations into a source program.	p. 205
INSERT	Inserts one or more rows into a table.	p. 207
LABEL ON	Replaces or adds a label on the description of a table, view, alias, or column.	p. 211
LOCK TABLE	Locks a table in shared or exclusive mode.	p. 213
OPEN	Opens a cursor.	p. 215
PREPARE	Prepares an SQL statement (with optional parameters) for execution.	p. 218
REVOKE (DATABASE PRIVILEGES)	Revokes privileges on databases.	p. 224
REVOKE (PLAN PRIVILEGES)	Revokes authority to bind or execute an application plan	p. 227
REVOKE (SYSTEM PRIVILEGES)	Revokes system privileges.	p. 229
REVOKE (TABLE or VIEW PRIVILEGES)	Revokes privileges on a table or view.	p. 231
REVOKE (USE PRIVILEGES)	Revokes authority to use specified buffer pools, storage groups, or table spaces.	p. 233
ROLLBACK	Terminates a unit of recovery and backs out the database changes made by that unit of recovery.	p. 235
SELECT INTO	Specifies a result table of no more than one row and assigns the values to host variables.	p. 236
SET CURRENT SQLID	Changes the value of the SQL authorization ID.	p. 238
UPDATE	Updates the values of one or more columns in one or more rows of a table.	p. 240
WHENEVER	Defines actions to be taken on the basis of SQL return codes.	p. 245

How SQL Statements Are Invoked

The SQL statements described in this chapter are classified as *executable* or *nonexecutable*. The 'Invocation' section in the description of each statement indicates whether or not the statement is executable.

An *executable statement* can be invoked in three ways:

- Embedded in an application program
- Dynamically prepared and executed
- Issued interactively.

Depending on the statement, you can use some or all of these methods. The 'Invocation' section in the description of each statement tells you which methods can be used.

A *nonexecutable statement* can only be embedded in an application program.

Besides the statements described in this chapter, there is one more SQL statement construct: the *select-statement*, as described under "select-statement" on page 93. It is not included in this chapter because it is used in a way different from other statements. A *select-statement* can be invoked in three ways:

- Included in DECLARE CURSOR and implicitly executed by OPEN
- Dynamically prepared, referenced in DECLARE CURSOR, and implicitly executed by OPEN
- Issued interactively.

The first two methods are called, respectively, the *static* and the *dynamic* invocation of *select-statement*.

The different methods of invoking an SQL statement are discussed below in more detail. For each method, the discussion includes: the mechanism of execution, the interaction with host variables, and testing whether the execution was successful or not.

Embedding a Statement in an Application Program

You may include SQL statements in a source program that will be submitted to the precompiler. Such statements are said to be *embedded* in the program. An embedded statement can be placed anywhere in the program where a host language statement would be 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 may contain references to host variables. A host variable referenced in this way may be used in 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, i.e., 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 of the SQLCODE field in SQLCA. You should therefore follow all executable statements by a test of SQLCODE. 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 precompiler reports any errors encountered in such statement. The statement is *never* executed, and acts as a no-operation if placed among executable statements of the application program. You should not, therefore, follow such statements by a test of the SQLCODE field in SQLCA.

Dynamic Preparation and Execution

Your application program may 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 program (for example, obtained 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. Alternatively, you can use the (embedded) statement EXECUTE IMMEDIATE to prepare and execute a statement in one step.

A statement to be prepared must not contain references to host variables. It may instead contain parameter markers. (See "PREPARE" on page 218 for rules concerning the 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 184 for rules concerning this replacement.) Note that, once prepared, a statement can be executed several times, with different values of host variables.

The parameter markers are not allowed by EXECUTE IMMEDIATE.

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

Static Invocation of a select-statement

You may 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.

The *select-statement* used in this way may contain references to host variables. These references are effectively replaced by the values that the variables have at the moment of executing OPEN.

The successful or unsuccessful execution of *select-statement* is indicated by setting of the SQLCODE field in SQLCA after the OPEN. You should check it as described above for embedded statements.

Dynamic Invocation of a select-statement

Your application program may 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 program (for example, a query expressed in terms of your application, obtained from a terminal). The statement so constructed may be prepared for execution by means of the (embedded) statement PREPARE, and referenced 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 this way must not contain references to host variables. It may instead contain parameter markers. (See "PREPARE" on page 218 for rules concerning the parameter markers.) The parameter markers are effectively replaced by the values of the host variables specified in the OPEN statement. (See "OPEN" on page 215 for rules concerning this replacement.)

The successful or unsuccessful execution of *select-statement* is indicated by setting of the SQLCODE field in SQLCA after the OPEN. You should check it as described above for embedded statements.

Interactive Invocation

A capability for entering SQL statements from a terminal is part of the architecture of the database management system. For this facility, DB2 provides SPUFI. 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, since these make sense only in the context of an application program. For the same reason, there is no SQLCA involved.

ALTER INDEX

The ALTER INDEX statement changes the description of a local index.

Invocation

This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared.

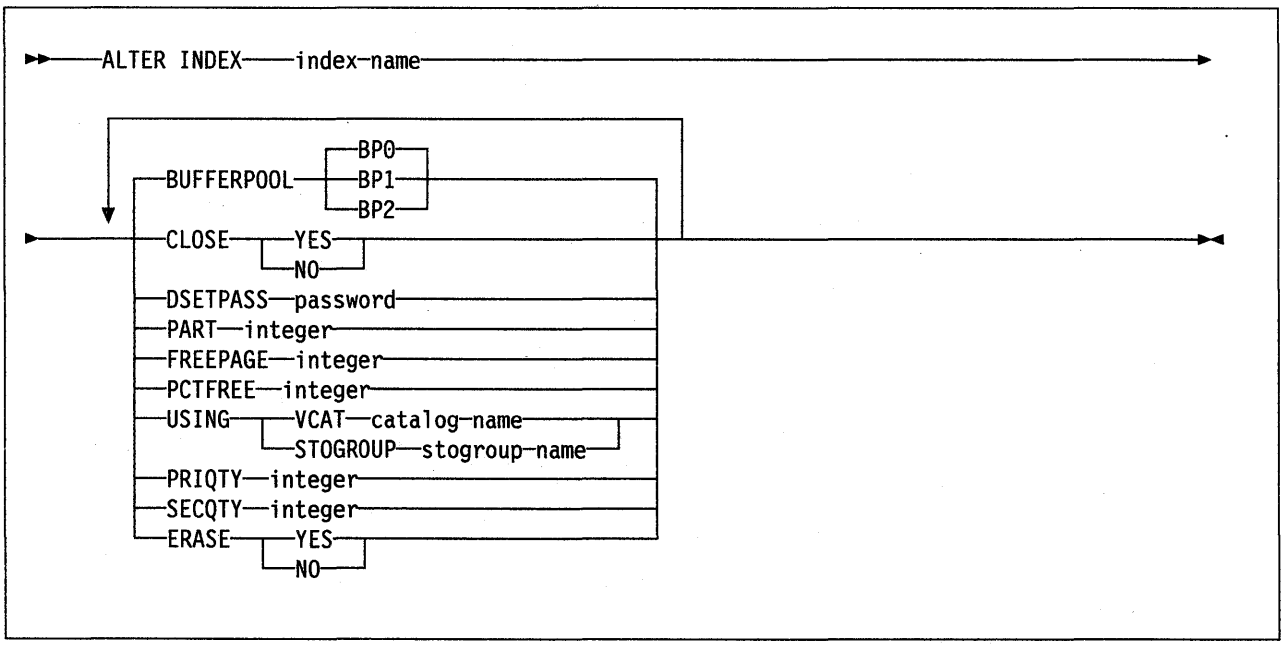
Authorization

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

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

If BUFFERPOOL or USING STOGROUP is specified, additional privileges may be required as explained in the description of those clauses.

If the statement is embedded in a program, the privilege set is the privileges designated by the authorization ID of the owner of the plan. If the statement is dynamically prepared, the privilege set is the union of the privileges designated by each authorization ID of the process.



Syntax note: Use at least one of the keywords from the diagram above, but **do not use the same keyword more than once.**

Description

index-name
Is the name of the index to be altered. It must be a user-created index described in the DB2 catalog.

BUFFERPOOL BP*n*
Names the buffer pool to be associated with the index. Use BP0, BP1, or BP2. The buffer pool must be activated, and you must have SYSADM authority or the USE privilege for the buffer pool. The change to the description of the index has no effect until the next time its data sets are opened.

CLOSE
Specifies whether or not the data sets for the index are closed when the number of processes using the index becomes zero.

YES
Closes the data sets.

NO
Does not close the data sets.

DSETPASS *password*
Specifies a password that is passed to VSAM when the data sets of the index are used by DB2. *password* is a VSAM master level password in the form of a short identifier. If the password is a delimited identifier, it may contain any special characters acceptable to VSAM Access Method Services. The change to the description of the index has no effect until the next time its data sets are opened.

Changing the password for the index does not change the password that protects its data sets. To change the data set password, use VSAM Access Method Services.

Note: The password does not apply to the data sets managed by Storage Management Subsystem. Data sets defined to SMS should be protected by RACF or some 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. You must use this clause if the index is partitioned and you use the FREEPAGE, PCTFREE, USING, PRIQTY, SECQTY, or ERASE 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 the index or partition is loaded or reorganized. You must specify an integer in the range of 0 to 255. If you specify 0, no free pages are left as free space. Otherwise, one free page is left after every *n* pages where *n* is the specified integer. The change to the description of the index or partition has no effect until it is loaded or reorganized.

PCTFREE *integer*
Specifies the amount of free space to leave in each nonleaf page and subpage when the index or partition is loaded or reorganized. You must specify an integer in the range 0 to 99. The integer indicates a percentage of free space that does not restrict the first entry in a nonleaf page or subpage. When additional entries are placed in a nonleaf page, at least *n*% of the page is left as free space, where *n* is the specified integer. When additional entries are placed in a subpage, at least *n*/*m*% of the subpage is left as free space, where m is the number of subpages in the leaf pages of the index. The change to the description of the index or partition has no effect until it is loaded or reorganized.

USING

Specifies whether a data set for the index or partition is user-managed or DB2-managed. If the index is partitioned, USING applies to the data set for the partition identified in the PART clause. If the index is not partitioned, USING applies to every data set that may be used for the index. (An unpartitioned index can have more than one data set if PRIQTY + 123 * SECQTY is at least 2 gigabytes.)

If you use the USING clause, the index must be in the stopped state when the ALTER INDEX statement is executed. Successful execution of the statement changes the description of the index but has no immediate effect on the index or partition. The new description is applied when the index or partition is recovered, reorganized, or extended to a new volume or data set. However, if you change the *catalog-name* (explicitly by USING VCAT or implicitly by USING STOGROUP), you must move the data while the index is in the stopped state as explained under "Notes" on page 104.

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 ICF catalog is longer than 8 characters. When the new description of the index is applied, the ICF catalog must contain an entry for the data set conforming to the DB2 naming conventions.

STOGROUP *stogroup-name*

Specifies a DB2-managed data set that resides on a volume of the specified storage group. You must specify a *stogroup-name* described in the DB2 catalog and the privilege set must include SYSADM 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 ICF catalog used for the storage group must not contain an entry for the data set and, if the ICF 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 DB2-managed, omission of the PRIQTY clause is an implicit specification of the current PRIQTY value, omission of the SECQTY clause is an implicit specification of the current SECQTY value, and 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 user-managed, omission of the PRIQTY clause is an implicit specification of PRIQTY 12, omission of the SECQTY clause is an implicit specification of SECQTY 12, and omission of the ERASE clause is an implicit specification of ERASE NO.

PRIQTY *integer*

Specifies the primary space allocation for a data set of the index or partition. This clause must not be specified for user-managed data sets. Thus, you may specify this clause only if:

- You also specify USING STOGROUP, or
- You do not specify the USING clause and the data set for the index or partition is DB2-managed.

If you specify PRIQTY, the index must be in the stopped state when the ALTER INDEX statement is executed. Successful execution of the statement changes the description of the index, but has no immediate effect on the index or partition. The new description is applied when the index or partition is recovered, reorganized, or extended to a new volume or data set. However, you may want to apply the new description before then by moving the data as explained under "Notes" on page 104.

You specify the primary space allocation in kilobytes. You may specify any integer, but 12 is used if the integer you specify is less than 12 and 4194304 is used if the integer you specify is greater than 4194304.

DB2 specifies the primary space allocation to VSAM as an integral number of pages which is calculated by dividing the number of kilobytes by 4 and rounding up to the next highest integer. VSAM then uses this number to specify an integral number of tracks. Thus, the primary space allocation can be greater than the number of kilobytes you specify.

SECQTY *integer*

Specifies the secondary space allocation for a data set of the index or partition. This clause must not be specified for user-managed data sets. Thus, you may specify this clause only if:

- You also specify USING STOGROUP, or
- You do not specify the USING clause and the data set for the index or partition is DB2-managed.

If you specify SECQTY, the index must be in the stopped state when the ALTER INDEX statement is executed. Successful execution of the statement changes the description of the index, but has no immediate effect on the index or partition. The new description is applied when the index or partition is recovered, reorganized, or extended to a new volume or data set. However, you may want to apply the new description before then by moving the data as explained under "Notes" on page 104 below.

You specify the secondary space allocation in kilobytes. You may specify any integer, but 131068 is used if the integer you specify is greater than 131068. If you specify 0, the data set cannot be extended.
DB2 specifies the secondary space allocation to VSAM as an integral number of pages which is calculated by dividing the number of kilobytes by 4 and rounding up to the next highest integer. VSAM then uses this number to specify an integral number of tracks. Thus, the secondary space allocation can be greater than the number of kilobytes you specify.

ERASE

Specifies whether the contents of a data set for the index or partition are erased when the index is dropped. This clause must not be specified for user-managed data sets. Thus, you may specify this clause only if:

- you also specify USING STOGROUP, or
- you do not specify the USING clause and the data set for the index or partition is DB2-managed.

If you specify ERASE, the index must be in the stopped state when the ALTER INDEX statement is executed. Successful execution of the statement changes the description of the index, but has no immediate effect on the index or partition. The new description is applied when the index or partition is recovered, reorganized, or extended to a new volume or data set. However,

you may want to apply the new description before then by moving the data as explained under "Notes" on page 104.

YES
Erases the contents of the data set when the index is dropped.

NO
Does not erase the contents of the data set.

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, or ERASE for more than one partition, you must use separate ALTER INDEX statements.

USING, PRIQTY, SECQTY, and ERASE specify the storage attributes of an index or partition. Changing these attributes requires changes to the description of the index and movement of the data. The ALTER statement only changes the description of the index. Movement of the data is a user responsibility which can be accomplished by the use of RECOVER, REORG, DSN1COPY, or a non-DB2 facility such as DFP/DFDSS.

Changing the storage attributes of an index or partition involves the use of a STOP DATABASE command, an ALTER INDEX statement, and a START DATABASE command, in that order. The usual procedure involves movement of the data while the index is in the stopped state, but this is necessary only if the *catalog-name*s changed which is the case if the purpose of the procedure is to move the data to a different device. If the purpose is only to change the space allocation parameters or erase rule, the data need not be moved before the index is restarted. In this case, the alterations will take effect when the index or partition is subsequently recovered or reorganized.

If the *catalog-name* is explicitly or implicitly changed by the ALTER, the data must be moved while the index is in the stopped state. If DSN1COPY or a non-DB2 facility is used, the move can be done either before or after the ALTER statement is executed. For example, DFP/DFDSS can be used to move the data and change the high level qualifier of the data set name before the ALTER statement is executed.

Any facility can be used to move the data while the index is in the stopped state if the *catalog-name* has not been changed. If RECOVER or REORG is used, the ALTER statement must be executed first and then the index started for utility use only. If another facility is used, the sequence is irrelevant, but the user is responsible for ensuring that the name of the data set conforms to the DB2 naming conventions when the index is restarted.

Examples

Example 1: Alter the index DSN8220.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 DSN8220.XEMP1
  CLOSE NO;
```

Example 2: Alter the index DSN8220.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 DSN8220.XPROJ1
  BUFFERPOOL BP1
  DSETPASS OSESAME;
```

ALTER STOGROUP

The ALTER STOGROUP statement changes the description of a local storage group.

Invocation

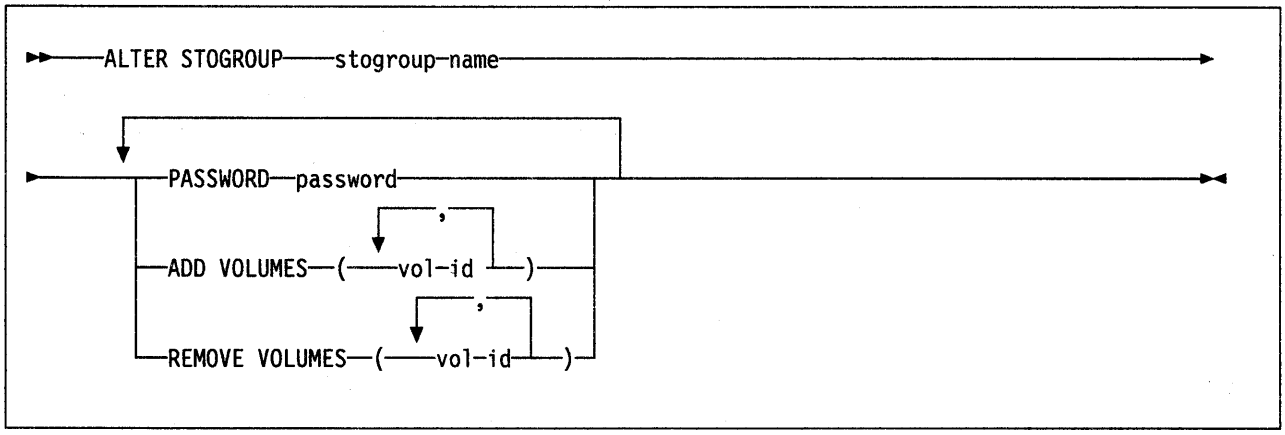
This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared.

Authorization

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

- Ownership of the storage group
- SYSADM authority.

If the statement is embedded in a program, the privilege set is the privileges designated by the authorization ID of the owner of the plan. If the statement is dynamically prepared, the privilege set is the union of the privileges designated by each authorization ID of the process.



Syntax note: Use at least one of the keywords PASSWORD, ADD VOLUMES, and REMOVE VOLUMES, but **do not use the same keyword more than once.**

Description

stogroup-name
Is the name of the storage group to be altered. It must be a storage group described in the DB2 catalog.

PASSWORD *password*
Gives a password that is passed to VSAM when DB2 accesses the ICF catalog. *password* is a VSAM master level password in the form of a short identifier. If the password is a delimited identifier, it may contain any special characters acceptable to VSAM Access Method Services. You can set the password that protects the catalog through VSAM Access Method Services.

ADD VOLUMES (*vol-id*)
Adds volumes to the storage group. *vol-id* is the volume serial number of one or more storage volumes to be added. It may have a maximum of 6 characters and is specified as an identifier or a string constant.

You cannot add a volume that is already in the storage group unless you also remove it with REMOVE.

REMOVE VOLUMES (*vol-id*)

Removes volumes from the storage group. *vol-id* is the volume serial number of one or more storage volumes to be removed. Each *vol-id* must identify a volume that is in the storage group.

Removing a volume from a storage group does not affect existing data, but a volume that has been removed will not be used again when the storage group is used to allocate storage for tablespaces or indexspaces.

Notes

If the storage group altered contains temporary file database (DSNDB07) data sets, the database must be stopped and restarted for the affects of the ALTER to be recognized. Issue the command -STOP DATABASE(DSNDB07), followed by -START DATABASE(DSNDB07).

For ADD VOLUMES: When the storage group is used, an error occurs if all volumes are not of the same device type or if any volume is not available to MVS for dynamic allocation of data sets.

When a storage group is used to extend a data set, the volumes must have the same device type as the volumes used when the data set was defined.

There is no specific limit on the number of volumes that can be defined for a storage group. However, the maximum number of volumes used for allocation is 133. Therefore, do not add volumes such that the number of volumes defined for the storage group exceeds 133.

DB2 acts solely as a conduit for the specified volumes. As such, it passes the list of volumes to DFP for initial dataset definitions and for dataset extensions. All valid volume serial parameters, acceptable to DFP (AMS and VSAM), are acceptable in this statement.

Examples

Example 1: Alter storage group DSN8G220. OSESAME is the password that is used to access the ICF catalog. DSNV04,DSNV05 are the volumes to be added.

```
ALTER STOGROUP DSN8G220
  PASSWORD OSESAME
  ADD VOLUMES (DSNV04,DSNV05);
```

Example 2: Alter storage group DSN8G220. DSNV04,DSNV05 are the volumes to be removed.

```
ALTER STOGROUP DSN8G220
  REMOVE VOLUMES (DSNV04,DSNV05);
```

ALTER TABLE

The ALTER TABLE statement changes the description of a local table.

Invocation

This statement can be embedded in an application or issued interactively. It is an executable statement that can be dynamically prepared.

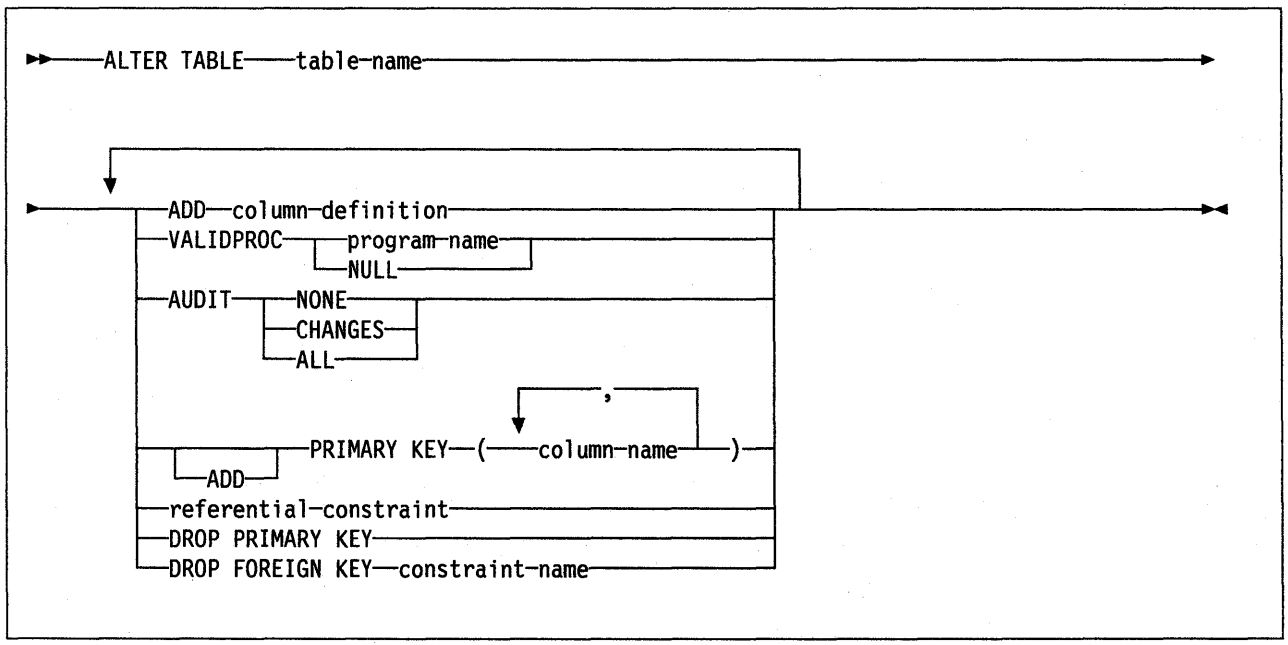
Authorization

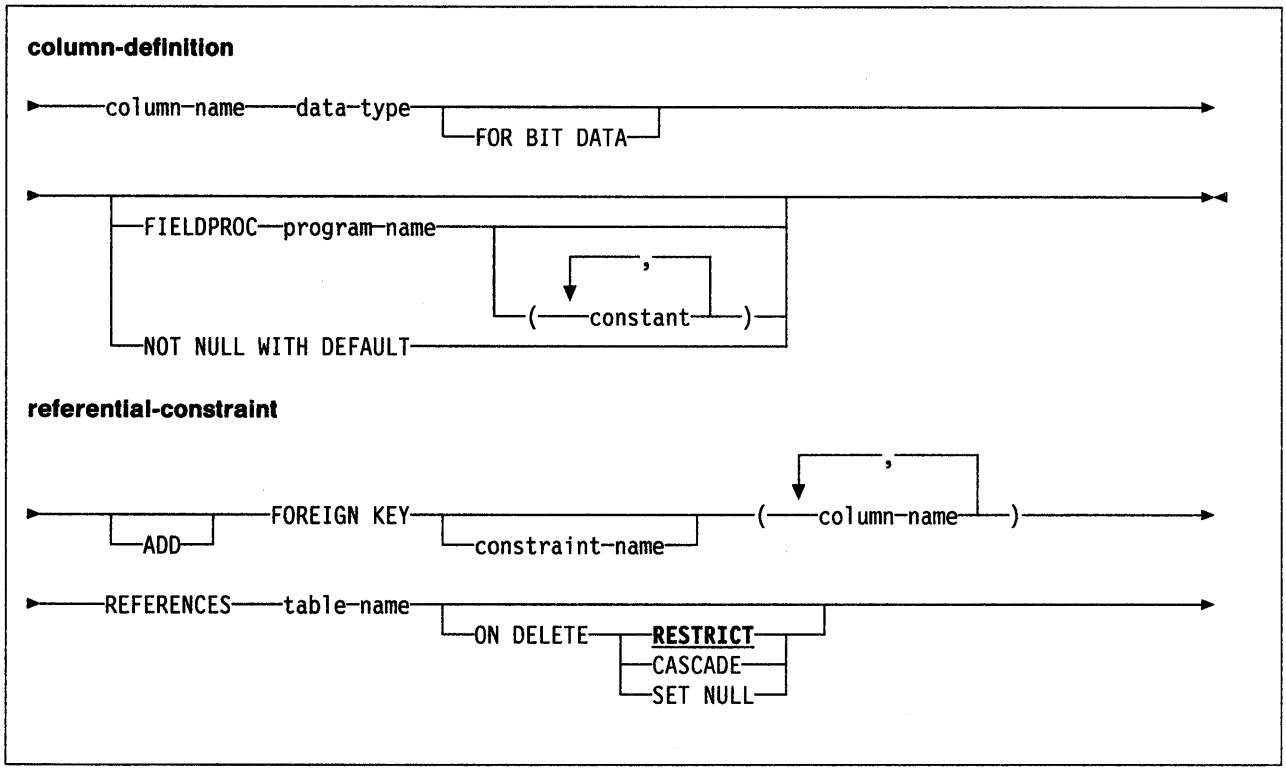
The privilege set must include at least one of the following:

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

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

If the statement is embedded in a program, the privilege set is the privileges designated by the authorization ID of the owner of the plan. If the statement is dynamically prepared, the privilege set is the union of the privileges designated by each authorization ID of the process.





Syntax note: Use at least one of the keywords from the diagram above, but **do not use the same keyword more than once.**

Description

table-name
Is the name of the table you want to change. It must be a table described in the catalog and must not be a view or a catalog table. Although a three-part name can be used, the identified table must be local.

ADD
Adds a column to the table. All values of the column in existing rows are its default value and the column is the 'last' column of the table. That is, if initially there are *n* columns, the added column is column *n*+1. The value of *n* cannot be greater than 299.

The columns 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 six less than the maximum record size shown under "Notes" on page 151.

column-definition

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
Is one of the data types listed under "CREATE TABLE" on page 144.

FOR BIT DATA
Specifies the contents of the column are to be treated as bit (binary) data for exchange with other systems. This option only applies to CHAR, VARCHAR, and LONG VARCHAR columns. If this option is not present, the character column is assumed to contain EBCDIC characters.

FIELDPROC *program-name*

Names a field procedure for the column. The procedure named *program-name* is provided by your installation. You may use a field procedure only with a short string column. You may not use a field procedure and NOT NULL WITH DEFAULT for the same column.

If you omit FIELDPROC, the column has no field procedure.

constant

Is a parameter passed to the field procedure when the ALTER TABLE statement invokes it. A parameter list is optional; the number of parameters and the data type of each are determined by the field procedure. The maximum length of the parameter list is 254 bytes, including commas but excluding insignificant blanks and the delimiting parentheses.

NOT NULL WITH DEFAULT

Prevents the column from containing null values, and allows a default value other than the null value. The default value used 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, the current date.
Time	For existing rows, a time corresponding to 0 hours, 0 minutes, and 0 seconds. For added rows, the 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 zero microseconds. For added rows, the current timestamp.

VALIDPROC

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

program-name

Names a validation routine for the table. The program, which must be provided by your site, is invoked during the execution of LOAD, INSERT, UPDATE, and DELETE operations on the table. The program receives a row and returns a value that indicates whether the operation should proceed for that row. A typical use is to impose constraints on the values of rows.

A table may have only one validation procedure at a time. If you name a new procedure, any existing procedure is no longer used. The new procedure is not used to validate existing rows of the table; it is only used for rows that are loaded, inserted, updated, or deleted after execution of the ALTER TABLE statement.

NULL
Inhibits execution of any validation procedure.

AUDIT
Alters the auditing attribute of the table.

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 trace class is active. For information about trace classes, see Section 6 (Volume 2) of *Administration Guide*.

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 recovery of a utility or application process. However, the auditing is done only if the appropriate trace class is active and the access is performed through the data manager (COPY, RECOVER, REPAIR, and the stand-alone utilities do not use the data manager.).

PRIMARY KEY (*column-name*)
Defines a primary key composed of the identified columns. Each *column-name* must identify a column of the table, and the same column must not be identified more than once. The number of identified columns must not exceed 16 and the sum of their length attributes must not exceed 254. The table must not have a primary key and the identified columns must be defined as NOT NULL or NOT NULL WITH DEFAULT.

The *table-name* 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 nth *column-name* of one is the same as the nth *column-name* of the other.

The identified columns are defined as the primary key of the table and all plans that reference the table are invalidated. 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.

referential-constraint

FOREIGN KEY (*column-name*) REFERENCES table-name
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 a *constraint-name* is specified, it must not be the same as the name of an existing referential constraint 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 name in the list of column names must identify a column of T1 and the same column must not be identified more than once. The number of identified columns must not exceed 16 and the sum of their length attributes must not exceed 254 minus the number of columns that

allow null values. If the specified list of column names is identical to a list of column names of an existing referential constraint on T1, the table identified in the FOREIGN KEY clause must not be the parent table of that referential constraint.

The table specified in a FOREIGN KEY clause must identify a table that is described in the catalog, but must not identify a catalog table. Let T2 denote the identified table (T1 and T2 may be the same table).

T2 must have a primary index and the privilege set must include the ALTER privilege on T2.

The specified foreign key must have the same number of columns as the primary key of T2 and, except for their names, default values, and null attributes, the description of the nth column of the foreign key must be identical to the description of the nth column of that primary key. If a column of the foreign key has a field procedure, the corresponding column of the primary key must have the same field procedure and an identical field description.

The table space that contains T1 must be available to DB2. If T1 is populated, its table space is placed in the CHECK PENDING state. A table in a segmented table space is populated if the table is not empty. A table in an unsegmented 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.

The delete rule of the relationship is determined by the ON DELETE clause. If T1 and T2 are the same table, CASCADE 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. Omission of the clause is an implicit specification of ON DELETE RESTRICT.

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 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. Thus, if T1 is a dependent of T3 in a relationship with a delete rule of *r* and

- T2 and T3 are the same table, or

- T2 is a descendant of T3 and the deletion of rows from T3 cascades to T2, or
- T2 and T3 are both descendants of the same table and the deletion of rows from that table cascades to both T2 and T3:

the referential constraint cannot be defined if *r* is SET NULL. If *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*.

If T1 and T2 are the same table, all plans that reference the table are invalidated. If T2 and T1 are different tables and the specified delete rule is CASCADE or SET NULL, all plans that reference T2 are invalidated and all plans that reference a table from which deletes cascade to T2 are invalidated.

DROP PRIMARY KEY
Drops the definition of the primary key and all referential constraints in which the table is a parent. The table must have a primary key and the privilege set must include the ALTER 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 and the privilege set must include the ALTER privilege on the parent table of that relationship.

Notes

The successful execution of an ALTER TABLE statement that includes the key word AUDIT is audited if the appropriate trace class is active. When the audit attribute of a table is changed, all plans that reference the table are invalidated.

The clauses of an ALTER TABLE statement are processed in the order in which they are specified.

Adding a column to a table has no effect on existing views.

When using ALTER TABLE, you cannot:

- Use NOT NULL. (You may use NOT NULL WITH DEFAULT.)
- Add a column if an edit procedure exists for the table.
- Execute ALTER TABLE while a utility has control of the table space that contains the table.

Any table that may be involved in a DELETE operation on table T is said to be delete-connected to T. Thus, a table is delete-connected to T if it is a dependent of T or it is a dependent of a table to which deletes from T cascade.

Examples

Example 1: Alter table DSN8220.DEPT. Add the column BLDG, which will contain character data.

```
ALTER TABLE DSN8220.DEPT
  ADD BLDG CHAR(3);
```

Example 2: Alter table DSN8220.EMP. DSN8EAEM is the validation routine that will check inserted or updated values.

```
ALTER TABLE DSN8220.EMP
  VALIDPROC DSN8EAEM;
```

Example 3: Alter table DSN8220.EMP. VALIDPROC NULL causes the validation procedure to be disconnected from the table.

```
ALTER TABLE DSN8220.EMP
  VALIDPROC NULL;
```

Example 4: Alter table DSN8220.DEPT. Establish the column ADMRDEPT as a foreign key of the parent table DSN8220.DEPT, and make any delete rules that apply to the parent table apply to ADMREDEPT as well.

```
ALTER TABLE DSN8220.DEPT
  FOREIGN KEY(ADMRDEPT) REFERENCES DSN8220.DEPT ON DELETE CASCADE;
```

ALTER TABLESPACE

The ALTER TABLESPACE statement changes the description of a local 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.

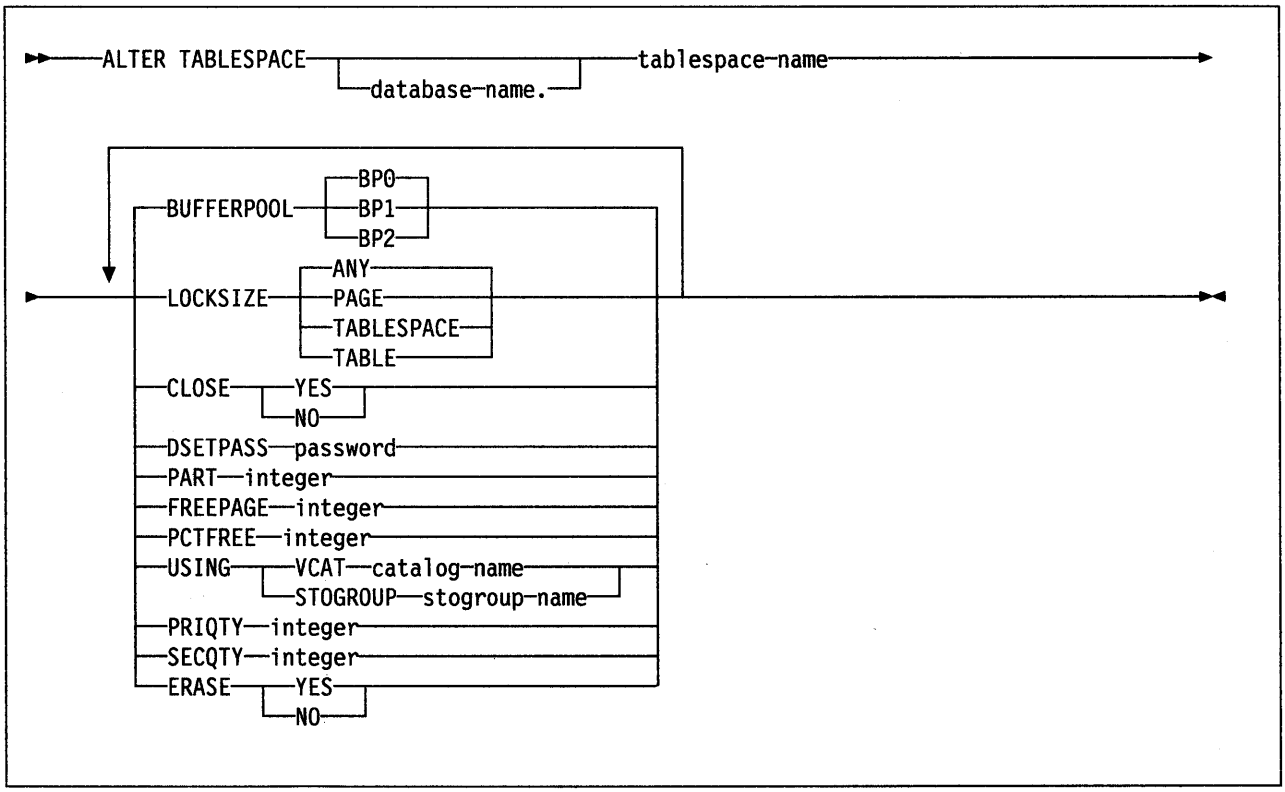
Authorization

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

- Ownership of the tablespace
- DBADM authority for its database
- SYSADM authority.

If BUFFERPOOL or USING STOGROUP is specified, additional privileges may be required as explained in the description of those clauses.

If the statement is embedded in a program, the privilege set is the privileges designated by the authorization ID of the owner of the plan. If the statement is dynamically prepared, the privilege set is the union of the privileges designated by each authorization ID of the process.



Syntax note: Use at least one of the keywords from the diagram above, but **do not use the same keyword more than once.**

Description

database-name.tablespace-name
Identifies the table space to be altered. You must identify a table space described in the DB2 catalog, but you must not identify a table space of the catalog. Omission of *database-name* is an implicit specification of DSNDB04.

If you identify a table space of DSNDB07, the database must be in the stopped state. If you identify a partitioned table space, you may have to use the PART clause as explained below.

BUFFERPOOL BP*n*
Names the buffer pool to be associated with the table space. Use BP0, BP1, or BP2. The buffer pool must be activated and and the privilege set must include SYSADM authority or the USE privilege for the buffer pool. The change to the description of the table space has no effect until the next time its data sets are opened. Do not use BUFFERPOOL if the page size of the table space is 32K bytes.

LOCKSIZE
Specifies the locking level for the table space. You must not use this clause for a table space in DSNDB07.

ANY
Specifies that DB2 may use any locking level. In most cases, DB2 will use page level locking. 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.

PAGE
Specifies page level locking.

TABLESPACE
Specifies table space level locking.

TABLE
Specifies table level locking. You must not specify TABLE for an unsegmented table space.

The change to the description of the table space has no effect on bound statements in existing application plans. The change applies to application plans that are created or rebound after the execution of the ALTER and to dynamic SQL statements that are prepared after the execution of the ALTER.

CLOSE
Indicates whether to close the data sets supporting the table space when there are no current users of the table space.

YES
Closes the data sets.

NO
Does not close the data sets.

DSETPASS *password*
Specifies the password that is passed to VSAM when the data sets of the table space are used by DB2. You must specify a VSAM master level password in the form of a short identifier. Use a delimited identifier for a password that contains any of the special characters acceptable to VSAM Access Method

Services. The change to the description of the table space has no effect until the next time its data sets are opened.

Changing the password for the table space does not change the password that protects its data sets. To change the data set password, use VSAM Access Method Services.

Note: The password does not apply to data sets managed by Storage Management Subsystem. Data set defined to SMS should be protected by RACF or some 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, or ERASE 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* may range from 0 to 255. FREEPAGE 0 leaves no free pages.

When you are dealing with segmented table spaces, 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 will be 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. Do not use this keyword with database DSNDB07.

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* % of free space is left on each page. *integer* may range from 0 to 99.

The change to the description of the table space or partition has no effect until it is loaded or reorganized. Do not use this keyword with database DSNDB07.

USING

Specifies whether a data set for the table space or partition is user-managed or DB2-managed. 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 may be used for the table space. (An unpartitioned table space can have more than one data set if PRIQTY + 123 * SECQTY is at least 2 gigabytes.) You must not specify USING for a table space in DSNDB07.

If you specify the USING clause, the table space must be in the stopped state when the ALTER TABLESPACE statement is executed. Successful execution of the statement changes the description of the table space, but has no immediate effect on the table space or partition. The new description is applied when the table space or partition is recovered, reorganized, or extended to a new volume or data set. However, if you change the catalog-name (explicitly by USING VCAT or implicitly by USING STOGROUP), you must move the data while the table space is in the stopped state as explained under "Notes" on page 120.

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 ICF catalog is longer than eight characters. When the new description of the table space is applied, the ICF catalog must contain an entry for the data set conforming to the DB2 naming conventions.

STOGROUP *stogroup-name*
Specifies a DB2-managed data set that resides on a volume of the identified storage group. You must identify a storage group described in the DB2 catalog and the privilege set must include SYSADM 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 ICF catalog used for the storage group must not contain an entry for the data set and, if the ICF 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 DB2-managed, omission of the PRIQTY clause is an implicit specification of the current PRIQTY value, omission of the SECQTY clause is an implicit specification of the current SECQTY value, and 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 user-managed:

- Omission of the PRIQTY clause is an implicit specification of PRIQTY 12 for a table space with 4K pages and PRIQTY 96 for a table space with 32K pages.
- Omission of the SECQTY clause is an implicit specification of SECQTY 12 for a table space with 4K pages and SECQTY 96 for a table space with 32K pages.
- Omission of the ERASE clause is an implicit specification of ERASE NO.

PRIQTY *integer*
Specifies the primary space allocation for a data set of the table space or partition. This clause must not be specified for user-managed data sets. Thus you may specify this clause only if:

- you also specify USING STOGROUP, or
- you do not specify the USING clause and the data set for the table space or partition is DB2-managed.

If you specify PRIQTY, the table space must be in the stopped state when the ALTER TABLESPACE statement is executed. Successful execution of the statement changes the description of the table space, but has no immediate effect on the table space or partition. The new description is applied when the table space or partition is recovered, reorganized, or extended to a new volume or data set. However, you may want to apply the new description before then by moving the data as explained under "Notes" on page 120.

You specify the primary space allocation in kilobytes. You may specify any integer, but 4194304 is used if the integer you specify is greater than 4194304. If you specify an integer less than 12 for a table space with a page size of 4K, the number used is 12. If you specify an integer less than 96 for a table space with a page size of 32K, the number used is 96.

DB2 specifies the primary space allocation to VSAM as an integral number of 4K pages. If the page size of the table space is 4K, this number is calculated by dividing the number of kilobytes by 4 and rounding up to the next highest integer. If the page size of the table space is 32K, this number is calculated by dividing the number of kilobytes by 32 and rounding up to the next highest integer. VSAM then uses this number to specify an integral number of tracks. Thus the primary space allocation can be greater than the number of kilobytes you specify.

SECQTY *integer*

Specifies the secondary space allocation for a data set of the table space or partition. This clause must not be specified for user-managed data sets. You may specify this clause only if:

- you also specify USING STOGROUP, or
- you do not specify the USING clause and the data set for the table space or partition is DB2-managed.

If you specify SECQTY, the table space must be in the stopped state when the ALTER TABLESPACE statement is executed. Successful execution of the statement changes the description of the table space, but has no immediate effect on the table space or partition. The new description is applied when the table space or partition is recovered, reorganized, or extended to a new volume or data set. However, you may want to apply the new description before then by moving the data as explained under "Notes" on page 120.

You specify the secondary space allocation in kilobytes. You may specify any integer but 131068 is used if the integer you specify is greater than 131068. If you specify 0 the data set cannot be extended.

DB2 specifies the secondary space allocation to VSAM as an integral number of 4K pages. If the page size of the table space is 4K, this number is calculated by dividing the number of kilobytes by 4 and rounding up to the next highest integer. If the page size of the table space is 32K, this number is calculated by dividing the number of kilobytes by 32 and rounding up to the next highest integer. VSAM then uses this number to specify an integral number of tracks. Thus the secondary space allocation can be greater than the number of kilobytes you specify.

ERASE

Specifies whether the contents of a data set for the table space or partition are erased when the table space is dropped. This clause must not be specified for user-managed data sets. You may specify this clause only if:

- you also specify USING STOGROUP, or
- you do not specify the USING clause and the data set for the table space or partition is DB2-managed.

If you specify ERASE, the table space must be in the stopped state when the ALTER TABLESPACE statement is executed. Successful execution of the statement changes the description of the table space but has no immediate effect on the table space or partition. The new description is applied when the table space or partition is recovered, reorganized, or extended to a new

volume or data set. However, you may want to apply the new description before then by moving the data as explained under "Notes" on page 120.

YES
Erases the contents of the data set when the table space is dropped.

NO
Does not erase the contents of the data set.

Notes

The ALTER TABLESPACE statement cannot be executed while a DB2 utility has control of the table space or the database is stopped.

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

USING, PRIQTY, SECQTY, and ERASE specify the storage attributes of a table space or partition. Changing these attributes requires changes to the description of the table space and movement of the data. The ALTER statement only changes the description of the table space. Movement of the data is a user responsibility which can be accomplished by the use of RECOVER, REORG, DSN1COPY, or a non-DB2 facility such as DFP/DFDSS.

Changing the storage attributes of a table space or partition involves the use of a STOP DATABASE command, an ALTER TABLESPACE statement, and a START DATABASE command, in that order. The usual procedure involves movement of the data while the table space is in the stopped state, but this is necessary only if the *catalog-name* is changed which is the case if the purpose of the procedure to to move the data to a different device. If the purpose is only to change the space allocation parameters or erase rule, the data does not have to be moved before the table space is restarted. In this case, the alterations will take effect when the table space or partition is subsequently recovered or reorganized.

If the catalog-name is implicitly or explicitly changed by the ALTER, the data must be moved while the table space is in the stopped state. If DSN1COPY or a non-DB2 facility is used, the move can be done either before or after the ALTER statement is executed. For example, DFP/DFDSS can be used to move the data and change the high level qualifier of the data set name before the ALTER statement is executed.

Any facility can be used to move the data while the table space is in the stopped state if the *catalog-name* has not been changed. If RECOVER or REORG is used, the ALTER statement must be executed first and then the table space started for utility use only. If another facility is used, the sequence is irrelevant, but the user is responsible for ensuring that the name of the data set conforms to the DB2 naming conventions when the table space is restarted.

Examples

Example 1: Alter table space DSN8S22E in database DSN8D22A. 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 DSN8D22A.DSN8S22E
  CLOSE NO
  DSETPASS OSESAME;
```

Example 2: Alter table space DSN8S22D in database DSN8D22A. BP2 is the buffer pool associated with the table space. PAGE is the level at which locking is to take place.

```
ALTER TABLESPACE DSN8D22A.DSN8S22D
  BUFFERPOOL BP2
  LOCKSIZE PAGE;
```

BEGIN DECLARE SECTION

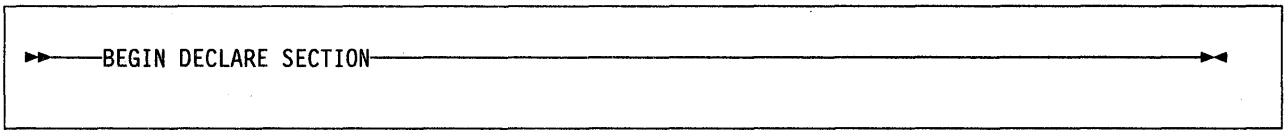
The BEGIN DECLARE SECTION statement marks the beginning of a host variable declare section.

Invocation

This statement can only be embedded in an application program. It is not an executable statement.

Authorization

None required.



Description

The BEGIN DECLARE SECTION statement may 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 on page 183.

The following rules are enforced by the precompiler only if the host language is C or the STDSQL(86) precompiler option is specified:

- A variable referenced 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.

Notes

Host variable declaration sections are only required if the STDSQL(86) option is specified or the host language is C. However, declare sections may be specified for any host language so that the source program can conform to the SAA definition of 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;
```

CLOSE

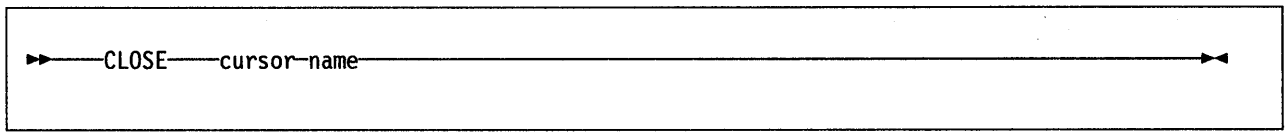
The CLOSE statement closes a cursor. If a temporary table was created when the cursor was opened, that table is destroyed.

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 165 for the authorization required to use a cursor.



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

The open cursors of an application process are implicitly closed at the termination of a unit of recovery. However, explicitly closing cursors as soon as possible can improve performance. CLOSE is not a commit or rollback operation.

Example

A cursor is used to fetch one row at a time into the 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 DSN8220.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 local DB2 catalog descriptions of tables, views, aliases, or columns.

Invocation

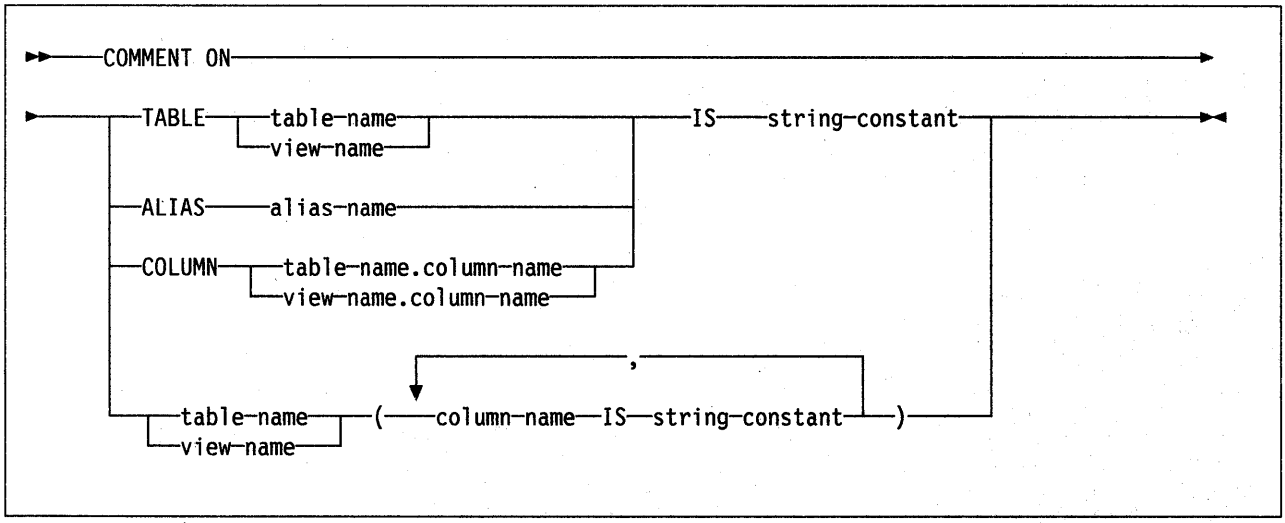
This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared.

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 authority.

If the statement is embedded in a program, the privilege set is the privileges designated by the authorization ID of the owner of the plan. If the statement is dynamically prepared, the privilege set is the union of the privileges designated by each authorization ID of the process.



Description

TABLE
Indicates that you want to comment on a table or view.

table-name or *view-name*
Must identify a table or view described in the local catalog. The comment is placed in the REMARKS column of the SYSIBM.SYSTABLES catalog table for the row that describes the table or view.

ALIAS
Indicates that you want to comment on an alias.

alias-name
Must identify an alias described in the local catalog. The comment is placed in the REMARKS column of the SYSIBM.SYSTABLES catalog table for the row that describes the alias.

COLUMN
Indicates that you want to comment on a column.

table-name.column-name or *view-name.column-name*
Is the name of the column, qualified by the name of the table or view in which it appears. The column must be described in the local catalog. The comment is placed into the REMARKS column of the SYSIBM.SYSCOLUMNS catalog table, for the row that describes the column.

To comment on more than one column in a table or view, do not use TABLE or COLUMN. Give the table or view name and then, in parentheses, a list of this 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 be described in the catalog.

IS
Introduces the comment you want to make.

string-constant
Can be any SQL character string constant of up to 254 characters.

Examples

Enter a comment on table DSN8220.EMP.

```
COMMENT ON TABLE DSN8220.EMP
  IS 'REFLECTS 1ST QTR 81 REORG'
```

Example 2: Enter a comment on view DSN8220.VDEPT.

```
COMMENT ON TABLE DSN8220.VDEPT
  IS 'VIEW OF TABLE DSN8220.DEPT'
```

Example 3: Enter a comment on the DEPTNO column of table DSN8220.DEPT.

```
COMMENT ON COLUMN DSN8220.DEPT.DEPTNO
  IS 'DEPARTMENT ID - UNIQUE'
```

Example 4: Enter comments on two columns in table DSN8220.DEPT.

```
COMMENT ON DSN8220.DEPT
  (MGRNO IS 'EMPLOYEE NUMBER OF DEPARTMENT MANAGER',
   ADMRDEPT IS 'DEPARTMENT NUMBER OF ADMINISTERING DEPARTMENT')
```

COMMIT

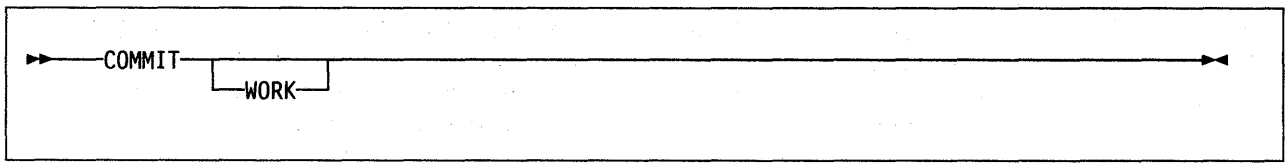
The COMMIT statement terminates a unit of recovery and commits the database changes that were made by that unit of recovery.

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.



Description

The unit of recovery in which the statement is executed is terminated and a new unit of recovery is initiated for the process. All changes made by ALTER, COMMENT ON, CREATE, DELETE, DROP, EXPLAIN, GRANT, INSERT, LABEL ON, REVOKE, and UPDATE statements executed during the unit of recovery are committed.

All locks implicitly acquired during the unit of recovery are released. See "LOCK TABLE" on page 213 for an explanation of the duration of explicitly acquired locks.

All cursors that were opened during the unit of recovery are closed. All statements that were prepared during the unit of recovery are destroyed and any cursors associated with the prepared statements are invalidated.

COMMIT WORK has the same effect as COMMIT.

Notes

COMMIT WORK, rather than COMMIT, should be used to conform to the SAA definition of SQL. However, neither form of the statement can be used in the IMS or CICS environment. To execute 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 initiated.

```
COMMIT WORK;
```

CREATE ALIAS

The CREATE ALIAS statement defines an alias for a table or view.

Invocation

This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared.

Authorization

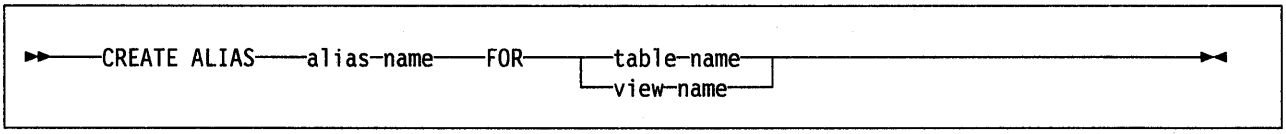
The privilege set defined below must include at least one of the following:

- The CREATEALIAS privilege
- SYSADM authority.

If the statement is embedded in a program, the privilege set is the privileges designated by the authorization ID of the owner of the plan. If the specified *alias-name* includes an *authorization-name* qualifier that is not the same as this authorization ID, the privilege set must include SYSADM authority.

If the statement is dynamically prepared, the privilege set is the privileges designated by the SQL authorization ID of the process. However, if the specified *alias-name* includes an *authorization-name* qualifier that is not the same as this authorization ID, the following rules apply:

- Any *authorization-name* qualifier is valid if the privilege set includes SYSADM authority.
- If the privilege set does not include SYSADM authority, the *authorization-name* qualifier is valid only if it is the same as one of the authorization IDs of the process, and the privilege set designated by that authorization ID includes the CREATEALIAS privilege. This is an exception to the rule that the privilege set is the privileges designated by the SQL authorization ID.



Description

alias-name

Is the name of the alias. The name supplied, including the implicit or explicit qualifiers, must not identify a table, view, alias, or synonym already desribed in the catalog.

If qualified, the name can be a two-part or three-part name. If it is a three-part name, the first qualifier must be the location-name of the local DB2 subsystem. In either case, 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 a program, the owner of the plan is the owner of the alias. 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 may be local, remote, or non-existent. The *table-name* or *view-name* must not be the same as the *alias-name* or the name of an alias described in the catalog.

Notes

If an alias is defined for a remote table or view, the existence of the table or view is not verified. A warning occurs if an alias is defined for a local table or view that does not exist.

Example

Create an alias for a remote catalog table:

```
CREATE ALIAS LATABLES FOR DB2USCALABOA5281.SYSIBM.SYSTABLES
```

CREATE DATABASE

The CREATE DATABASE statement defines a local database in which table spaces and index spaces may later be created.

Invocation

This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared.

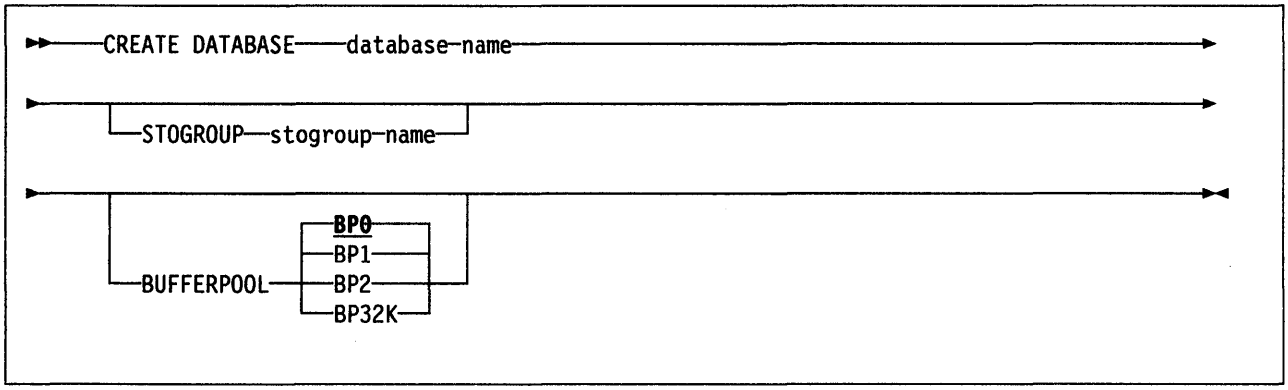
Authorization

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

- The CREATEDBA privilege
- The CREATEDBC privilege
- SYSADM authority.

If the statement is embedded in a program, the privilege set is the privileges designated by the authorization ID of the owner of the plan. If the statement is dynamically prepared, the privilege set is the privileges designated by the SQL authorization ID of the process.

See the Notes below for the authorization effect of a successful CREATE DATABASE statement.



Description

database-name
Names the database. The name you supply must not be the name of a database already described in the DB2 catalog, nor can it start with DSNDB.

STOGROUP *stogroup-name*
Names the storage group that will 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**.

BUFFERPOOL BP*n*
Names the buffer pool to be used, as required, as a default buffer pool for table spaces and indexes within the database. Use BP0, BP1, BP2, or BP32K.

BP32K applies only to table spaces. If BP32K is specified, the default buffer pool for indexes in the database is BP0.

The default is **BP0**.

Notes

If the statement is embedded in an application program, the owner of the plan 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 privilege, the owner acquires DBADM authority for the database.

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 DSN8D22P. DSN8G220 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 DSN8D22P
  STOGROUP DSN8G220
  BUFFERPOOL BP2;
```

Example 2: Create database DSN8TEMP. Use the DB2 default storage group and buffer pool.

```
CREATE DATABASE DSN8TEMP;
```

CREATE INDEX

The CREATE INDEX statement creates an index on a local 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 privilege set must include at least one of the following:

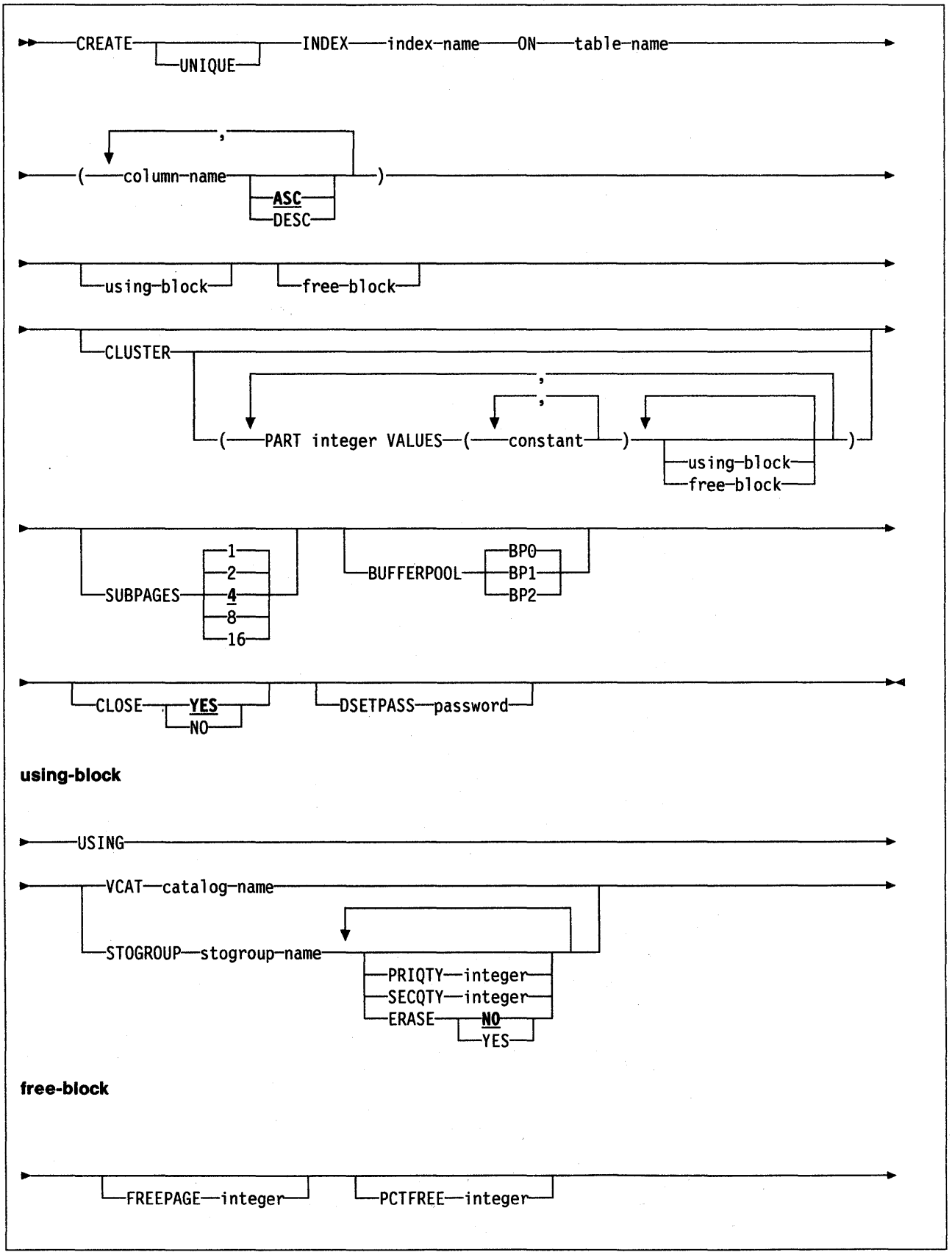
- The INDEX privilege on the table
- Ownership of the table
- DBADM authority for the database
- SYSADM authority.

If BUFFERPOOL or USING STOGROUP is specified, additional privileges may be required as explained in the description of those clauses.

If the statement is embedded in a program, the privilege set is the privileges designated by the authorization ID of the owner of the plan. If the specified index name includes a qualifier that is not the same as this authorization ID, the privilege set must include SYSADM authority, DBADM authority for the database, or DBCTRL authority for the database.

If the statement is dynamically prepared, the privilege set is the privileges designated 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:

- Any qualifier is valid if the privilege set includes SYSADM authority, DBADM authority for the database, or DBCTRL authority for the database.
- If the privilege set does not include any 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 designated by that authorization ID includes the INDEX privilege on the identified table. This is an exception to the rule that the privilege set is the privileges designated by the SQL authorization ID.


CREATE
UNIQUE
INDEX
index-name
ON
table-name
(
column-name
,
ASC
DESC
)
using-block
free-block
CLUSTER
(
PART integer VALUES
(
constant
,
)
,
using-block
free-block
)
SUBPAGES
1
2
4
8
16
BUFFERPOOL
BP0
BP1
BP2
CLOSE
YES
NO
DSETPASS
password
using-block
USING
VCAT
catalog-name
STOGROUP
stogroup-name
PRIQTY
integer
SECQTY
integer
ERASE
NO
YES
free-block
FREEPAGE
integer
PCTFREE
integer

Description

UNIQUE
Prevents the table from containing two or more rows with the same value of the index key. The constraint is enforced when rows of the table are updated or new rows are inserted.

The 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.

When UNIQUE is used, null values are treated as any other values. For example, if the key is a single column that may contain null values, that column may contain no more than one null value.

If the definition of the identified table is incomplete because it does not have a primary index, the creation of the index completes the definition of the table if UNIQUE is specified and the index key is identical to the primary key.

A primary key and and index key are identical only if they have the same columns in the same order. The description of an index that completes the definition of a table indicates that it is a primary index.

INDEX *index-name*
Names the index. The name you supply, including the implicit or explicit qualifier, must not identify an index already described in the catalog.

If the data sets for the index are defined by DB2, the index name is used to derive a unique name for the index space that is created for the index. If the data sets for the index are not defined by DB2, the name of the index space is the same as the second (or only) part of the index-name. If this identifier consists of more than 8 characters, only the first 8 characters are used as the name of the index space. The name of the index space must not be the same as the name of an index space or table space of the database that contains the identified table.

If the index name is qualified, the qualifier is the owner of the index. If the index name is unqualified and the statement is embedded in a program, the owner of the plan is the owner of the index. If the index name is unqualified and the statement is dynamically prepared, the SQL authorization ID is the owner of the index. The owner has the privilege of altering and dropping the index.

ON *table-name*
Names the table on which you want the index to be created. The *table-name* must name a base table (not a view) described in the catalog. It must *not* name a catalog table. Although a three-part name may be used, the table must be local.

The table space that contains the named table must be available to DB2 so that its data sets can be opened.

column-name
Names a column that is to be part of the index key.

Each *column-name* identifies a column of the table. Do not specify more than 16 columns, or the same column more than once. Do not qualify the *column-name*. For a partitioned index, do not identify a nullable column of a foreign key that has a delete rule of SET NULL.

The sum of the length attributes of the columns must not be greater than *m-n*, where:

- *n* is the number of columns that may contain null values;
- *m* depends on whether the index is partitioned, on the number of subpages, and on whether UNIQUE is used, according to the following table:

Partitioned	SUBPAGES	UNIQUE	m
No	<8	-	254
No	8	Yes	241
No	8	No	238
No	16	Yes	114
No	16	No	111
Yes	-	-	40

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 unpartitioned indexes and then for partitioned indexes. If you omit USING, the default storage group of the database must exist.

USING Clause for Unpartitioned Indexes

For unpartitioned indexes, the USING clause indicates whether the data sets for the index are defined by you or by DB2. If DB2 is to define the data sets, the clause also gives space allocation parameters and an erase rule.

If you omit USING, DB2 defines the data sets in the default storage group of the database, using implicit specifications of PRIQTY 3, SECQTY 3, and ERASE NO.

VCAT *catalog-name*
Defines the data sets for the index.

The ICF catalog named by *catalog-name* must already contain an entry for the first data set of the index, conforming to the DB2 naming convention for data sets. If the name of the ICF catalog is longer than 8 characters, you must use an alias.

STOGROUP *stogroup-name*
Defines the data sets for the index, using space from the named storage group. If PRIQTY + 123 * SECQTY is 2 gigabytes[4] or greater, more than one data set may eventually be used, but only the first is defined during execution of this statement.

stogroup-name must name a storage group described in the catalog, and SYSADM authority, or the USE privilege for that storage group, is required.

[4] 1 gigabyte is 2^{30} bytes.

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.

The ICF catalog used for the storage group must NOT contain an entry for the first data set of the index. If the ICF catalog is password protected, the description of the storage group must include a valid password.

PRIQTY *integer*
Indicates the minimum primary space allocation for the DB2-defined data sets. *integer* is a number of kilobytes not greater than 4194304. That maximum is used if you specify any larger value.

DB2 asks VSAM for an integral number, 3 or more, of pages. (Two pages of the primary space are used by DB2 for purposes other than storing index entries.) The actual number of pages requested is given by these rules:

- If you do not use PRIQTY, the number is 3.
- If you do use PRIQTY, the number is *integer*/4 rounded up to the next highest integer, but not less than 3.

The amount of space allocated is usually greater than that requested. DB2 requests a number of records, which VSAM changes to a request for tracks. To more closely estimate the actual amount, see a description of the DEFINE CLUSTER command in one of the VSAM Access Method Services publications.

Syntax note: Do not use PRIQTY more than once in the same using-block.

SECQTY *integer*
Indicates the minimum secondary space allocation for the DB2-defined data sets. *integer* is a number of kilobytes not greater than 131068. The maximum is used if you specify any larger value.

DB2 requests from VSAM a number of pages given by these rules:

- If you do not use SECQTY, the number is 3.
- If *integer* is 0, no secondary space allocation is used.
- If you do use SECQTY, the number is *integer*/4 rounded up to the next highest integer, but not more than 32767.

The amount of space allocated is usually greater than that requested. DB2 requests a number of records, which VSAM changes to a request for tracks. To more closely estimate the actual amount, see a description of the DEFINE CLUSTER command in one of the VSAM Access Method Services publications.

Syntax note: Do not use SECQTY more than once in the same using-block.

ERASE
Specifies whether the DB2-defined data sets are to be erased (filled with 0's) when the index is dropped.

NO
Does not erase the data sets. This is the default.

YES
Erases the data sets.

Syntax note: Do not use ERASE more than once in the same using-block.

USING Clause for Partitioned Indexes:

If the index 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 an unpartitioned index.

The USING clause that governs 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
- A default USING STOGROUP clause that specifies the default storage group for the database

Do not use more than one using-block in any PART clause.

VCAT *catalog-name*
Defines the data sets for the index.

If *n* is the number of the partition, the ICF catalog named by *catalog-name* must already contain an entry for the *n*th data set of the index, conforming to the DB2 naming convention for data sets.

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 ICF catalog used for the storage group must NOT contain an entry for the *n*th data set of the index.

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.

free-block

FREEPAGE *integer*
Specifies how often to leave a page of free space when index entries are created by a load operation or the index is reorganized. One free page is left after every *integer* pages; *integer* may range from 0 to 255.

The default is **FREEPAGE 0**, leaving no free pages.

Do not use FREEPAGE more than once in any free-block.

PCTFREE *integer*
Specifies what percentage of each page to leave as free space when index entries are created by a load operation or the index is reorganized. *integer* may range from 0 to 99.

The first entry in a subpage or nonleaf page is loaded without restriction. When additional entries are loaded, at least *n* % of free space is left on each nonleaf page, and at least *n*/*m* on each subpage (where *n* is the value of *integer* and *m* is the number of subpages).

The default is **PCTFREE 10**.

Do not use PCTFREE more than once in any free-block.

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.

CLUSTER

Makes the index a cluster index. Do not use CLUSTER if the table already has a cluster index. If you do not use CLUSTER, the index is not a cluster index.

PART *integer*

A PART clause tells what is 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. If there are *p* partitions, the value of *integer* must range from 1 through *p*.

If you do not use CLUSTER, the index is not a cluster index; do not use the PART clause.

VALUES *(constant, ... , constant)*

You must use at least one constant after VALUES in each PART clause. You may 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.
- 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.
- The highest value of the key in the last partition is always the highest possible value of the key, no matter which constants you use after VALUES.

Syntax note: After the list of values you may use a using-block, a free-block, or both. But do not use more than one using-block or more than one free-block in any PART clause.

SUBPAGES *integer*
Gives the number of subpages for each physical page. (The subpage is the unit of index locking.) Use 1, 2, 4, 8, or 16. The default is **4**.

BUFFERPOOL BP*n*
Names the buffer pool to be associated with the index. Use BP0, BP1, or BP2. The buffer pool must be activated, and SYSADM authority, or the USE privilege for the buffer pool, is required.

The default is the default buffer pool of the database (BP0 if the default is BP32K).

CLOSE
Indicates whether to close the data sets supporting the index when there are no current users of the index.

YES
Closes the data sets. This is the default.

NO
Does not close the data sets.

DSETPASS *password*
Gives a password that is passed to VSAM when the data sets are used by DB2. *password* is a VSAM master level password in the form of a short, ordinary identifier. If the password is a delimited identifier, it may contain any special characters acceptable to VSAM Access Method Services. If DSETPASS is omitted, a password is not passed to VSAM.

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 don't use a storage group, you define the password that protects the data sets through VSAM 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.

Note: The password does not apply to data sets managed by Storage Management Subsystem. Data set defined to SMS should be protected by RACF or some similar external security system.

Notes

The CREATE INDEX statement cannot be executed while a DB2 utility has control of the table space that contains the identified table.

If the named table already contains data, 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 primary 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 primary key.

Two DB2 subsystems can be cataloged on the same ICF catalog. But two DB2 subsystems must not share the same ICF catalog alias because this is the only parameter that makes the data set unique. Take care to ensure that the VCAT

names specified for a user-defined data set will identify a data set unique to this DB2 subsystem.

Examples

Example 1: Create a unique index, named XDEPT1, on table DSN8220.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 DSN8G220; the data sets need not be erased if the index is dropped.

Use 8 subpages for each physical page, and associates 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 UNIQUE INDEX DSN8220.XDEPT1
  ON DSN8220.DEPT
    (DEPTNO ASC)
  USING STOGROUP DSN8G220
    ERASE NO
  SUBPAGES 8
  BUFFERPOOL BP1
  CLOSE YES
  DSETPASS OSESAME;
```

Example 2: Create an index named XEMP2 on table EMP in database DSN8220.. Put the entries in ascending order by column EMPNO. Let DB2 define the data sets for the index.

The index is to be a cluster index. DB2 will define the data sets for each partition using storage group DSN8G220. The primary space allocation is 36 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 follow:

Partition 1: entries up to H99.
Partition 2: I00 to P99
Partition 3: Q00 to Z99
Partition 4: entries above Z99.

Use 8 subpages for each physical page, and associate the index with bufferpool 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 DSN8220.XEMP2
  ON DSN8220.EMP
    (EMPNO ASC)
  USING STOGROUP DSN8G220
    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;
```

CREATE STOGROUP

The CREATE STOGROUP statement creates a local storage group. Storage from the identified volumes may subsequently be allocated for table spaces and index spaces.

Invocation

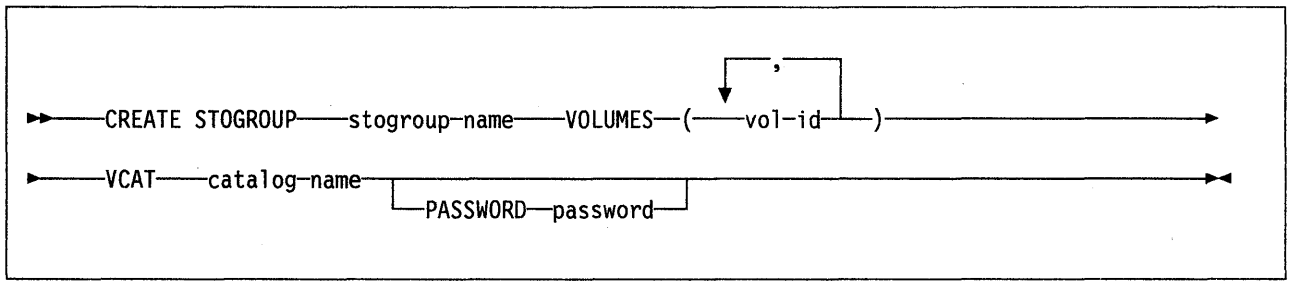
This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared.

Authorization

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

- the CREATESG privilege
- SYSADM authority.

If the statement is embedded in a program, the privilege set is the privileges designated by the authorization ID of the owner of the plan. If the statement is dynamically prepared, the privilege set is the privileges designated by the SQL authorization ID of the process.



Description

stogroup-name
Is the name you want to assign to this storage group. You must not name a storage group already described in the DB2 catalog.

VOLUMES (*vol-id*)
Gives the volume serial number of one or more OS/VS storage volumes. A *vol-id* is a maximum of 6 characters specified as an identifier or a string constant. The same volume serial number must not appear more than once.

VCAT *catalog-name*
Gives the name or alias of an ICF catalog. If the name of the ICF catalog is longer than 8 characters, you must use an alias.

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 may contain any special characters acceptable to VSAM access method services. The password will be used to access the ICF catalog. The password that protects the catalog must be established by the installation by use of VSAM Access Method Services. If this clause is not specified, no password will be used by DB2 to access the ICF catalog.

Notes

When the storage group is used, an error occurs if the volumes are not of the same device type or are not available to MVS for dynamic allocation of data sets.

There is no specific limit on the number of volumes that can be defined for a storage group. However, the maximum number of volumes used for allocation is 133. Thus there is no point in creating a storage group with more than 133 volumes.

If the statement is embedded in a program, the owner of the plan is the owner of the storage group. If the statement is dynamically prepared, the SQL authorization ID is the owner of the storage group. The owner has the privilege of altering and dropping the storage group.

DB2 acts solely as a conduit for the specified volumes. As such, it passes the list of volumes to DFP for initial dataset definitions and for dataset extensions. All valid volume serial parameters, acceptable to DFP (AMS and VSAM), are acceptable in this statement.

Example

Create storage group, DSN8G220, of volumes ABC005 and DEF008. DSNCAT is the ICF catalog name, and OSESAME is the VSAM password.

```
CREATE STOGROUP DSN8G220
  VOLUMES (ABC005,DEF008)
  VCAT DSNCAT
  PASSWORD OSESAME;
```

CREATE SYNONYM

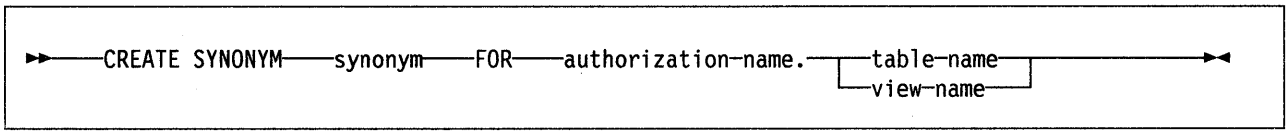
The CREATE SYNONYM statement defines a synonym for a table or view.

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.



Description

synonym
Is the synonym for the table or view. It must be an SQL identifier that is not identical to a synonym owned by the authorization ID of the statement. Nor can it be identical to the unqualified name of a table, view, or alias that this authorization ID owns. If the statement is embedded in a program, the authorization ID of the statement is the owner of the plan. If the statement is dynamically prepared, the authorization ID is the SQL authorization ID.

FOR
Identifies the object to which the synonym will apply. The name must consist of exactly two parts and must denote a local table, view, or alias already described in the catalog. If it denotes a table or view, the object identified is this table or view. If it denotes an alias, the alias must represent a local table or view, and the object identified is then this local table or view.

Notes

If the statement is embedded in a program, the owner of the plan is the owner of the synonym. If the statement is dynamically prepared, the SQL authorization ID is the owner of the synonym. Only the owner has the privilege of dropping the synonym. Furthermore, the synonym is only defined for its owner. However, in cases where the statement is dynamically prepared, users with SYSADM authority can create synonyms for other users. This can be done by changing the value of the CURRENT SQLID special register before issuing the CREATE SYNONYM statement. See "SET CURRENT SQLID" on page 238 for details on changing the value of the CURRENT SQLID special register.

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. This severs the connection between synonym and alias: Even if the alias were dropped and redefined, the synonym would still be in effect and would designate the original table or view.

Example

Define an alternative name, DEPT, for table DSN8220.DEPT.

```
CREATE SYNONYM DEPT
  FOR DSN8220.DEPT;
```

CREATE TABLE

The CREATE TABLE statement defines a local table. The definition must include its name and the names and attributes of its columns. The definition may also include other attributes of the table, such as its primary key and tablespace.

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:

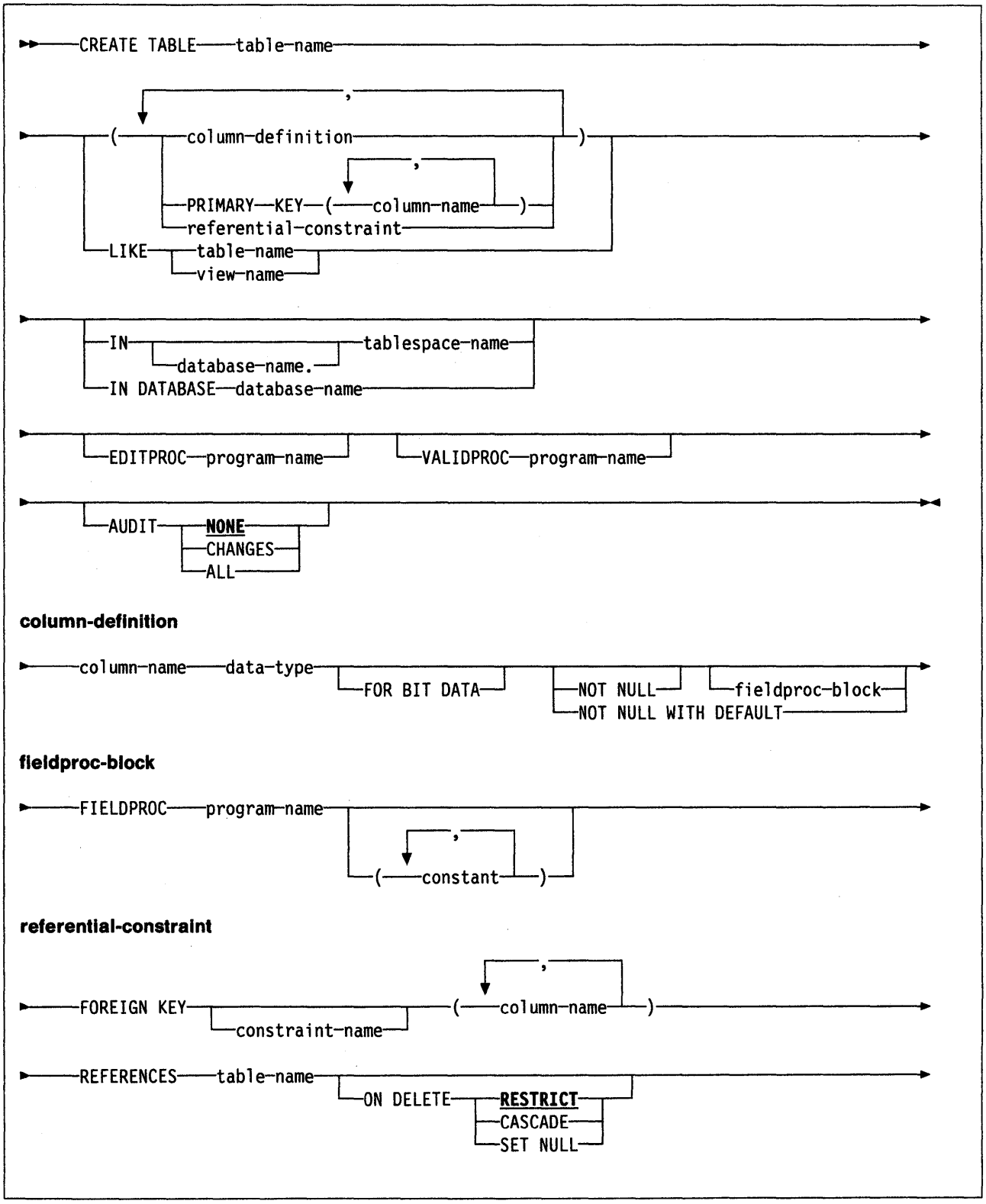
- The CREATETAB privilege for the database that is implicitly or explicitly specified by the IN clause
- DBADM, DBCTRL, or DBMAINT authority for the database
- SYSADM authority.

If IN, LIKE or FOREIGN KEY is specified, additional privileges may be required as explained in the description of those clauses.

If the statement is embedded in a program, the privilege set is the privileges designated by the authorization ID of the owner of the plan. If the specified table name includes a qualifier that is not the same as this authorization ID, the privilege set must include SYSADM authority, DBADM authority for the database, or DBCTRL authority for the database.

If the statement is dynamically prepared, the privilege set is the privileges designated 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:

- Any qualifier is valid if the privilege set includes SYSADM authority, DBADM authority for the database, or DBCTRL authority for the database.
- If the privilege set does not include any 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 designated by that authorization ID includes all privileges needed to create the table. This is an exception to the rule that the privilege set is the privileges designated by the SQL authorization ID.


CREATE TABLE
table-name
(
column-definition
,
)
PRIMARY KEY (
column-name
,
)
referential-constraint
LIKE
table-name
view-name
IN
database-name.
tablespace-name
IN DATABASE
database-name
EDITPROC
program-name
VALIDPROC
program-name
AUDIT
NONE
CHANGES
ALL
column-definition
column-name
data-type
FOR BIT DATA
NOT NULL
NOT NULL WITH DEFAULT
fieldproc-block
fieldproc-block
FIELDPROC
program-name
(
constant
,
)
referential-constraint
FOREIGN KEY
constraint-name
(
column-name
,
)
REFERENCES
table-name
ON DELETE
RESTRICT
CASCADE
SET NULL

Description

table-name
Is the name of the table. The name supplied, including the implicit or explicit qualifiers, must not identify a table, view, alias, or synonym already described in the catalog.

If qualified, the name can be a two-part or three-part name. If it is a three-part name, the first qualifier must be the location-name of the local DB2 subsystem. In either case, 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 plan is the owner of the table. 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

column-name
Is the name of 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.

data-type
Is one of the types in the following list. Use:

INTEGER or **INT**
For a large integer.

SMALLINT
For a small integer.

FLOAT(*integer***)**
For a floating-point number. If the integer is between 1 and 21 inclusive, the format is that of single precision floating-point. If the integer is between 22 and 53 inclusive, the format is that of double precision floating-point.

You may also specify:

REAL	for single 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; it may range from 1 to 15. The second integer is the scale of the number; that is, the number of digits to the right of the decimal point; it may range from 0 to the precision. You may also specify:

DECIMAL(*integer***)**	for DECIMAL(*integer,0*)
DECIMAL	for DECIMAL(*5,0*)

NUMERIC(*integer, integer***)**
NUMERIC is an alternate name for DECIMAL: Anywhere that DECIMAL can appear, NUMERIC can replace it. For example, NUMERIC(8) is equivalent to DECIMAL(8). Unlike DECIMAL, however, NUMERIC has no allowable abbreviation.

CHARACTER(*integer*) or **CHAR(*integer*)**
For a fixed-length character string of length *integer*, which may range from 1 to 254. If the length specification is omitted, a length of 1 character is assumed.

VARCHAR(*integer*)
For a varying-length character string of maximum length *integer*, which may range from 1 to the maximum row size. An integer greater than 254 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 to calculate the maximum length, see "Notes" on page 151. If the maximum length is greater than 254, the column is a long string column.

GRAPHIC (*integer*)
For a fixed-length graphic string of length *integer*, which may 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. 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 to calculate the maximum length, see "Notes" on page 151.

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.

FOR BIT DATA
Specifies that the contents of the column are to be treated as bit (binary) data for exchange with other systems. This option only applies to CHAR, VARCHAR, and LONG VARCHAR columns. If this option is not present, the character column is assumed to contain EBCDIC characters.

NOT NULL
Prevents the column from containing null values.

NOT NULL WITH DEFAULT
Prevents the column from containing null values, and allows a default value other than the null value. The default value used 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	the current date
Time	the current time
Timestamp	the current timestamp

If you use neither NOT NULL nor NOT NULL WITH DEFAULT, the column may contain null values, and its default value is the null value.

fieldproc-block

FIELDPROC *program-name*
Identifies *program-name* as the field procedure exit routine for the column. Writing a field procedure exit routine is described in an appendix of *Administration Guide*. Field procedures can only be used for short string columns. They cannot be used for columns with the NOT NULL WITH DEFAULT attribute.

A field procedure receives a single string value, and may transform that value (encode it) in any way. Also, it receives an encoded value and must decode it back to the original string value. A field procedure might typically be used to alter the sorting sequence of values entered in a column.

constant
Is a parameter passed to the field procedure when the CREATE TABLE statement invokes it. A parameter list is optional; the number of parameters and the data type of each are determined by the 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.

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 or NOT NULL WITH DEFAULT. Each *column-name* must identify a column of the table and the same column must not be identified more than once. The number of identified columns must not exceed 16 and the sum of their length attributes must not exceed 254.

The description of the table as recorded in the catalog includes the primary key and an indication that the definition of the table is incomplete until its primary index is created.

referential-constraint

FOREIGN KEY (*column-name***) REFERENCES table-name**
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 *constraint-name* is specified, it must not be the same as the name of a referential constraint that was established by a previously specified FOREIGN KEY clause of the CREATE TABLE statement.

The foreign key of the referential constraint is composed of the identified columns. Each name in the column list must identify a column of the table and the same column must not be identified more than once. The number of identified columns must not exceed 16, and the sum of their length attributes must not exceed 254 minus the number of columns that allow null values. If the same list of column names is specified in more than one FOREIGN KEY clause, those clauses must not identify the same table.

The *table-name* specified in a FOREIGN KEY clause must identify a table that is described in the catalog, but must not identify a catalog table. In the following discussion, let T2 denote an identified table and let T1 denote the table being created (T1 and T2 cannot be the same table.).

T2 must have a primary index and the privilege set must include the ALTER privilege on T2.

The specified foreign key must have the same number of columns as the primary key of T2 and, except for their names, default values, and null attributes, the description of the nth column of the foreign key must be identical to the description of the nth column of that primary key. If a column of the foreign key has a field procedure, the corresponding column of the primary key must have the same field procedure and an identical field description.

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.

The delete rule of the relationship is determined by the ON DELETE clause. SET NULL must not be specified unless some column of the foreign key allows null values. Omission of the clause is an implicit specification of ON DELETE RESTRICT.

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 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 descendant of T3 and the deletion of rows from T3 cascades to T2
- T2 and T3 are both descendants of the same table and the deletion of rows from that table cascades to both T2 and T3.

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 described in the catalog 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 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 the catalog description of view columns do not include field procedures.

The implicit definition does not include any other attributes of the identified table or view. Thus, the new table does not have a primary or foreign key. The table is created in the table space implicitly or explicitly specified by the IN clause, and it has an edit or validation procedure only if the EDITPROC or VALIDPROC clauses are specified.

IN *database-name.tablespace-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 may name a database (with *database-name*), a table space (with *tablespace-name*), or both. If you name a database, it must be described in the DB2 catalog, and must not be DSNDB06 or DSNDB07.

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 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 authority, DBADM authority for the database, or the USE privilege for the table space.

EDITPROC *program-name*
Names an edit routine for the table. The program, which must be provided by your 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 may transform that row in any way. Also, it receives a transformed row and must change it back to its original form. A typical use is to compress the storage representation of rows to save space on DASD.

For information on writing an EDITPROC exit routine, see Appendixes of *Administration Guide*.

If you omit EDITPROC, the table has no edit procedure.

VALIDPROC *program-name*
Names a validation routine for the table. The program, which must be provided by your site, is invoked during the execution of LOAD, INSERT, UPDATE and DELETE operations on the table. The program receives a row and returns a value that indicates whether the operation should proceed for that row. A typical use is to impose constraints on the values of rows.

For more information on writing a VALIDPROC exit routine, see Appendixes of *Administration Guide*.

If you omit VALIDPROC, the table has no validation procedure.

AUDIT
Identifies the types of access to this table that will cause auditing to be performed.

NONE
Specifies that no auditing is to be done when this table is accessed. AUDIT NONE 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 recovery. However, the auditing is done only if the appropriate trace class is active. For information about trace classes, see Section 6 (Volume 2) of *Administration Guide*.

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 recovery of a utility or application process. However, the auditing is done only if the appropriate trace class is active and the access is performed through the data manager. (COPY, RECOVER, REPAIR, and the stand-alone utilities do not use the data manager.)

Notes

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 tablespace was created.

Maximum Record Size, in bytes:

EDITPROC	Page Size = 4K	Page Size = 32K
NO	4056	32714
YES	4046	32704

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 list that follows 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 list.

Data type	Byte count
INTEGER	4
SMALLINT	2
FLOAT(*n*)	If *n* is between 1 and 21, the byte count is 4. If *n* is between 22 and 53, the byte count is 8.
DECIMAL	INTEGER(*p*/2) + 1, where *p* is the precision.
CHAR(*n*)	*n*
VARCHAR(*n*)	*n* + 2
LONG VARCHAR	See "Byte Count of a LONG column," below.
GRAPHIC(*n*)	2*n*
VARGRAPHIC(*n*)	2*n* + 2
LONG VARGRAPHIC	See "Byte Count of a LONG column."
DATE	4
TIME	3
TIMESTAMP	10

Byte Count of a LONG column: To calculate the count, let:

m be the maximum row size (6 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*(INTEGER((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." Add one if the column allows null values.
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.

The successful execution of a CREATE TABLE statement is audited if the statement includes AUDIT CHANGES or AUDIT ALL.

Examples

Example 1: Create DSN8220.DEPT. DEPTNO, DEPTNAME, MGRNO, and ADMRDEPT are column names. CHAR means the column will contain character data. NOT NULL means that the column cannot contain a null value. VARCHAR means the column will contain varying-length character data. DEPTNO is the name of the column that comprises the primary key. The table is to be in database DSN8D22A and table space DSN8S22D.

```
CREATE TABLE DSN8220.DEPT
  (DEPTNO   CHAR(3)     NOT NULL,
   DEPTNAME VARCHAR(36) NOT NULL,
   MGRNO    CHAR(6),
   ADMRDEPT CHAR(3)     NOT NULL,
   PRIMARY KEY(DEPTNO))
  IN DSN8D22A.DSN8S22D;
```

Example 2: Create DSN8220.PROJ. PROJNO, PROJNAME, DEPTNO, RESPEMP, PRSTAFF, PRSTDATE, PRENDATE, and MAJPROJ are column names. CHAR means the column will contain character data. DECIMAL means the column will contain packed decimal data. 5,2 means the following: 5 indicates the number of decimal digits, and 2 indicates the number of digits to the right of the decimal point. NOT NULL means that the column cannot contain a null value. VARCHAR means the column will contain varying-length character data. DSN8EAPR is the name of the validation procedure for the table. The table is in an implicitly created table space of database DSN8D22A.

```
CREATE TABLE DSN8220.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 DSN8D22A
  VALIDPROC DSN8EAPR;
```

CREATE TABLESPACE

The CREATE TABLESPACE statement defines a simple, segmented, or partitioned tablespace in a local database.

Invocation

This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared.

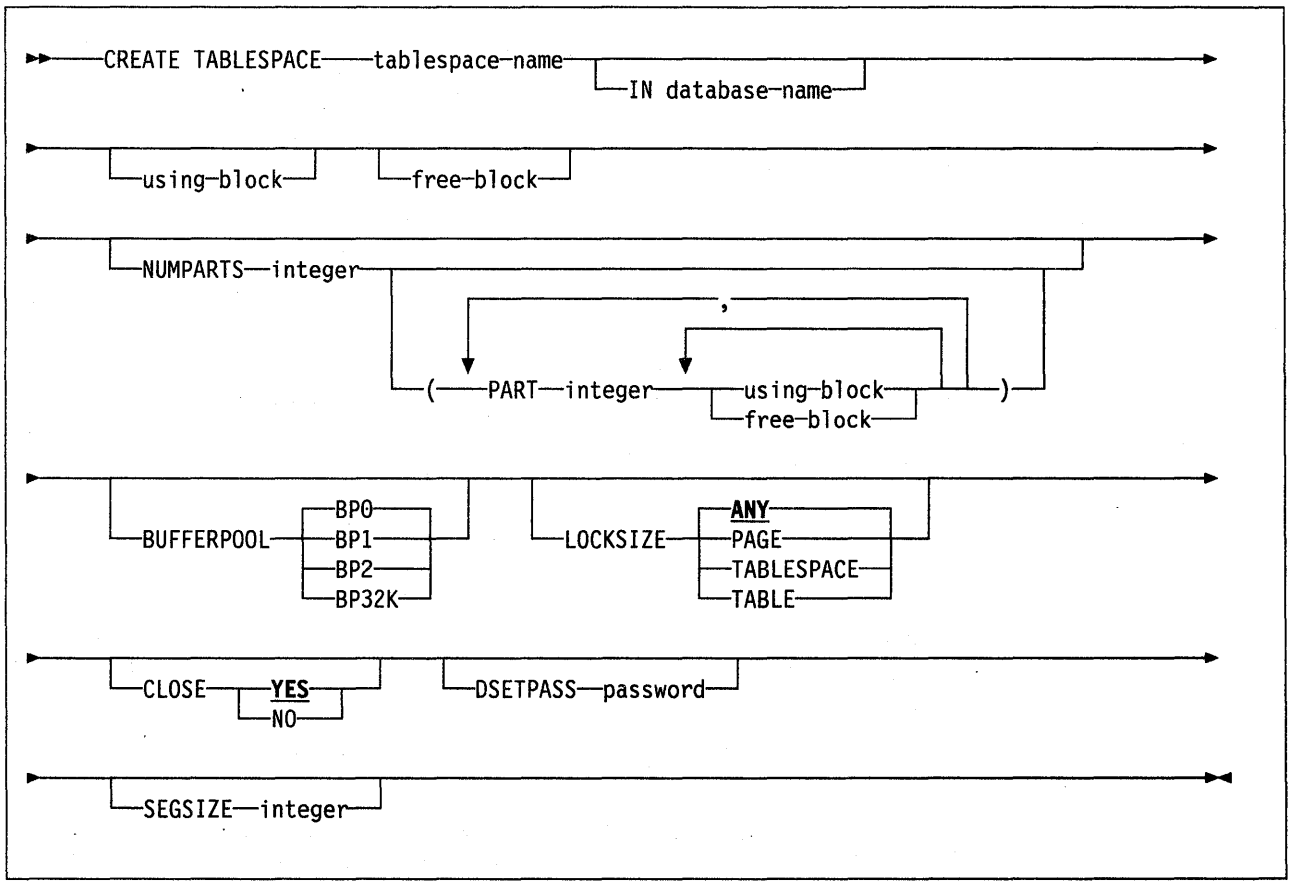
Authorization

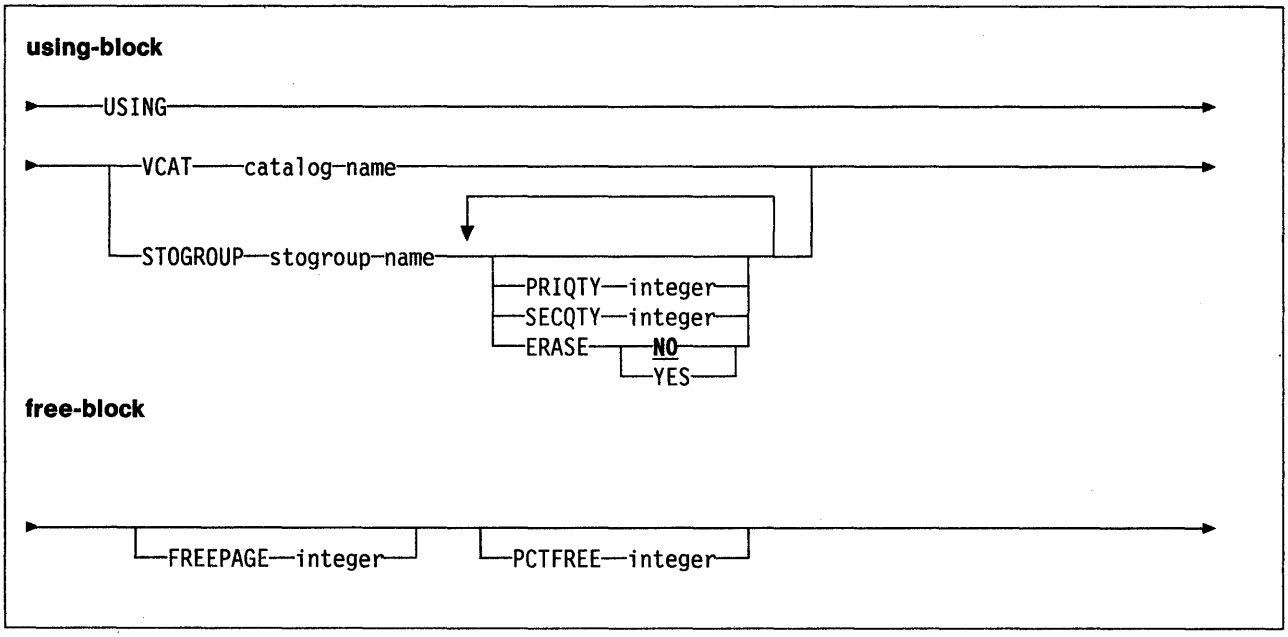
The privilege set defined below must include at least one of the following:

- The CREATETS privilege for the database.
- DBADM, DBCTRL, or DBMAINT authority for the database.
- SYSADM authority.

If BUFFERPOOL or USING is specified, additional privileges may be required as explained in the description of those clauses.

If the statement is embedded in a program, the privilege set is the privileges designated by the authorization ID of the owner of the plan. If the statement is dynamically prepared, the privilege set is the privileges designated by the SQL authorization ID of the process.





Description

tablespace-name
Is the name you want to give the table space. It must not be the name of a tablespace or indexspace already in the database implicitly or explicitly specified by the IN clause.

IN *database-name*
Names the database in which you want to create the table space. *database-name* must name a database described in the DB2 catalog, and must not be DSNDB06. If DSNDB07 is named, it must be in the stopped state. The default is **DSNDB04**.

using-block

The components of the USING clause are discussed below, first for unpartitioned table spaces and then for partitioned table spaces. If you omit USING, the default storage group of the database must exist.

USING Clause for Unpartitioned Table Spaces:

For unpartitioned table spaces, the USING clause tells 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.

Default: If you omit USING, DB2 defines the data sets in the default storage group of the database, using implicit specifications of PRIQTY 3, SECQTY 3, and ERASE NO.

VCAT *catalog-name*
Defines the data sets for the table space.

The ICF catalog named by *catalog-name* must already contain an entry for the first data set of the table space, conforming to the DB2 naming convention for data sets. If the name of the ICF catalog is longer than 8 characters you must use an alias.

STOGROUP *stogroup-name*
Lets DB2 define the data sets for the table space using space from the named storage group. If PRIQTY + 123 * SECQTY is 2 gigabytes or greater, more than one data set may eventually be used; only the firstdata set is defined during execution of this statement.

stogroup-name must name a storage group described in the local catalog, and SYSADM authority, or the USE privilege on that storage group, is required.

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.

The ICF catalog used for the storage group must NOT contain an entry for the first data set of the table space. If the ICF catalog is password protected, the description of the storage group must include a valid password.

PRIQTY *integer*
Gives the primary space allocation for the DB2-defined data sets. *integer* is a number of kilobytes not greater than 4194304. That maximum is used if you specify any larger value. Note that the amount of storage space specified by *integer* must be available on the first volume.

DB2 asks VSAM for an integral number, 3 or more, of pages. (Two pages of the primary space are used by DB2 for purposes other than storing user data.) The actual number of pages requested is given by these rules:

- If you do not use PRIQTY, the number is 3.
- If you do use PRIQTY, and if the page size of the table is:

  4K bytes, then the number is *integer*/4 rounded up to the next highest integer, but not less than 3.

  32K bytes, then the number is *integer*/32 rounded up to the next highest integer, but not less than 3.

The amount of space allocated is usually greater than that requested. DB2 requests a number of records, which VSAM changes to a request for tracks. To more closely estimate the actual amount, see a description of the DEFINE CLUSTER command in one of the VSAM Access Method Services publications.

Syntax note: Do not use PRIQTY more than once in the same using-block.

SECQTY *integer*
Gives the secondary space allocation for the DB2-defined data sets. *integer* is a number of kilobytes not greater than 131068. That maximum is used if you specify any larger value. If *integer* is 0, no secondary space allocation is used for the data sets.

DB2 asks VSAM for a number of pages, this number being determined by the following rules:

- If you do not use SECQTY, the number is 3.
- If you do use SECQTY, and if the page size of the table is:

4K bytes, then the number is *integer*/4 rounded up to the next highest integer, but not more than 32767.

32K bytes, then the number is *integer*/32 rounded up to the next highest integer, but not more than 4096.

The amount of space allocated is usually greater than that requested. DB2 requests a number of records, which VSAM changes to a request for tracks. To more closely estimate the actual amount, see a description of the DEFINE CLUSTER command in one of the VSAM Access Method Services publications.

Syntax note: Do not use SECQTY more than once in the same using-block.

ERASE
Indicates whether the DB2-defined data sets are to be erased (filled with 0's) when the table space is dropped.

NO
Does not erase the data sets. This is the default.

YES
Erases the data sets.

Syntax note: Do not use ERASE more than once in the same using-block.

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 an unpartitioned 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
- A default USING STOGROUP clause that specifies the default storage group for the database

VCAT *catalog-name*
Lets you define the data sets for the table space.

If *n* is the number of the partition, the ICF catalog named by *catalog-name* must already contain an entry for the *n*th data set of the table space, conforming to the DB2 naming convention for data sets.

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 TABLESPACE statement, using space from the named storage group. The ICF catalog used for the storage group must NOT contain an entry for the *n*th data set of the table space.

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.

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.

Do not use FREEPAGE more than once in any free-block. You must not use FREEPAGE for a table space in DSNDB07.

The default is **FREEPAGE 0**, leaving no free pages.

PCTFREE *integer*
Indicates what percentage of each page to leave as free space when the table is loaded or reorganized. *integer* may range from 0 to 99. The first record on each page is loaded without restriction. When additional records are loaded, at least *integer* % of free space is left on each page.

The default is **PCTFREE 5**. Do not use this keyword with database DSNDB07.

Do not use PCTFREE more than once in any free-block.

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 applies:

- 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 FREEPAGE 0 and PCTFREE 5

NUMPARTS *integer*
Partitions the table space. Do not use this keyword with database DSNDB07.

integer is the number of partitions, and may range from 1 to 64 inclusive. The maximum partition size is determined by the number of partitions as shown in the following table.

Number of Partitions	Maximum Size in Gigabytes[5]
1 to 16	4
17 to 32	2
33 to 64	1

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.

Default: If you omit NUMPARTS the table space is not partitioned and initially occupies one data set.

[5] 1 gigabyte is 2^{30} bytes.

PART *integer*
Specifies to which partition the following using-block or free-block applies. *integer* may range from 1 to the number of partitions given by NUMPARTS.

You may 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 BP*n*
Names the buffer pool to be associated with the table space. Use BP0, BP1, BP2, or BP32K. The buffer pool must be activated, and SYSADM authority, or the USE privilege on the buffer pool, is required.

This clause also sets the page size of the table space. If you use BP32K the page size is 32K bytes; otherwise it is 4K bytes.

The default is the default buffer pool of the database.

LOCKSIZE
Specifies the locking level for the table space. You must not use this clause for a table space in DSNDB07.

ANY
Specifies that DB2 may use any locking level. In most cases DB2 will use page level locking. 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. This is the default.

PAGE
Specifies page level locking.

TABLESPACE
Specifies table space level locking.

TABLE
Specifies table level locking. You must not specify TABLE for an unsegmented table space.

CLOSE
Indicates whether to close the data sets supporting the table space when there are no current users of the table space.

YES
Closes the data sets. This is the default.

NO
Does not close the data sets.

DSETPASS *password*
Specifies a password that is passed to VSAM when the data sets are used by DB2. *password* is a VSAM master level password in the form of a short identifier. If the password is a delimited identifier, it may contain any special characters acceptable to VSAM Access Method Services. If DSETPASS is omitted, a password is not passed to VSAM.

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 don't use a storage group, you define the password that protects the data sets through VSAM 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.

Note: The password does not apply to data sets managed by Storage Management Subsystem. Data sets defined to SMS should be protected by RACF or some similar external security system.

SEGSIZE *integer*
Indicates that the table space is to be segmented. *integer* specifies how many pages are to be assigned to each segment. If the SEGSIZE clause is not provided, the table space will not be segmented. *integer* must be a multiple of 4 such that $4 \leq integer \leq 64$.

Note that a segmented table space may not be partitioned and cannot be created in the database DSNDB07.

Notes

In order to create a table space in temporary file database DSNDB07, you must first issue the -STOP DATABASE(DSNDB07) command. Following your CREATE, issue -START DATABASE(DSNDB07). This process will make the new temporary table space you have created available for use by DB2.

Two DB2 subsystems can be cataloged on the ICF catalog. But two DB2 subsystems must not share the same ICF catalog alias because this is the only parameter that makes the data set names unique. Take care to ensure that the VCAT name specified for a user-defined data set will identify a data set unique to this DB2 subsystem.

Example

Create table space DSN8S22D in database DSN8D22A. Let DB2 define the data sets, using storage group DSN8G220. The primary space allocation is 52 kilobytes; the secondary, 20 kilobytes. The data sets need not be erased if the table space is dropped.

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 DSN8S22D
  IN DSN8D22A
  USING STOGROUP DSN8G220
    PRIQTY 52
    SECQTY 20
    ERASE NO
  LOCKSIZE PAGE
  BUFFERPOOL BP1
  CLOSE YES
  DSETPASS OSESAME;
```

CREATE VIEW

The CREATE VIEW statement creates a view on one or more local tables 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

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.

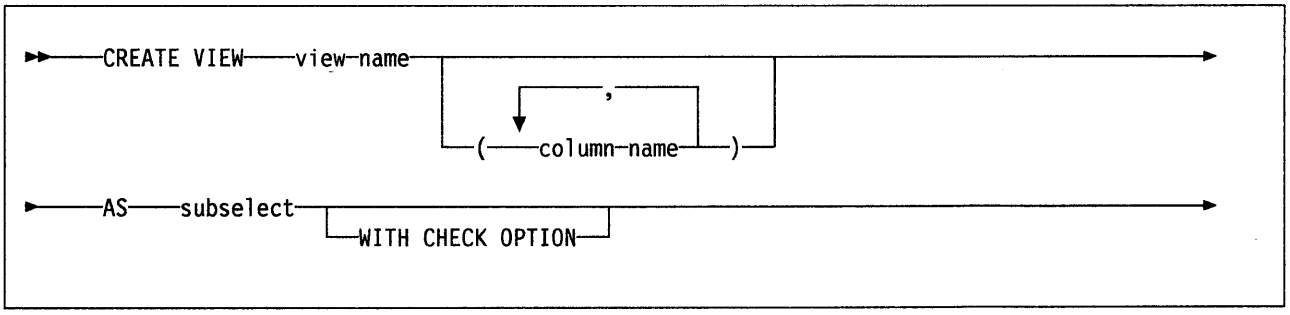
Authority requirements depend in part on the choice of the view's owner. For information on how to designate the owner, see *view-name* under *Description*.

If the statement is embedded in a program, the privilege set is the privileges designated by the authoriation ID of the owner of the plan:

- If this privilege set includes SYSADM authority, the owner of the view can be any authorization ID. With no errors in the statement, the view will be 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 authority, the owner of the view must be the owner of the application plan.

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. With no errors in the statement, the view will be created, even if the owner has no privileges at all on the tables and views identified in the view's subselect.
- If the SQL authorization ID of the process lacks SYSADM authority, only the primary and secondary authorization IDs of the process can own the view. In this case, the privilege set is the privileges designated by the authorization ID selected for ownership.



Description

view-name
Is the name of the view. The name supplied, including the implicit or explicit qualifiers, must not identify a table, view, alias, or synonym already described in the catalog.

If qualified, the name can be a two-part or three-part name. If it is a three-part name, the first qualifier must be the location-name of the local DB2 subsystem. 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 a program, the owner of the plan is the owner of the view. If the view name is unqualified and the statement is dynamically prepared, the SQL authorization ID is the owner of the view. 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 may 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.

Note that 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 with no other authority on the view.

column-name
Is the name of a column 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).

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 reference host variables or remote objects . For an explanation of *subselect*, see Chapter 5, "Queries" on page 83.

WITH CHECK OPTION
Indicates all inserts and updates against the view are to be checked against the view definition (the search condition of the WHERE clause) and rejected if the inserted or updated row does not conform to that definition. If the view does not have a WHERE clause, WITH CHECK OPTION is ignored.

If the view is read-only (see Notes below), or if the SELECT statement includes a subselect, this clause must not be specified. If the view definition is such that updates to some columns are allowed, but inserts into the view are not allowed, then WITH CHECK OPTION applies only to updates.

If the clause is omitted, inserts and updates are not checked against the view definition. If the view is dependent on other views, checking is performed according to the following rules:

- If this view, and the views on which it is dependent, use WITH CHECK OPTION, all inserts and updates to this view will be checked against the definitions of both this view, and the views on which it is dependent.
- If this view uses WITH CHECK OPTION, but none of the views on which it is dependent do so, all inserts and updates to this view will be checked against the definition of this view only.
- If this view does not use WITH CHECK OPTION, but the views on which it is dependent do use it, all inserts and updates to this view will be checked only against the definitions of the views on which it is dependent.
- If no view uses WITH CHECK OPTION, no checking is performed.

Notes

Read-only views: A view is read-only if its definition involves any of the following:

- The first FROM clause identifies more than one table or view
- The keyword DISTINCT in the first SELECT clause
- A GROUP BY clause in the outer subselect
- A HAVING clause in the outer subselect
- A column function in the first SELECT clause
- A subquery such that the base object of the SELECT statement, 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.

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 referenced in a subquery of a basic predicate.

A view cannot map to more than 16 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. The 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 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;
```

The addition of columns to A.T and the dropping of S have no effect on the operational form of the view definition. Thus, if columns are added to A.T or S is redefined, the source form of the view definition can be misleading.

Example

Create the view DSN8220.VPROJRE1. PROJNO, PROJNAME, PROJDEP, RESPEMP, EMPNO, FIRSTNME, MIDINIT, and LASTNAME are column names. The view is a join of tables DSN8220.PROJ and DSN8220.EMP, where a value in the RESPEMP column is equal to a value in the EMPNO column.

```
CREATE VIEW DSN8220.VPROJRE1
  (PROJNO,PROJNAME,PROJDEP,RESPEMP,
   FIRSTNME,MIDINIT,LASTNAME)
  AS SELECT ALL
  PROJNO,PROJNAME,DEPTNO,EMPNO,
  FIRSTNME,MIDINIT,LASTNAME
  FROM DSN8220.PROJ, DSN8220.EMP
  WHERE RESPEMP = EMPNO
```

Note: A column named in the WHERE, GROUP BY, or HAVING clause need not be part of the view. In the example, for instance, the WHERE clause refers to the colmun EMPNO, which is contained in one of the base tables but is not part of the view.

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.

The SELECT statement of the cursor is either:

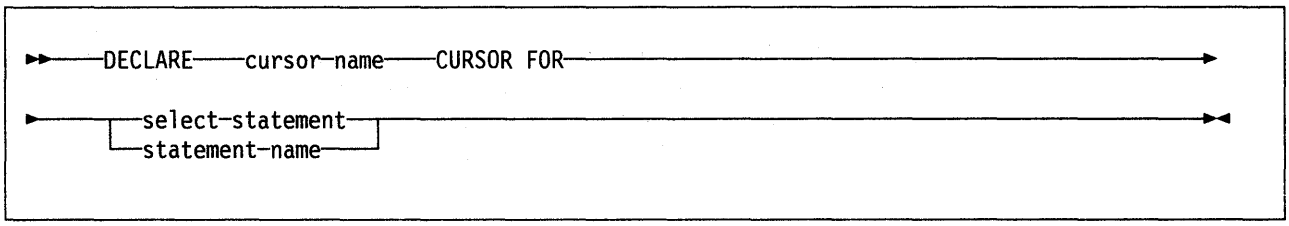
- The prepared *select-statement* identified by the *statement-name*, or
- The specified *select-statement*.

If *statement-name* is specified:

- For local execution, the privilege set is the union of the privileges designated by each authorization ID of the process. For remote execution, the privilege set consists of all the privileges recorded in the remote subsystem's catalog for an authorization ID derived from the primary authorization ID of the process. The derivation process, known as "translation," is discussed in Section 5 (Volume 2) of *Administration Guide*.
- 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:

- For local execution, the privilege set consists of the privileges designated by the authorization ID of the owner of the plan. For remote execution, the privilege set consists of all the privileges recorded in the remote subsystem's catalog for an authorization ID derived from the authorization ID of the owner of the plan. The derivation process, known as "translation," is discussed in Section 5 (Volume 2) of *Administration Guide*.
- If the plan 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 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 recovery. The OPEN is unsuccessful if any required privilege does not exist.



Description

A cursor with the specified name is defined. The name must not be the same as the name of another cursor declared in your source program.

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:

- The SELECT statement includes:
  - The keyword DISTINCT in the first SELECT clause
  - A UNION or UNION ALL operator
  - A column function in the first SELECT clause
  - A GROUP BY or HAVING clause in the outer SELECT statement
  - An ORDER BY clause
  - A subquery such that the base object of the SELECT statement and of the subquery is the same table
  - The FOR FETCH ONLY clause .
- The first FROM clause of the SELECT statement identifies:
  - More than one table or view
  - A read-only view.

Specifying the SELECT Statement: The *select-statement* must not include parameter markers, but may 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 93 for an explanation of *select statement*.

Naming the SELECT Statement: If a *statement-name* is specified, the SELECT statement of the cursor is the prepared SELECT statement identified by the *statement-name* when the cursor is opened.[6]

For an explanation of prepared SELECT statements, see "PREPARE" on page 218.

[6] The PREPARE statement used to create the prepared *select-statement* must precede the DECLARE CURSOR statement in your source program.

Notes

In COBOL and FORTRAN source programs, the DECLARE CURSOR statement must precede all statements that explicitly reference 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. The rule does apply to the other host languages if the two-pass option is not used.

The scope of *cursor-name* is the source program in which it is defined; that is, the program submitted to the precompiler. Thus, you can only reference 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 referenced in that subprogram.

Example

The DECLARE CURSOR statement associates the cursor name C1 with the results of the SELECT.

```
EXEC SQL DECLARE C1 CURSOR FOR
   SELECT DEPTNO, DEPTNAME, MGRNO
   FROM DSN8220.DEPT
   WHERE ADMRDEPT = 'A00'
   END-EXEC.
```

DECLARE STATEMENT

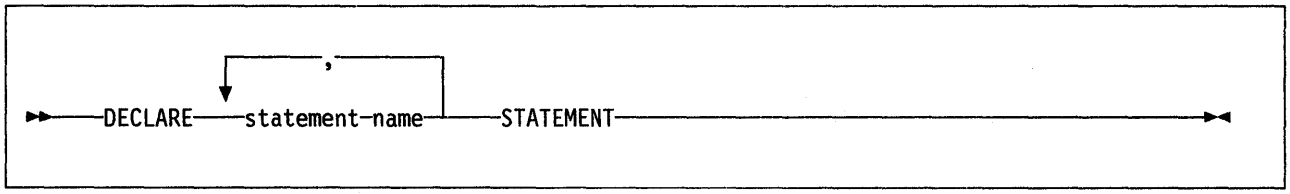
The DECLARE STATEMENT statement is used for program documentation. It declares names that are used to identify prepared SQL statements.

Invocation

This statement can only be embedded in an application program. It is not an executable statement.

Authorization

None required.



Description

statement-name **STATEMENT**
Lists one or more names that are used in your 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;

( SOURCE_STATEMENT is "SELECT DEPTNO, DEPTNAME,
  MGRNO FROM DSN8220.DEPT WHERE ADMRDEPT = 'A00'"  )

EXEC SQL INCLUDE SQLDA;
EXEC SQL DECLARE C1 CURSOR FOR OBJECT_STATEMENT;

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

The DECLARE TABLE statement is used for program documentation. It also provides the precompiler with information used to check your embedded SQL statements.

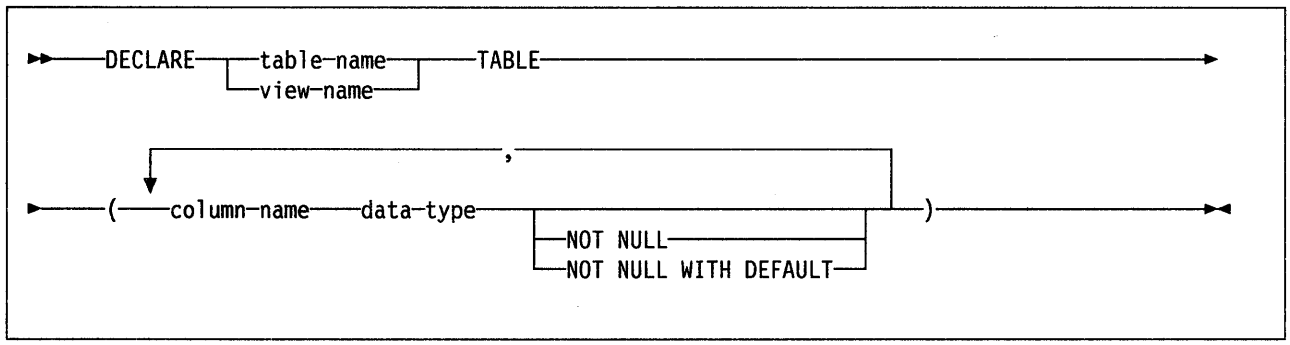
Note: DCLGEN can be used to generate table declarations for tables and views, either local or remote. For a given table or view, DCLGEN obtains the information from the appropriate system catalog. For more on DCLGEN, see *Command and Utility Reference*.

Invocation

This statement can only be embedded in an application program. It is not an executable statement.

Authorization

None required.



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 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. Use:

INTEGER For a large integer. You may also use INT.

SMALLINT
For a small integer.

FLOAT(*integer*)
For a floating-point number, the integer specifying the format. If the integer is between 1 and 21 inclusive, the format is that of single precision floating-point. If the integer is between 22 and 53 inclusive, the format is that of double precision floating-point. You may also specify:

FLOAT	for double precision floating-point
REAL	for single precision floating-point
DOUBLE PRECISION	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; it may range from 1 to 15. The second integer is the scale of the number, that is, the number of digits to the right of the decimal point; it may range from 0 to the precision. You may also specify:

DECIMAL(*integer*)	for DECIMAL(*integer*,0)
DECIMAL	for DECIMAL(5,0)

CHARACTER(*integer*) or CHAR(*integer*)
For a fixed-length character string of length *integer*, which may range from 1 to 254. If the length specification is omitted, a length of 1 character is assumed.

VARCHAR(*integer*)
For a varying-length character string of maximum length *integer*, which may range from 1 to 32767.

LONG VARCHAR
For a varying-length character string whose maximum length is determined by DB2.

DATE For a date.

TIME For a time.

TIMESTAMP
For a timestamp.

GRAPHIC (*integer*)
For a fixed-length string of double-byte characters, of length *integer*, which may 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 may range from 1 to 16383.

LONG VARGRAPHIC
For a varying-length string of double-byte characters whose maximum length is determined by DB2.

NOT NULL
Is used for a column that does not allow null values, and does not provide a default value.

NOT NULL WITH DEFAULT
Is used for a column that does not allow null values, but provides a default value.

Notes

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, DSN8220.EMP.

```
EXEC SQL DECLARE DSN8220.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)     NOT NULL,
   PHONENO  CHAR(4)              ,
   HIREDATE DATE                 ,
   JOBCODE  DECIMAL(3)           ,
   EDUCLVL  SMALLINT             ,
   SEX      CHAR(1)              ,
   BRTHDATE DATE                 ,
   SALARY   DECIMAL(8,2)          );
```

DELETE

The DELETE statement deletes rows from a table or view. Deleting a row from a view deletes the row from the table on which the view is based.

The forms of this statement are:

- The searched DELETE, which is used to delete one or more rows (optionally determined by a search condition).
- The positioned DELETE, which 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

When a 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 view is identified, the privilege set must include at least one of the following:

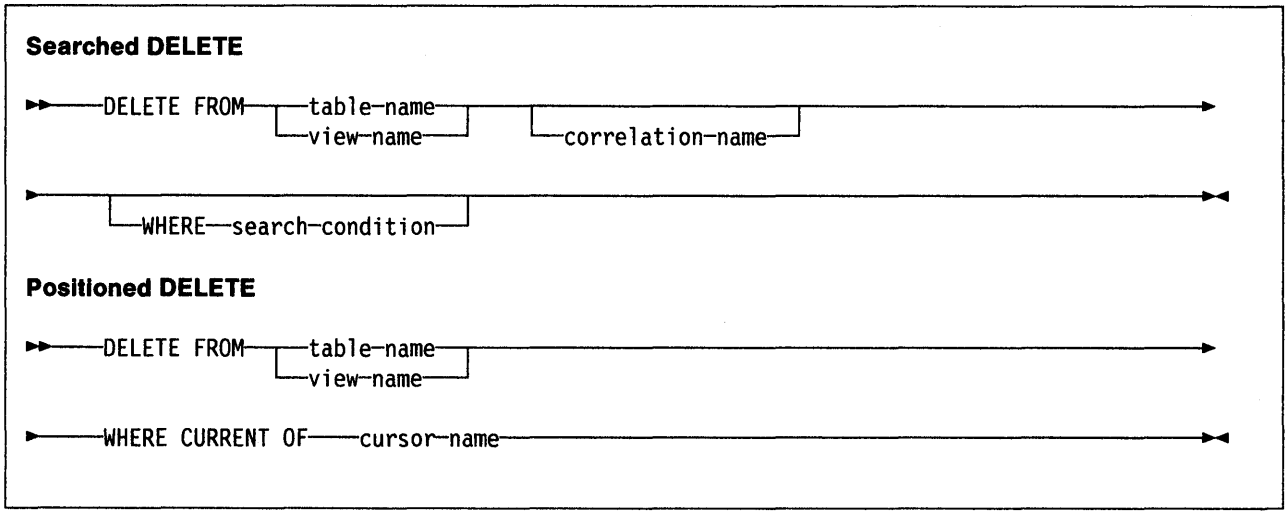
- The DELETE privilege on the view
- SYSADM authority.

Note that the owner of a view, unlike the owner of a table, might not have DELETE authority on the view (or may have DELETE authority without being able to grant it to others). Indeed, the nature of the view itself may preclude its use for DELETE. For more on all this, see the discussion of authority under "CREATE VIEW" on page 161.

For local execution of the statement: If the statement is embedded in a program, the privilege set is the privileges designated by the authorization ID of the owner of the plan. If the statement is dynamically prepared, the privilege set is the union of the privileges designated by each authorization ID of the process.

For remote execution of the statement: The privilege set consists of all those privileges recorded in the remote subsystem's catalog for a certain authorization ID. For a statement embedded in a program, this authorization ID is derived from the one that owns the plan. For a dynamically prepared statement, it is derived from the primary authorization ID of the process. The derived authorization ID may, of course, be equal to the original.

The derivation, known as "translation," is described in Section 5 (Volume 2) of *Administration Guide*. Controlling the procedure are the SYSIBM.SYSUSERNAMES tables in the local and remote communication databases. For a description of commmunication databases, see Appendix D, "The Communications Database" on page 295.



Description

FROM *table-name* or *view-name*

Names the table or view from which you want to delete. The *table-name* or *view-name* must identify a table or view described in the catalog of the DB2 subsystem identified by the implicitly or explicitly specified location-name. But it must not identify a catalog table, a view of a catalog table, or a read-only view. (For an explanation of read-only views, see "CREATE VIEW" on page 161.)

The referenced table or view can be remote. But it must be local if the process is attached to DB2 through the CICS or IMS/VS attachment facilities.

correlation-name

May be used within the *search-condition* to designate the table or view. (For an explanation of *correlation-name*, see Chapter 3.)

WHERE

Specifies a condition that selects the rows to be deleted. 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 deleted.

search-condition

Is any search condition as described in Chapter 3. Each *column-name* in the search condition, other than in a subquery, must name a column of the table or view. The search condition must not include a subquery such that the base object of both the DELETE and of the subquery is the same table.

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, the subquery is executed for each row only if it contains a correlated reference to a column of the table or view.

Let T2 denote the object table of a DELETE statement and let T1 denote a table that is referenced in the FROM clause of a subquery of that statement. T1 must not be a dependent of T2 in a relationship with a delete rule of CASCADE or SET NULL. Furthermore, 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. The *cursor-name* must identify a declared cursor as explained in the Notes for the DECLARE CURSOR statement. If the DELETE statement is embedded in a program, the DECLARE CURSOR statement must include a *select-statement* rather than a *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 165.)

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. An immediate FETCH will return the next row. If there is no next row, the cursor is positioned after the last row.

If the object table is self-referencing, WHERE CURRENT OF must not be specified.

Notes

If the object table of the delete operation is a parent table, the rows selected for deletion must not have any dependents in a relationship with a delete rule of RESTRICT and the delete operation must not cascade to descendent rows that are dependents in a relationship with a delete rule of RESTRICT. If the delete operation is not prevented by a RESTRICT delete rule, the selected rows are deleted and:

- The nullable columns of the foreign keys of any rows that are their dependents in a relationship with a delete rule of SET NULL are set to the null value
- Any rows that are their dependents in a relationship with a delete rule of CASCADE are also deleted and these rules apply, in turn, to those rows.

If an error occurs during the execution of any delete operation, no rows are deleted. If an error occurs during the execution of a cursor-controlled 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.

Unless appropriate locks already exist, one or more exclusive locks are acquired by the execution of a successful delete operation. Until the locks are released, they may prevent other processes from performing operations on the table.

If WHERE *search-condition* is omitted and the table from which the rows are deleted is contained in a segmented tablespace, SQLERRD(3) is set to -1. Otherwise, SQLERRD(3) is set to the number of deleted rows. However, this number does not include any rows that were deleted as a result of a CASCADE delete rule.

Examples

Example 1: From the table DSN8220.EMP delete the row on which the cursor C1 is positioned.

```
DELETE FROM DSN8220.EMP WHERE CURRENT OF C1;
```

Example 2: From the table DSN8220.EMP, delete all rows for departments E11 and D21.

```
DELETE FROM DSN8220.EMP
  WHERE WORKDEPT = 'E11'
  OR WORKDEPT = 'D21'
```

DESCRIBE

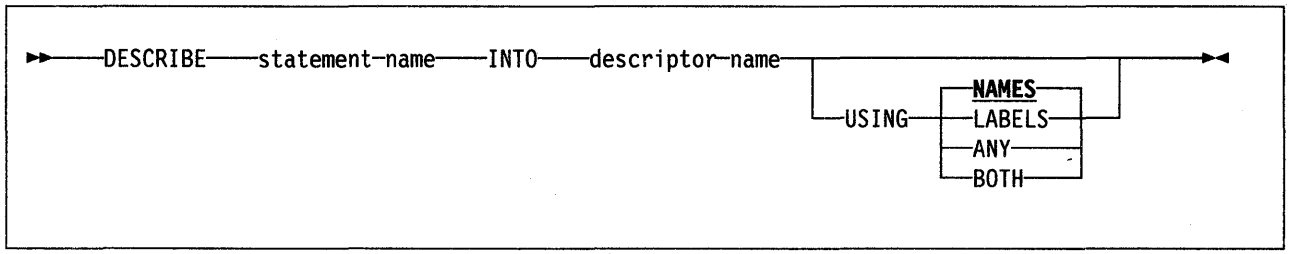
The DESCRIBE statement obtains information about a prepared statement. For an explanation of prepared statements, see "PREPARE" on page 218.

Invocation

This statement can only be embedded in an application program. It is an executable statement that cannot be dynamically prepared.

Authorization

None required. See "PREPARE" on page 218 for the authorization required to create a prepared statement.



Description

statement-name
Names the statement about which you want to obtain information. When the DESCRIBE statement is executed, the name must identify a prepared statement.

INTO *descriptor-name*
Names an SQL descriptor area (SQLDA). These areas are described in detail in Appendix B, "SQLCA and SQLDA" on page 249. When the host application is written in C, *descriptor-name* must be a pointer variable with pointer notation (for example, *sqlptr). When the DESCRIBE statement is executed, values are assigned to the variables of the SQLDA as follows:

Field	Information Entered
SQLDAID	'SQLDA'
SQLDABC	16 + SQLN*44
	If USING(BOTH) is used, and SQLN is not large enough for both column names and labels, SQLD is set to twice the number of columns.
SQLVAR	If the value of SQLD is 0, or greater than the value of SQLN, no values are assigned to occurrences of SQLVAR.
	If the value is *n*, where *n* is greater than 0 but less than or equal to the value of SQLN, values are assigned to the first *n* occurrences of SQLVAR so that the first occurrence of SQLVAR contains a description of the first column of the result table, the second occurrence of SQLVAR contains a description of the second column of the result table, and so on. The description of a column consists of the values assigned to SQLTYPE, SQLLEN, and SQLNAME.
	If USING(BOTH) is used, the first *n* occurrences of SQLVAR contain column names (where they exist). The second *n* occurrences contain

column labels (where they exist), but SQLTYPE and SQLLEN are not used. Occurrence $n+1$ contains the label of column 1; occurrence $n+2$, the label of column 2; and so on.

SQLTYPE A code showing the data type of the column and whether or not it can contain null values.

Value	Data Type	Nulls
384/385	date	no/yes
388/389	time	no/yes
392/393	timestamp	no/yes
448/449	varying-length character string	no/yes
452/453	fixed-length character string	no/yes
456/457	long character string	no/yes
464/465	varying-length graphic string	no/yes
468/469	fixed-length graphic string	no/yes
472/473	long graphic string	no/yes
480/481	floating-point	no/yes
484/485	decimal	no/yes
496/497	large integer	no/yes
500/501	small integer	no/yes

SQLLEN A value depending on the data type of the columns, as follows:

Data Type	Value
any string	the length attribute of the column; that is, the maximum number of characters in a value (either EBCDIC or double-byte)
FLOAT	4 for single precision; 8 for double precision.
INTEGER	4
SMALLINT	2
DECIMAL(*p*,*s*)	*p* in byte 1; *s* in byte 2.
DATE	10
TIME	8
TIMESTAMP	26

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 appropriate name or label does not exist.

USING

Indicates what value to assign to each SQLNAME variable in the SQLDA. If the requested value does not exist, SQLNAME is set to a length of 0.

NAMES

Assigns the name of the column. This is the default.

LABELS
Assigns the label of the column. (Column labels are defined by the LABEL ON statement.)

ANY
Assigns the column label, and if the column has no label, the column name.

BOTH
Assigns both the label and name of the column. In this case, two occurrences of SQLVAR per column will be needed to accommodate the additional information. To specify this expansion of the SQLVAR array, set SQLN to 2*n* on the PREPARE statement (where *n* is the number of columns in the result table). Then, on any later FETCH statement, set SQLN to *n*. 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.

Notes

Information about a prepared statement can also be obtained by using the INTO clause of the PREPARE statement.

Before the DESCRIBE or PREPARE INTO statement is executed, the value of SQLN must be set to indicate how many occurrences of SQLVAR are provided in the SQLDA. To obtain the description of the columns of the result table of a prepared SELECT statement, the number of occurrences of SQLVAR must not be less than the number of columns.

If USING BOTH is specified and SQLN is less than 2*SQLD, then SQLD is set to 2 * (number of columns). If USING BOTH is specified and SQLN is greater than or equal to 2*SQLD, then SQLD is set to the number of columns.

Because the maximum number of columns is 300, a simple technique is to provide an SQLDA with 300 occurrences of SQLVAR. However, such an SQLDA will occupy 13216 bytes, and most of this space will not be needed for most prepared statements. Thus you might want to consider another technique, such as the following:

Execute a DESCRIBE or PREPARE INTO 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.

Example

This PL/I example uses the technique described above. SOURCE is a varying-length string variable and SHORTDA is an SQLDA with no occurrences of SQLVAR.

```
EXEC SQL INCLUDE SQLDA;
```

(Read an SQL statement into SOURCE)

```
EXEC SQL PREPARE OBJSTATE INTO :SHORTDA
  FROM :SOURCE;
```

(Check for successful execution. If the value of SQLD is greater than 0, the source statement was SELECT; use the value of SQLD to allocate and initialize SQLDA.)

```
EXEC SQL DESCRIBE OBJSTATE INTO :SQLDA;
```

DROP

The DROP statement deletes a local object. Any objects that are directly or indirectly dependent on that object are also deleted. Whenever an object is deleted, its description is deleted from the DB2 catalog, and any application plans that reference 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.

Authorization

To drop a table, tablespace, or index, the privilege set defined below must include at least one of the following:

- Ownership of the object
- DBADM authority
- SYSADM authority.

To drop an alias, storage group, or view, the privilege set defined below must include at least one of the following:

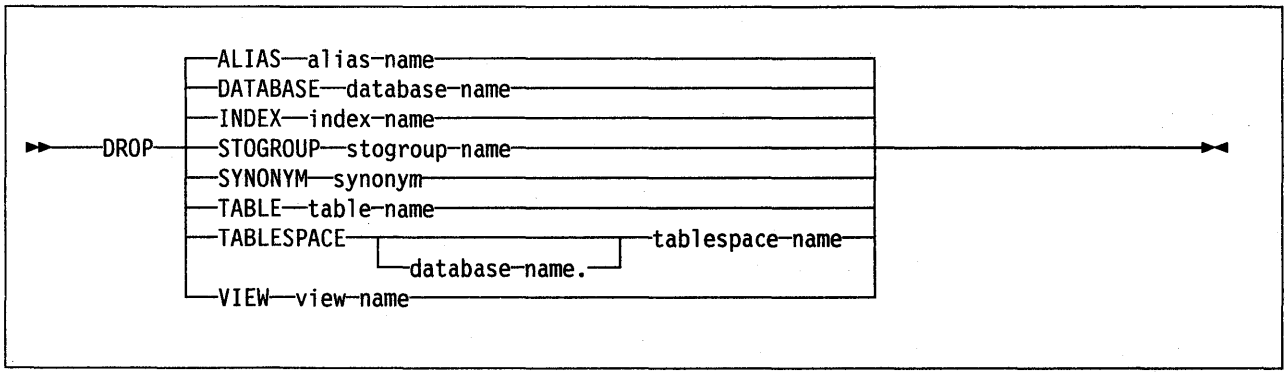
- Ownership of the object
- SYSADM authority.

To drop a database, the privilege set defined below must include at least one of the following:

- The DROP privilege on the database
- DBCTRL authority for the database
- DBADM authority for the database
- SYSADM authority.

If the statement is embedded in a program, the privilege set is the privileges designated by the authorization ID of the owner of the plan. If the statement is dynamically prepared, the privilege set is the union of the privileges designated by each authorization ID of the process.

To drop a synonym, ownership is required as explained below under SYNONYM.



Description

ALIAS *alias-name*
Identifies the alias you want to drop. The alias specified must be described in the catalog. Dropping an alias has no effect on any view or synonym that was defined using the alias.

DATABASE *database-name*
Dropping a database drops all table spaces, tables, and indexes in the database.

The *database-name* must identify a database described in the catalog, other than DSNDB04 and DSNDB06. If DSNDB07 is named, it must be in the stopped state.

INDEX *index-name*
Identifies an unpartitioned and user-created index described in the catalog. A partitioned index can only be dropped by dropping its associated table space.

Dropping an index always drops the index space. If you drop an index, you must issue a COMMIT before attempting to reuse its name.

If a primary index is dropped, the definition of its table is changed to incomplete.

STOGROUP *stogroup-name*
Identifies the storage group you want to drop. The *stogroup-name* must be a storage group that is described in the catalog, but not one that is used by any table space or index space. You may drop the default storage group of a database; for the effect, see "Notes" on page 181.

Note: SYSDEFLT must *not* be identified.

SYNONYM *synonym*
For synonym, specify the synonym you want to drop. In a static DROP SYNONYM statement, the name must identify a synonym that is owned by the owner of the plan. In a dynamic DROP SYNONYM statement, the name must identify a synonym that is owned by the SQL authorization ID.

An authorization ID with SYSADM privileges can drop synonyms on behalf of other users by changing the value of the CURRENT SQLID special register.

Dropping a synonym has no effect on any view or alias that was defined using the synonym.

TABLE *table-name*
Identifies a table you want to drop. The table specified must be described in the catalog and cannot be a catalog table or a table in a partitioned table space. Although a three-part name may be specified, the table must be local. The table is deleted from the database. All synonyms, indexes, and views defined on the table, all privileges granted on the table, and all referential constraints in which it is a parent or dependent are also dropped. If the table space for the table was implicitly created, it is also dropped.

A table in a partitioned table space can only be dropped by dropping the table space.

TABLESPACE
The combination of the *database-name* and *tablespace-name* must identify a table space described in the catalog, other than a table space of DSNDB06.

database-name
Identifies the database that contains the table space you want to drop. The default is DSNDB04.

tablespace-name
Identifies the table space you want to drop. All tables contained in the table space and all objects dependent on these tables will also be dropped.

If you drop a table space, you must issue a COMMIT before attempting to reuse its name.

VIEW *view-name*
Identifies the view you want to drop. The view specified must be described in the catalog. Although a three-part name may be specified, the view must be local. The definition of the view is deleted from the catalog. The definition of any view that is directly or indirectly dependent on that view is also deleted. Whenever the definition of a view is deleted from the catalog, all privileges on that view are also deleted.

Notes

DROP is subject to these restrictions:

- DROP DATABASE cannot be performed while a DB2 utility has control of any part of the database.
- DROP INDEX cannot be performed while a DB2 utility has control of the index or its associated table space.
- DROP TABLE cannot be performed while a DB2 utility has control of the table space that contains the table.
- DROP TABLESPACE cannot be performed while a DB2 utility has control of the table space.

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, it ceases to exist. Until you create another storage group with the same name, you must use USING when creating a table space or index in the database.

Dropping a temporary table space: In order to drop a tablespace in temporary file database DSNDB07, you must first issue the -STOP DATABASE(DSNDB07) command. Following your DROP, issue -START DATABASE(DSNDB07). This process will remove the temporary table space you dropped from the pool of table spaces available to DB2.

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 the active RLST.

Dropping an object in the communications database: You cannot drop an object in the communications database while the distributed data facility (DDF) is active.

Dropping an alias: Dropping a table or view does not drop its aliases. To drop an alias, use the DROP ALIAS statement.

Examples

Example 1: Drop table DSN8220.DEPT.

```
DROP TABLE DSN8220.DEPT
```

Example 2: Drop table space DSN8S22D in database DSN8D22A.

```
DROP TABLESPACE DSN8D22A.DSN8S22D
```

Example 3: Drop the view VDEPT.

```
DROP VIEW VDEPT
```

END DECLARE SECTION

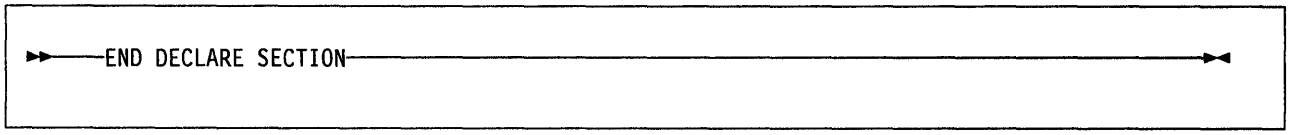
The END DECLARE SECTION statement marks the end of a host variable declare section.

Invocation

This statement can only be embedded in an application program. It is not an executable statement.

Authorization

None required.



Description

The END DECLARE SECTION statement may 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 on page 122.

The following rules are enforced by the precompiler only if the host language is C or the STDSQL(86) precompiler option is specified:

- A variable referenced 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.

Notes

Host variable declaration sections are only required if the STDSQL(86) option is specified or the host language is C. However, declare sections may be specified for any host language so that the source program can conform to the SAA definition of 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

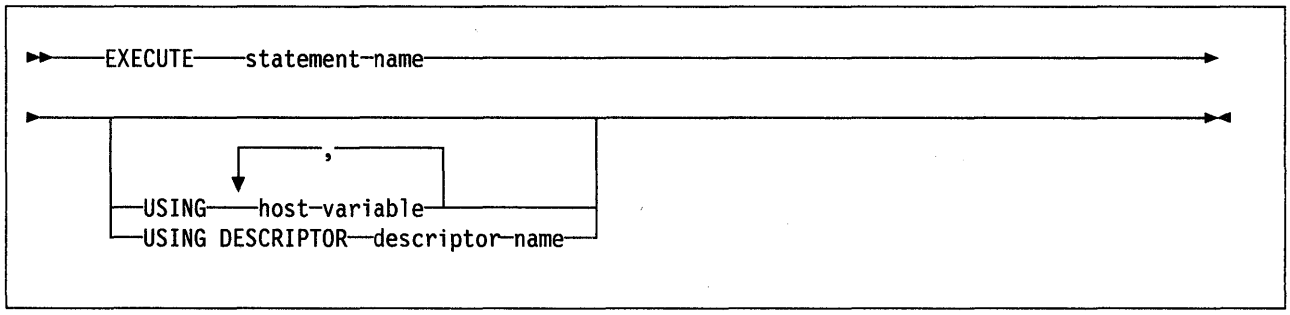
The EXECUTE statement executes a prepared SQL statement.

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 218 for the authorization required to create a prepared statement.



Description

statement-name
Identifies the prepared statement to be executed. *statement-name* must identify a statement that was previously prepared within the unit of recovery 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 218.) If the prepared statement includes parameter markers, you must use USING. USING is ignored if there are no parameter markers.

host-variable
Identifies a structure or variable that is described in the program in accordance with the rules for declaring host structures and variables. When the statement is executed, a reference to a structure has been 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 must contain a valid description of host variables. The number of variables, as indicated by SQLD, must be the same as the number of parameter markers in the prepared statement and the length of the SQLDA, as indicated by SQLDABC, must be sufficient to describe that number of variables. The *n*th variable described by the SQLDA corresponds to the *n*th parameter marker in the prepared statement.

If the host application is written in C, then *descriptor-name* must be a pointer with pointer notation (for example, *sqlprt). (For a description of an SQLDA, see "SQL Descriptor Area (SQLDA)" on page 252.)

Notes

Before the prepared statement is executed, each parameter marker is effectively replaced by the value of its corresponding host variable. The following rules apply to these parameter values:

- If the parameter marker appears in the place of an insert or update value, the parameter value is the insert or update value and must therefore be compatible with the column to which it is assigned as specified in the description of the INSERT and UPDATE statements. Any conversion of the value is performed in accordance with the assignment rules described in Chapter 3.
- If the parameter marker appears as the operand of a unary minus, the parameter value must be a number. If it is not a double precision floating number, it is converted to double precision floating point before the operation is performed.
- If the parameter marker appears as the operand of an infix arithmetic operator, the parameter value must be a number. If the data type, precision, and scale of the parameter value is not the same as the data type, precision, and scale of the other operand of the arithmetic operator, the parameter value is converted to the data type, precision, and scale of that operand before the operation is performed.
- If the parameter marker appears in a predicate, the parameter value must be compatible with the other operand of that predicate. If the parameter value is a string, it must not be longer than the other operand. If the parameter value is a number that does not have the same data type, precision, and scale as the other operand, the parameter value is converted to the data type, precision, and scale of that operand before the operation is performed. In the case of the BETWEEN predicate, the 'other operand' is the first operand that is specified solely as a column-name. If no operand of BETWEEN is specified solely as a column-name, the 'other operand' is the closest operand (in left to right order) that is not specified with a parameter marker. In the case of the IN predicate, the 'other operand' is the closest operand (in left to right order) that is not specified with a parameter marker.

Example

In this example, an INSERT statement with parameter markers is prepared and executed. S1 is a structure that corresponds to the format of DSN8220.DEPT.

```
EXEC SQL PREPARE DEPT_INSERT FROM
  'INSERT INTO DSN8220.DEPT VALUES(?,?,?,?)';

(Check for successful execution and read values into S1)

EXEC SQL EXECUTE DEPT_INSERT USING S1;
```

EXECUTE IMMEDIATE

The EXECUTE IMMEDIATE statement:

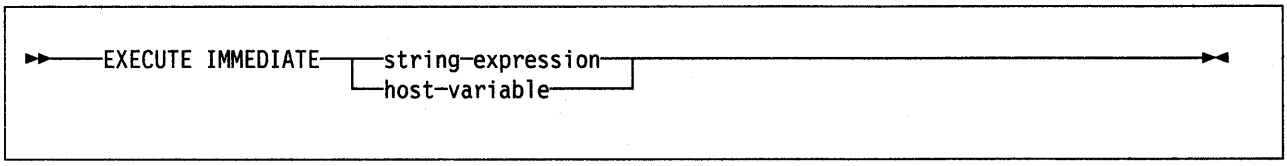
- Prepares an executable form of an SQL statement from a character string form of the statement.
- Executes the SQL statement.
- Destroys the executable form.

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 207 for the authorization rules that apply when an INSERT statement is executed using EXECUTE IMMEDIATE.



Description

string-expression
A *string-expression* is any PL/I expression that yields a character string.

host-variable
For languages other than PL/I, *host-variable* must be specified. It must identify a host variable that is described in the program in accordance with the rules for declaring character string variables. In Assembler language, C, and COBOL, the host variable must be a varying-length string variable. In C, it must not be a null-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, COMMENT ON, COMMIT, CREATE, DELETE, DROP, EXPLAIN, GRANT, INSERT, LABEL ON, LOCK TABLE, REVOKE, ROLLBACK, SET, or UPDATE.

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.

The Resource Limit Facility (Governor) may stop 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.

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.

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 DSN8220.DEPT
  WHERE' || PREDS;
```

EXPLAIN

The EXPLAIN statement obtains access path selection information about a SELECT, INSERT, UPDATE, or DELETE statement. The information is placed in a user-supplied table.

Invocation

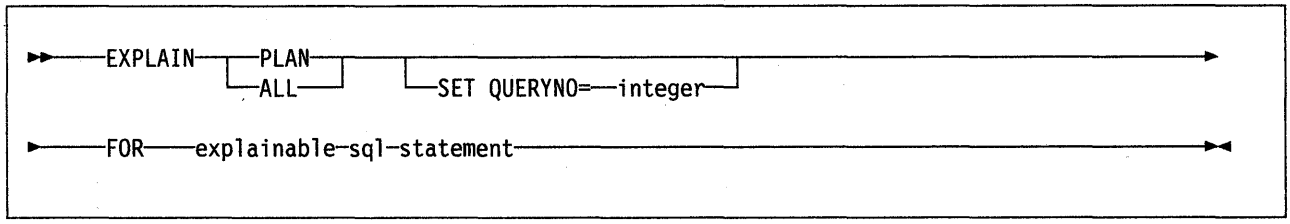
This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared.

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 specified in an EXPLAIN statement.

If the EXPLAIN statement is embedded in a program, the authorization rules that apply are those defined for embedding the specified SQL statement in a program. In addition, the authorization ID of the owner of the plan must also be the owner of a 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 table named PLAN_TABLE.



Description

PLAN
Inserts one row in '*userid*.PLAN_TABLE' for each step used in executing *explainable-sql-statement*. Not included are the steps for enforcing referential constraints. The table is described under "Output" on page 189.

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 in *userid*.PLAN_TABLE by the EXPLAIN statement.

FOR *explainable-sql-statement*
Identifies the SQL statement that is to be explained. *explainable-sql-statement* may be any SQL statement that begins with SELECT, DELETE, INSERT, or UPDATE. The statement must refer to local DB2 objects only.

explainable-sql-statement cannot be a statement-name or a host-variable. If you want to use EXPLAIN to get information about dynamic SQL statements, you must prepare the entire EXPLAIN statement dynamically.

Output

Output from EXPLAIN is one or more rows of data inserted in the table *userid*.PLAN_TABLE. The table must have been created by or for the user of the EXPLAIN statement before that statement is executed. For information on using the table, see Section 7 (Volume 3) of *Administration Guide*. The table is described below.

PLAN_TABLE: To create PLAN_TABLE, execute this SQL statement:

```
CREATE TABLE 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,
        ACCESSTYPE     CHAR(2)       NOT NULL,
        MATCHCOLS      SMALLINT      NOT NULL,
        ACCESSCREATOR  CHAR(8)       NOT NULL,
        ACCESSNAME     CHAR(18)      NOT NULL,
        INDEXONLY      CHAR(1)       NOT NULL,
        SORTN_UNIQ     CHAR(1)       NOT NULL,
        SORTN_JOIN     CHAR(1)       NOT NULL,
        SORTN_ORDERBY  CHAR(1)       NOT NULL,
        SORTN_GROUPBY  CHAR(1)       NOT NULL,
        SORTC_UNIQ     CHAR(1)       NOT NULL,
        SORTC_JOIN     CHAR(1)       NOT NULL,
        SORTC_ORDERBY  CHAR(1)       NOT NULL,
        SORTC_GROUPBY  CHAR(1)       NOT NULL,
        TSLOCKMODE     CHAR(3)       NOT NULL,
        TIMESTAMP      CHAR(16)      NOT NULL,
        REMARKS        VARCHAR(254)  NOT NULL,
        PREFETCH       CHAR(1)       NOT NULL WITH DEFAULT,
        COLUMN_FN_EVAL CHAR(1)       NOT NULL WITH DEFAULT,
        MIXOPSEQ       SMALLINT      NOT NULL WITH DEFAULT)
   IN database-name.tablespace-name;
```

where *database-name.tablespace-name* names a database and table space you have authorization to use.

The list of columns up through REMARKS is a required minimum. Be certain to define the columns in the order shown. The last three columns are optional. You can either include all three of them, or none of them at all. You cannot include additional columns in the table definition.

The following table explains the columns in PLAN_TABLE.

Table 5 (Page 1 of 2). Columns in PLAN_TABLE. The results of the EXPLAIN statement are stored here.

Column Name	Description
QUERYNO	A number that identifies the EXPLAIN statement. You can specify a number with the SET QUERYNO clause; otherwise, DB2 will assign one.
QBLOCKNO	A number that identifies the query or subquery for the row. The number reflects the query's order of appearance in the EXPLAIN statement.
APPLNAME	For an EXPLAIN statement embedded in an application program, the name of the application plan. For a dynamic EXPLAIN statement, blank.
PROGNAME	For an EXPLAIN statement embedded in an application program, the name of the program; otherwise, blank.
PLANNO	A number identifying the step of the plan in which the subquery in QBLOCKNO was processed. This column indicates the order in which the steps of the plan were executed.
METHOD	A number (0, 1, 2, or 3) that indicates the join method used for this step of the plan: 0 First table accessed (PLANNO = 1). 1 *Nested loop* join. For each row of the present composite table, matching rows of a new table are found and joined. 2 *Merge scan* join. The present composite table and another table are scanned in the order of the join column, and matching rows are joined. 3 Additional sorts needed by ORDER BY, GROUP BY, SELECT DISTINCT or a quantified predicate. This step does not access a new table.
CREATOR	Creator of the new table accessed in this step; if METHOD is 3, blank.
TNAME	Name of the new table accessed in this step; if METHOD is 3, blank.
TABNO	A number that identifies the FROM-clause table reference for the row. The number reflects the position of the reference in the EXPLAIN statement. 0 if METHOD is 3.
ACCESSTYPE	Method of accessing the new table: I By an index (identified in ACCESSCREATOR and ACCESSNAME). I1 One-fetch index scan. N Index scan when predicate contains IN keyword. R By sequential scan of its pages. M By a multiple index scan; followed by MX, MI, or MU MX By a multiple index scan on the index named in ACCESSNAME MI By an intersection of multiple indexes MU By a union of multiple indexes blank Either: By the first index created on the table (that is, by QBLOCKNO 1 of an INSERT statement); or By UPDATE and DELETE statements that use WHERE CURRENT OF cursor; those are accessed by the cursors.
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 sufficient to satisfy the request, or whether data too must be accessed. Y = Yes; N = No.
SORTN_UNIQ	Whether a sort is performed on the new table to remove duplicate rows. Y = Yes; N = No.

Table 5 (Page 2 of 2). Columns in PLAN_TABLE. The results of the EXPLAIN statement are stored here.	
Column Name	**Description**
SORTN_JOIN	Whether a sort is performed on the new table if METHOD is 2. 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 2. Y = Yes; N = No.
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	The lock mode of the table space that contains the new table. IS = intent share; IX = intent exclusive; S = share; X = exclusive.
TIMESTAMP	The time at which the EXPLAIN statement was processed.
REMARKS	A field into which you can insert any character string of 254 or fewer characters.
PREFETCH	A character that indicates whether data pages were read in advance by PREFETCH. S = pure sequential PREFETCH; L = PREFETCH through a page list; blank = unknown or no prefetch.
COLUMN_FN_EVAL	A character that indicates when an SQL column function was evaluated. R = at data retrieval time; S = at sort time; blank = either at data manipulation time or unknown.
MIXOPSEQ	A number that indicates the sequence of steps in a multiple index operation. 1, 2, ... n — For the steps of the multiple index procedure (ACCESSTYPE is MX, MI, or MU) 0 — For any other rows (ACCESSTYPE is I, I1, M, N, R, or blank)

Example

Find information about an SQL statement:

```
EXPLAIN PLAN SET QUERYNO = 1
FOR SELECT X.ACTNO, X.PROJNO, X.EMPNO, Y.JOBCODE, Y.EDUCLVL
    FROM DSN82.EMPPROJACT X, DSN82.EMP Y
       WHERE X.EMPNO = Y.EMPNO
          AND X.EMPTIME > .5
          AND (Y.JOBCODE >= 58 OR Y.EDUCLVL >= 18)
       ORDER BY X.ACTNO, X.PROJNO
```

FETCH

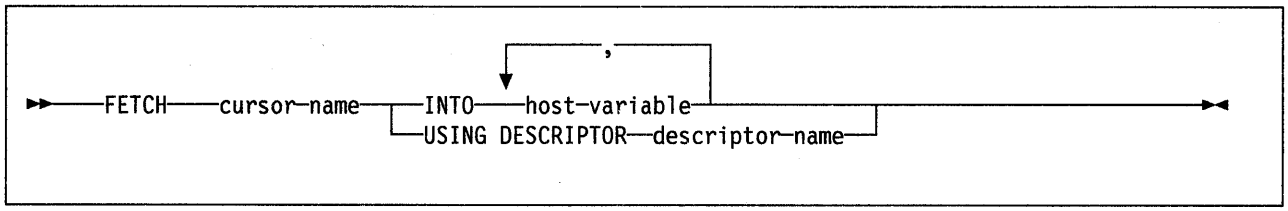
The FETCH statement positions a cursor on the next row of its result table and assigns the values of that row to host variables.

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 165 for an explanation of the authorization required to use a cursor.



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 Notes for the DECLARE CURSOR statement. 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, 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*

If INTO is used, each occurrence of *host-variable* must identify a structure or variable that is described in your program in accordance with the rules for declaring host structures and variables. In the operational form of INTO, a reference to a structure has been replaced by a reference to each of its variables. If the number of variables is less than the number of values, the variable 'W' is assigned to SQLWARN3.

USING DESCRIPTOR *descriptor-name*

Identifies an SQLDA that contains a valid description of zero or more host variables. The length of the SQLDA, as indicated by SQLDABC, must be sufficient to describe the number of variables indicated by SQLD. The first value of a row corresponds to the first variable described by the SQLDA, the second value corresponds to the second variable, etc. When the host application is written in C, *descriptor-name* must be a pointer variable with pointer notation (for example, *sqlptr).

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 date/time value, the variable must be a character string variable of a minimum length as defined in "String Representations of Date/Time Values" on page 29. 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 17. If the number of variables is less than the number of values in the row, the SQLWARN3 field of the SQLDA 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 subsequent fetch are unpredictable. It is possible for an error to occur that makes the position of the cursor invalid, in which case the cursor is closed.

Notes

An open cursor can be positioned before a row, on a row, or after the last row. If a cursor is on a row, that row is called the current row of the cursor. A cursor referenced in an UPDATE or DELETE statement must be positioned on a row. A cursor can be on a row only as a result of a FETCH statement.

Example

The FETCH statement fetches the results of the SELECT statement into the program variables DNUM, DNAME, and MNUM. When no more rows remain to be fetched, the 'notfound' condition is returned.

```
EXEC SQL DECLARE C1 CURSOR FOR
  SELECT DEPTNO, DEPTNAME, MGRNO FROM DSN8220.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:

- Database
- Plan
- System
- Table
- Use.

The applicable objects are always local.

Invocation

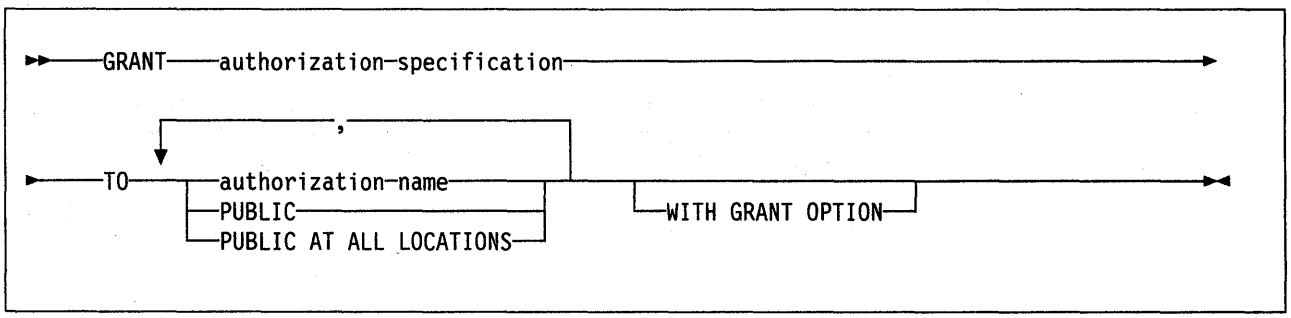
This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared. If the authorization mechanism was not activated when the DB2 subsystem was installed, an error condition occurs.

Authorization

The privilege set defined below must include the GRANT option for every privilege specified in the statement. The GRANT option may have been explicitly granted when the privilege was granted or it may be inherent in another privilege.

The GRANT option for any privilege is inherent in SYSADM authority. 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 161 for a description of rules that apply to views.

If the statement is embedded in a program, the privilege set is the privileges designated by the authorization ID of the owner of the plan. If the statement is dynamically prepared, the privilege set is the privileges designated by the SQL authorization ID of the process.



Description

authorization-specification
Names one or more privileges in one of the formats described on the following pages.

TO
Specifies to whom the privileges are granted.

authorization-name
Lists one or more authorization IDs. You cannot use the ID of the GRANT statement itself. (You cannot grant privileges to yourself.)

PUBLIC
Grants the privileges to all users at the local subsystem.

PUBLIC AT ALL LOCATIONS
Grants the privileges to all users in the network. Applies to table privileges only, excluding ALTER and INDEX.

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 designated privileges are granted, but without GRANT authority.

Notes

For more on DB2 privileges, read Section 5 (Volume 2) of *Administration Guide*.

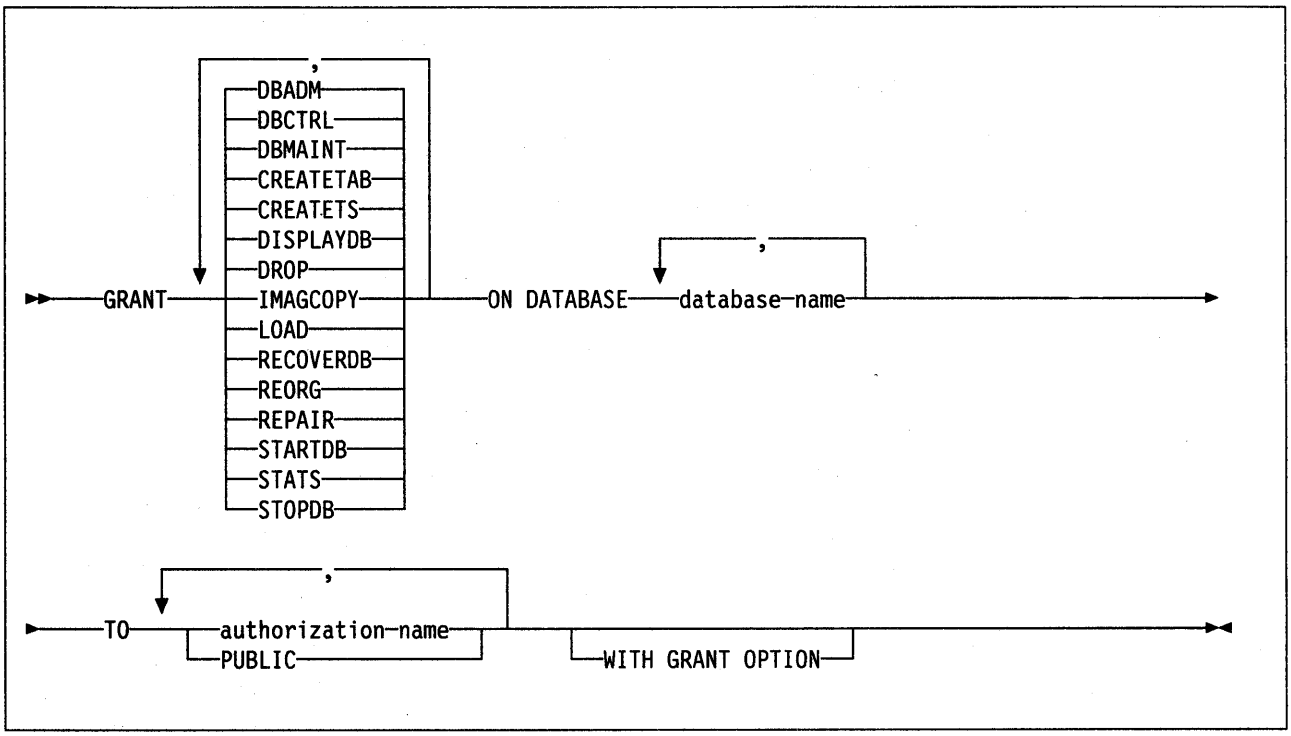
The result of executing a GRANT statement is recorded as one or more "grants" in the local catalog. A grant is the granting of a specific privilege by a specific grantor to a specific grantee. The grantor for a given GRANT statement is the authorization ID for the privilege set; that is, the SQL authorization ID of the process or the authorization ID of the plan's owner. As recorded in the catalog, the grantee is either an authorization ID, "PUBLIC," or "PUBLIC*," where PUBLIC* denotes PUBLIC AT ALL LOCATIONS.

Multiple grants are recorded for a statement in which more than one privilege is specified for *authorization-specification* or more than one grantee is specified after the keyword TO. If, for some reason, one of the grants is in error, execution of the statement is stopped and all its grants are canceled.

Different grantors may grant the same privilege to a single grantee. The grantee retains this privilege as long as one or more of these grants are recorded in the catalog. The grantee also retains the privilege if some grant is recorded of another privilege that implies the original privilege. Privileges that imply other privileges are also termed "authorities." Grants are removed from the catalog by executing SQL REVOKE statements.

GRANT (DATABASE PRIVILEGES)

This form of the GRANT statement grants privileges on databases.



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 drop 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 to delete records from the SYSIBM.SYSCOPY catalog table and the SYSIBM.SYSGRNG directory table.

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. Note, however, that only someone with Install SYSADM authority can can use the RECOVER utility against the catalog and the directory.

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. Note, however, that only someone with Install SYSADM authority can use the REPAIR utility against the catalog and directory.

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. Note, however, that only someone with Install SYSADM or Install SYSOPR authority can use the CHECK utility on the catalog and directory.

STOPDB
Grants the privilege to issue the -STOP DATABASE command.

ON DATABASE *database-name*
Lists one or more databases on which privileges are to be granted. For each designated database, the grantor must have all the specified privileges with the GRANT option. Each database must be described in the catalog. Privleges cannot be granted on database DSNDB01, which is not described in the catalog.

TO
Refer to "GRANT" on page 194 for a description of the TO clause.

Examples

Example 1: Grant drop privileges on database DSN8D22A to user PEREZ.

```
GRANT DROP
  ON DATABASE DSN8D22A
  TO PEREZ;
```

Example 2: Grant repair privileges on database DSN8D22A to all local users.

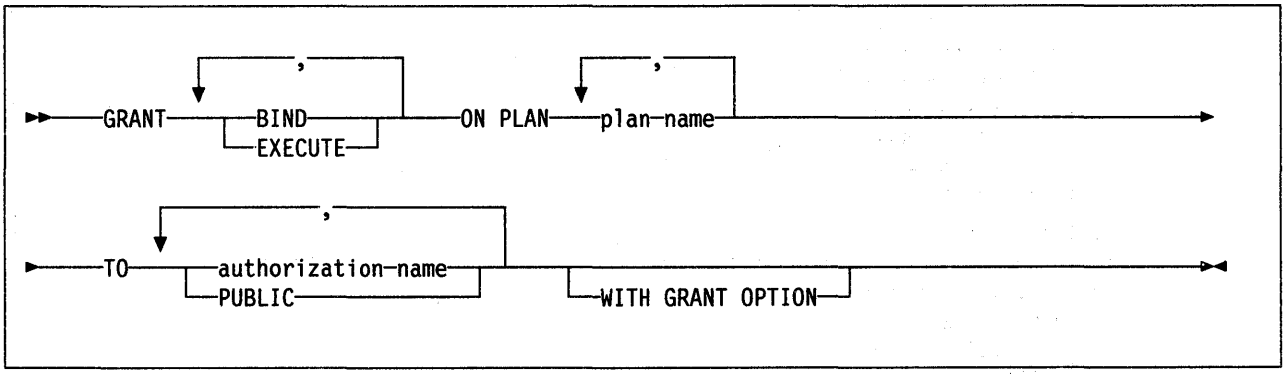
```
GRANT REPAIR
  ON DATABASE DSN8D22A
  TO PUBLIC;
```

Example 3: Grant authority to create new tables and load tables in database DSN8D22A, to users WALKER, PIANKA, and YOSHIMURA, and give them grant privileges.

```
GRANT CREATETAB,LOAD
  ON DATABASE DSN8D22A
  TO WALKER,PIANKA,YOSHIMURA
  WITH GRANT OPTION;
```

GRANT (PLAN PRIVILEGES)

This form of the GRANT statement grants privileges on application plans.



Description

BIND
Grants the privilege to use the BIND, REBIND, and FREE subcommands against the application plans named. (The authority to create new plans using BIND ADD is a system privilege.)

EXECUTE
Grants the privilege to run programs that use the application plans named.

ON PLAN *plan-name*
Lists one or more application plans for which you are granting privileges. For each plan you name, you must have all specified privileges with the GRANT option.

TO
Refer to "GRANT" on page 194 for a description of the TO clause.

Examples

Example 1: Grant authority to bind plan DSN8IP22 to user JONES.

```
GRANT BIND ON PLAN DSN8IP22 TO JONES;
```

Example 2: Grant authority to bind and execute plan DSN8CP22 to all local users.

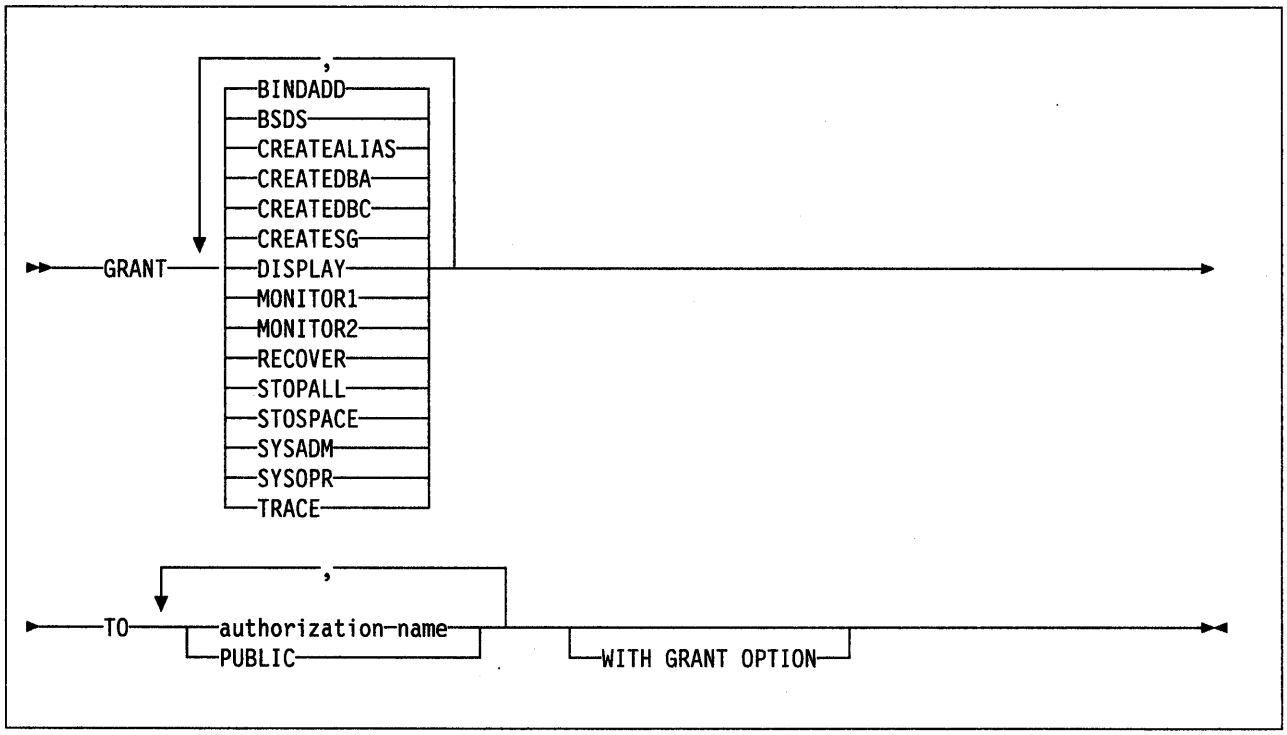
```
GRANT BIND,EXECUTE ON PLAN DSN8CP22 TO PUBLIC;
```

Example 3: Grant authority to execute plan DSN8CP22 to users ADAMSON and BROWN with grant option.

```
GRANT EXECUTE ON PLAN DSN8CP22 TO ADAMSON,BROWN WITH GRANT OPTION;
```

GRANT (SYSTEM PRIVILEGES)

This form of the GRANT statement grants system privileges.



Description

BINDADD
Grants the privilege to create application plans by using the BIND subcommand with the ADD option.

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 create new databases and acquire DBADM authority over those databases.

CREATEDBC
Grants the privilege to create new databases and acquire DBCTRL authority over those databases.

CREATESG
Grants the privilege to create new storage groups.

DISPLAY
Grants the privilege to do the following:

- Use the -DISPLAY THREAD command for information on active threads within DB2
- Use the -DISPLAY DATABASE command for the status of all databases.
- Use the -DISPLAY LOCATION and -DISPLAY TRACE commands.

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 Install SYSADM authority. The privileges the user possesses are all grantable, including the SYSADM authority itself. The privlieges 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 authority, see Section 5 (Volume 2) 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 194 for a description of the TO clause.

WITH GRANT OPTION
If you grant the SYSADM system privilege, WITH GRANT OPTION is valid but unnecessary. It is unnecessary because whoever is granted SYSADM authority 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 GRANT OPTION.

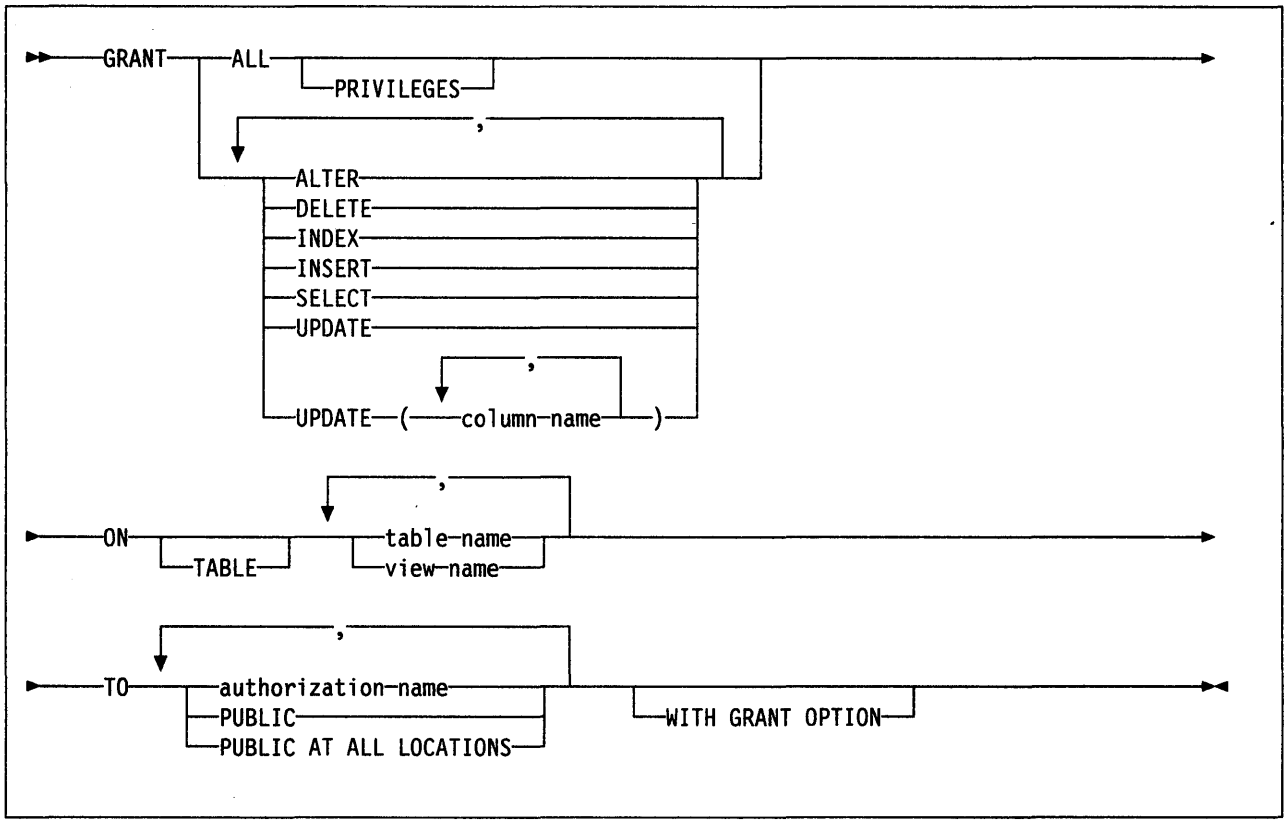
```
GRANT BSDS,RECOVER
  TO PARKER,SETRIGHT
  WITH GRANT OPTION;
```

Example 3: Grant TRACE privileges to all local users.

```
GRANT TRACE
  TO PUBLIC;
```

GRANT (TABLE or VIEW PRIVILEGES)

This form of the GRANT statement grants privileges on table and views.



Description

ALL or **ALL PRIVILEGES**
Grants all table or view privileges for which you have GRANT authority, for the table or view named in the ON clause. Does not include ALTER or INDEX 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. It cannot 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.

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. The UPDATE privilege on a column cannot be granted with the GRANT option. If UPDATE *column-name* and WITH GRANT OPTION are used together, the statement is executed, but a warning is issued and the UPDATE privilege on the columns is granted without GRANT authority.

ON or **ON TABLE**
Names the tables or views on which you are granting the privileges. The list may 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 current SQL authorization ID must have at least one privilege with the GRANT option.

TO
Refer to "GRANT" on page 194 for a description of the TO clause.

Examples

Example 1: Grant SELECT privileges on table DSN8220.EMP to user PULASKI.

```
GRANT SELECT
  ON DSN8220.EMP
  TO PULASKI
```

Example 2: Grant UPDATE privileges on columns EMPNO and WORKDEPT in table DSN8220.EMP to all local users.

```
GRANT UPDATE (EMPNO,WORKDEPT)
  ON TABLE DSN8220.EMP
  TO PUBLIC
```

Example 3: Grant all privileges on table DSN8220.EMP to users KWAN and THOMPSON, WITH GRANT OPTION.

```
GRANT ALL
  ON TABLE DSN8220.EMP
  TO KWAN,THOMPSON
  WITH GRANT OPTION;
```

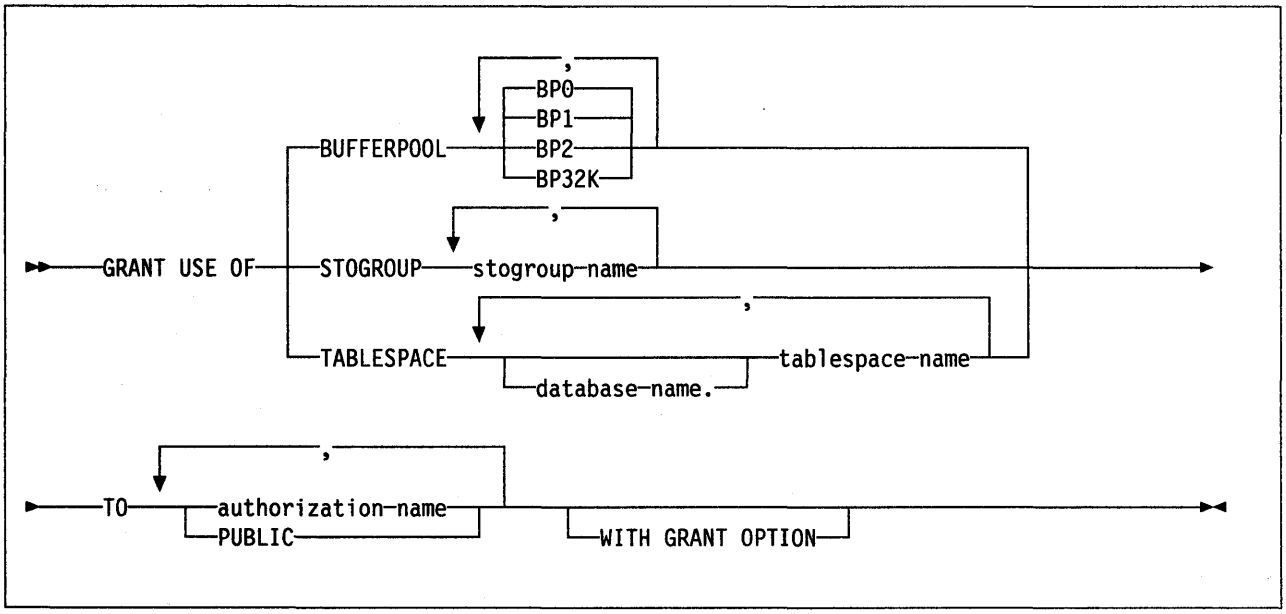
Example 4: Grant the SELECT and UPDATE privileges on the table JONES.DSNNETUSE to every user in the network.

```
GRANT SELECT, UPDATE
  ON TABLE JONES.DSNNETUSE
  TO PUBLIC AT ALL LOCATIONS
```

Please note that even with this grant, some network users may have no 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. Communications databases are described in Appendix D, "The Communications Database" on page 295. Controlling access is described in Section 5 (Volume 2) of *Administration Guide*.

GRANT (USE PRIVILEGES)

This form of the GRANT statement grants authority to use particular buffer pools, storage groups, or table spaces.



Description

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 name, you must have the USE privilege with GRANT authority.

BUFFERPOOL BP*n*
Grants the USE privilege on one or more buffer pools. The USE privilege for a buffer pool allows a user to name that buffer pool in a CREATE INDEX, CREATE TABLESPACE, ALTER INDEX, or ALTER TABLESPACE statement.

STOGROUP *stogroup-name*
Grants the USE privilege on one or more storage groups. The USE privilege for a storage group allows a user to name that storage group in a CREATE INDEX, CREATE TABLESPACE, ALTER INDEX, or ALTER TABLESPACE statement.

TABLESPACE *database-name.tablespace-name*
Grants the USE privilege on one or more table spaces. The USE privilege for a table space allows a user to name that table space in a CREATE TABLE statement. The default for *database-name* is DSNDB04.

TO
Refer to "GRANT" on page 194 for a description of the TO clause.

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 DSN8S22D in database DSN8D22A.

```
GRANT USE OF TABLESPACE
  DSN8D22A.DSN8S22D
  TO PUBLIC;
```

INCLUDE

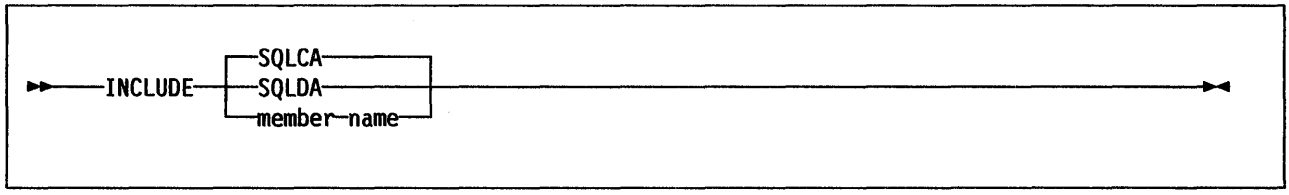
The INCLUDE statement inserts declarations or code into a source program.

Invocation

This statement can only be embedded in an application program. It is not an executable statement.

Authorization

None required.



Description

SQLCA
Indicates the description of an SQL Communication Area (SQLCA) is to be included. INCLUDE SQLCA must not be specified more than once in the same program. For a description of the SQLCA, see "SQL Communication Area (SQLCA)" on page 249.

In COBOL, INCLUDE SQLCA must be specified in the WORKING STORAGE section.

SQLDA
Indicates 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 252.

member-name
Names a member of the partitioned data set to be the library input when your program is precompiled. It must be a short, ordinary identifier.

The member may 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 program is precompiled, the INCLUDE statement is replaced by source statements. Thus the INCLUDE statement should be specified at a point in your program such that the resulting source statements are acceptable to the compiler.

The INCLUDE statement cannot refer to source statements that themselves contain INCLUDE statements.

The declarations that are generated by DCLGEN can be used in an application program by specifying the same member in the INCLUDE statement as in the DCLGEN LIBRARY parameter.

Example

Include an SQLCA in a program:

```
EXEC SQL INCLUDE SQLCA;
```

INSERT

The INSERT statement inserts rows into a local or remote table or view. Inserting a row into a view inserts the row into the table on which the view is based.

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 or not a view is used for the insertion:

If a view is not used, the privilege set defined below 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.

If a view is used, the privilege set must contain at least one of the following:

- The INSERT privilege on the view
- SYSADM authority.

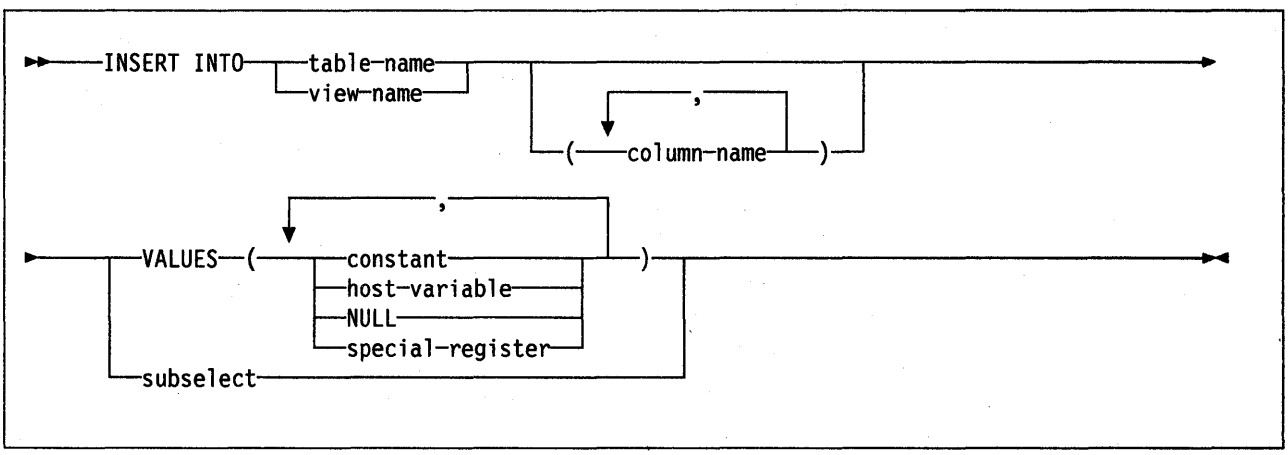
Note that the owner of a view, unlike the owner of a table, might not have INSERT authority on the view (or may have INSERT authority without being able to grant it to others). Indeed, the nature of the view itself may preclude its use for INSERT. For more on all this, see the discussion of authority under "CREATE VIEW" on page 161.

Note also that if a subselect is specified, the privilege set must include authority to execute the subselect. This implies that the privilege set must also include the SELECT privilege on every table and view identified in the subselect. The privilege may have been explicitly granted or may be inherent in another privilege. The SELECT privilege on a table or view is inherent in SYSADM authority and ownership of the table or view. The SELECT privilege on a table is also inherent in DBADM authority for its database.

For local execution of the statement: If the statement is embedded in a program, the privilege set is the privileges designated by the authorization ID of the owner of the plan. If the statement is dynamically prepared, the privilege set is the union of the privileges designated by each authorization ID of the process.

For remote execution of the statement: The privilege set consists of all those privileges recorded in the remote subsystem's catalog for a certain authorization ID. For a statement embedded in a program, this authorization ID is derived from the one that owns the plan. For a dynamically prepared statement, it is derived from the primary authorization ID of the process. The derived authorization ID may, of course, be equal to the original.

The derivation, known as "translation," is described in Section 5 (Volume 2) of *Administration Guide*. Controlling the procedure are the SYSIBM.SYSUSERNAMES tables in the local and remote communication databases. For a description of commmunication databases, see Appendix D, "The Communications Database" on page 295.



Description

INTO *table-name* or *view-name*

Names the table or view into which an insertion is to be made. The table or view must be described in the catalog of the DB2 subsystem identified by the implicitly or explicitly specified location-name. But it must not be a catalog table or any of the following types of view:

- A read-only view (for a description, see "CREATE VIEW" on page 161)
- A view of a catalog table
- A view with a column that is derived from a constant, function, or expression
- A view with two columns derived from the same column of the underlying table.

The referenced table or view can be remote. But it must be local if the process is attached to DB2 through the CICS or IMS/VS attachment facilities.

column-name

Lists the names of one or more columns for which you provide insert values. You may name the columns in any order. Each must belong to the table or view named, and you may not name the same column more than once. The column names must not be qualified.

If you omit the column list, you are implicitly using a list of all the columns in the order they exist in the table or view.

The implicit column list is established at bind time. Hence an INSERT statement embedded in an application program does not use any columns that might have been added to the table or view after bind time.

VALUES

Introduces one row of values to be inserted. The values of the row are the values of the keywords, constants, or host variables specified in the clause. NULL specifies the null value.

Each host variable you name must be a structure or variable that is described in your program in accordance with the rules for declaring host structures and variables. In the operational form of the statement, a reference to a structure has been 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. For a description of *special-register*, see "Special Registers" on page 41.

subselect
Inserts the rows of the result table of a subselect. There may be one, more than one, or none. If there is none, SQLCODE is set to +100.

(For an explanation of subselect, see Chapter 5, "Queries" on page 83.)

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 an INSERT statement has a subselect and the object table is self-referencing, the subselect must not return more than one row.

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.

- *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 default values.
- *Length*: If the insert value of a column is a number, the column must be a numeric column with the capacity to represent the integral part of the number. If the insert value of a column is a string, the column must either be a string column with a length attribute at least as great as the length of the string, or a date/time column if the string represents a date, time, or timestamp.
- *Assignment*: Insert values are assigned to columns in accordance with the assignment rules described in Chapter 3.
- *Validity*: If the table named, or the base table of the view named, has one or more unique indexes, each row inserted into the table must conform to the constraints imposed by those indexes. If either table has a field or validation procedure, each row inserted must conform to the constraints imposed by those procedures.

  If you name a view whose definition includes WITH CHECK OPTION, each row inserted 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 161.
- *Referential Integrity*: Each non-null insert value of a foreign key must be equal to some value of the primary key of the parent table in the relationship.

Notes

If you name a view whose definition does not include WITH CHECK OPTION, rows can be inserted 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.

After execution of an INSERT statement that is embedded within a program, the value of the third variable of the SQLERRD portion of the SQLCA indicates the number of rows that were inserted.

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, an inserted row can only be accessed by the application process that performed the insert.

Examples

Example 1: Insert values into table DSN8220.EMP.

```
INSERT INTO DSN8220.EMP
  VALUES ('000205','MARY','T','SMITH','D11','2866',
          '08-10-1981',42,16,'F','05-22-1956',16345)
```

Example 2: Load the temporary table SMITH.TEMPEMPL with data from table DSN8220.EMP.

```
INSERT INTO SMITH.TEMPEMPL
  SELECT *
  FROM DSN8220.EMP
```

Example 3: Load the temporary table SMITH.TEMPEMPL with data from Department D11 from DSN8220.EMP.

```
INSERT INTO SMITH.TEMPEMPL
  SELECT *
  FROM DSN8220.EMP
  WHERE WORKDEPT='D11'
```

LABEL ON

The LABEL ON statement adds or replaces labels in the local catalog descriptions of tables, views, aliases, or columns.

Invocation

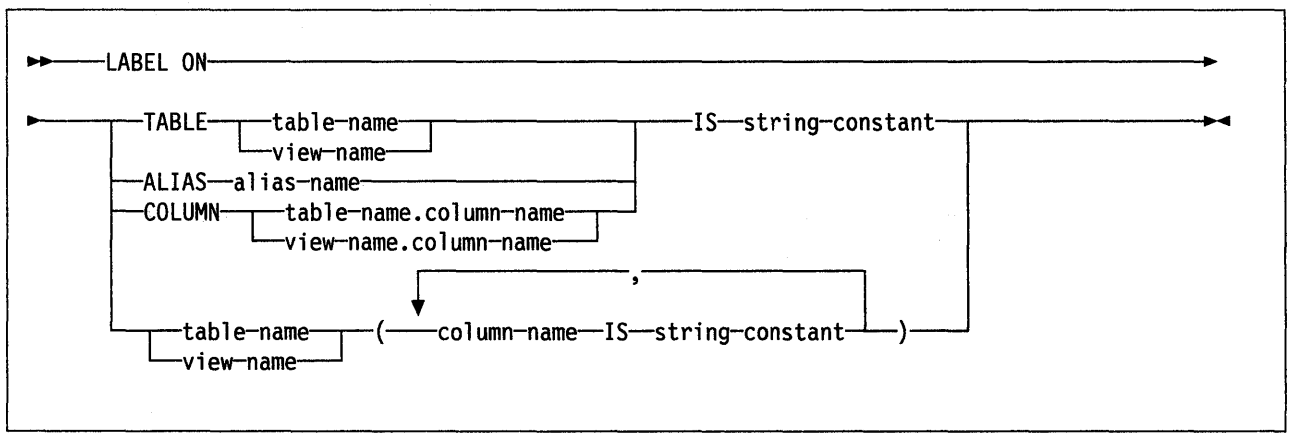
This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared.

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 (for tables only)
- SYSADM authority.

If the statement is embedded in a program, the privilege set is the privileges designated by the authorization ID of the owner of the plan. If the statement is dynamically prepared, the privilege set is the union of the privileges designated by each authorization ID of the process.



Description

TABLE
Indicates that the label is for a table or a view.

table-name or *view-name*
Must identify a table or view described in the local catalog. The label is placed into the LABEL column of the SYSIBM.SYSTABLES catalog table for the row that describes the table or view.

ALIAS
Indicates that the label is for an alias.

alias-name
Must identify an alias described in the local catalog. 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*
Is the name of the column, qualified by the name of the table or view in which it appears. The column named must be described in the local catalog. The label is placed in the LABEL column of the SYSIBM.SYSCOLUMNS catalog table in the row that describes the column.

To define a label for more than one column in a table or view, do not use TABLE or COLUMN. Give the table or view name and then, in parentheses, a list of this 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 be described in the catalog.

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 DSN8220.DEPT.

```
LABEL ON COLUMN DSN8220.DEPT.DEPTNO
  IS 'DEPARTMENT NUMBER'
```

Example 2: Enter labels on two columns in table DSN8220.DEPT.

```
LABEL ON DSN8220.DEPT
 (MGRNO IS 'MANAGER''S EMPLOYEE NUMBER',
  ADMRDEPT IS 'ADMINISTERING DEPARTMENT')
```

LOCK TABLE

The LOCK TABLE statement acquires a shared or exclusive lock on a local table or tablespace. The lock is not acquired if the process already holds an appropriate lock.

Invocation

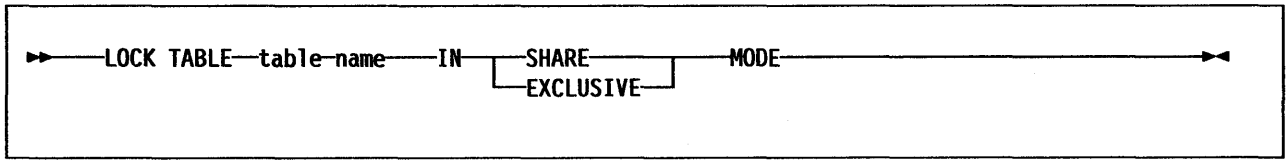
This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared.

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 authority.

If the statement is embedded in a program, the privilege set is the privileges designated by the authorization ID of the owner of the plan. If the statement is dynamically prepared, the privilege set is the union of the privileges designated by each authorization ID of the process.



Description

table-name
Names the table to be locked. The table must be a base table described in the catalog, but not a catalog table. The object named must be a table, not a view. The table must be local.

IN SHARE MODE
Acquires a shared lock for the application process. Until the lock is released, it prevents concurrent processes from executing insert, update, or delete operations on the table.

IN EXCLUSIVE MODE
Acquires an exclusive lock for the application process. Until the lock is released, it prevents concurrent processes from executing any operations on the table.

If the table is contained in an unsegmented table space, all tables in the table space are locked. If the table is contained in a segmented table space, only the identified table is locked.

The lock is acquired when the LOCK TABLE statement is executed. If the program contains embedded SQL statements (other than LOCK TABLE) that reference any table in the table space, and the plan was bound with RELEASE(DEALLOCATE),

the lock is not released until the process terminates.[7] In all other cases the lock is released by the next commit or rollback operation. If LOCK TABLE is executed using SPUFI, the lock is released by the next implicit or explicit commit or rollback operation. Thus LOCK TABLE should not be used if the SPUFI autocommit option is in effect.

Example

Obtain a lock on the table space containing table DSN8220.EMP. Do not allow other programs either to read or update the table.

```
LOCK TABLE DSN8220.EMP IN EXCLUSIVE MODE
```

[7] However, if the plan is bound with RELEASE(DEALLOCATE) and ACQUIRE(USE), and if none of the other embedded SQL statements is executed, the lock is released by the next commit or rollback operation.

OPEN

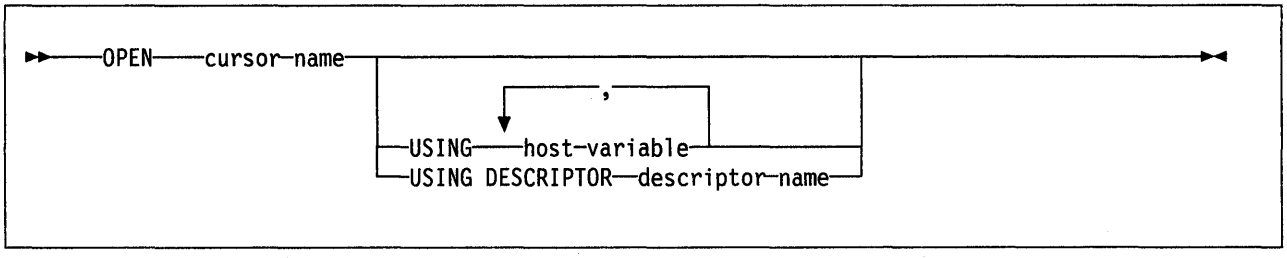
The OPEN statement opens a cursor so that it can be used to fetch rows from its result table.

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 165 for the authorization required to use a cursor.



Description

cursor-name

Identifies the cursor to be opened. The *cursor-name* must identify a declared cursor as explained in the Notes for the DECLARE CURSOR statement. When the OPEN statement is executed, the cursor must be in the closed state.

The SELECT statement associated with 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 identified SELECT statement has not been successfully prepared, 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 may be derived during the execution of the OPEN statement, and a temporary table created to hold them; or they may be derived during the execution of subsequent 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 state of the cursor is effectively "after the last row". A value of 100 for SQLCODE (empty result table) is returned on the first fetch rather than the open.

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 218.) 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 a structure or variable that is described in the program in accordance with the rules for declaring host structures and variables. When the statement is executed, a reference to a structure has been 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 host variables. The number of variables, as indicated by SQLD, must be the same as the number of parameter markers in the prepared statement and the length of the SQLDA, as indicated by SQLDABC, must be sufficient to describe that number of variables. 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 B, "SQLCA and SQLDA" on page 249.) When the host application is written in C, *descriptor-name* must be a pointer variable with pointer notation (for example, *sqlptr).

When the SELECT statement of the cursor is evaluated, each parameter marker is effectively replaced by the value of its corresponding host variable. The following rules apply to these parameter values:

- If the parameter marker appears as the operand of a unary minus, the parameter value must be a number. If it is not a double precision floating number, it is converted to double precision floating point before the operation is performed.
- If the parameter marker appears as the operand of an infix operator, the parameter value must be a number. If the data type, precision, and scale of the parameter value is not the same as the data type, precision, and scale of the other operand of the arithmetic operator, the parameter value is converted to the data type, precision, and scale of that operand before the operation is performed.
- If the parameter marker appears in a predicate, the parameter value must be compatible with the other operand of that predicate. If the parameter value is a string, it must not be longer than the other operand. If the parameter value is a number that does not have the same data type, precision, and scale as the other operand, the parameter value is converted to the data type, precision, and scale of that operand before the operation is performed. In the case of the BETWEEN predicate, the 'other operand' is the first operand that is specified solely as a column-name. If no operand of BETWEEN is specified solely as a column-name, the 'other operand' is the closest operand (in left to right order) that is not specified with a parameter marker. In the case of the IN predicate, the 'other operand' is the closest operand (in left to right order) that is not specified with a parameter marker.

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. Thus the effect is to override the host variables in the SELECT statement of the cursor with the host variables specified in the USING clause.

Notes

Closed state of cursors: All cursors in a program are in the closed state when:

- The program is initiated
- A new unit of recovery is initiated for the process in which the program executes.

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 temporary tables: If the result table of a cursor is not read-only, its rows are derived during the execution of subsequent FETCH statements. The same method may be used for a read-only result table. However, if a result table is read-only, DB2 may choose to use the temporary table method instead. With this method the entire result table is transferred to a temporary database during the execution of the OPEN statement. When a temporary table is used, the results of a program can differ in these two ways:

- An error can occur during OPEN that would otherwise not occur until some later FETCH statement.
- INSERT, UPDATE, and DELETE statements executed while the cursor is open cannot affect the result table.

Conversely, if a temporary table is not used, INSERT, UPDATE, and DELETE statements executed while the cursor is open can affect the result table if issued from the same program. Section 4 of *Application Programming and SQL Guide* describes how locking can be used to control the effect of INSERT, UPDATE, and DELETE operations executed by concurrent units of work. Your result table can also be affected by operations executed by your own unit of recovery, and 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. Thus a subsequent FETCH C may or may not retrieve the new row of T.

Example

The OPEN statement places the cursor at the beginning of the rows to be fetched.

```
   EXEC SQL DECLARE C1 CURSOR FOR
     SELECT DEPTNO, DEPTNAME, MGRNO FROM DSN8220.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

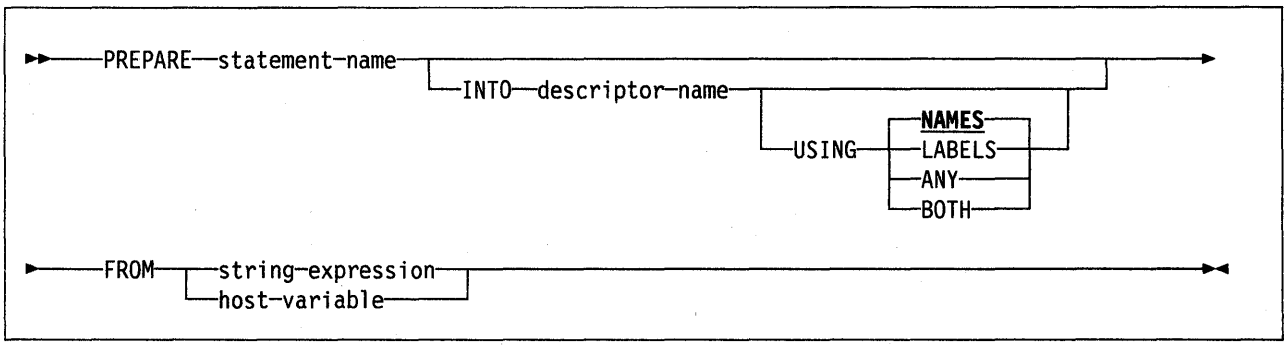
The PREPARE statement is used by application programs to dynamically prepare an SQL statement for execution. The PREPARE statement creates an executable SQL statement, called a *prepared statement*, from a character string form of the statement, called a *statement string*. The prepared statement is a named object that can be referred to only within the unit of recovery in which it is created.

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 83 for the authorization rules that apply when a SELECT statement is prepared.



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 176 for an explanation of the information that is placed in the SQLDA.

descriptor-name
Is SQLDA or the name of an SQLDA. When the host application is written in C, *descriptor-name* must be a pointer variable with pointer notation (for example, *sqlptr).

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 this case, two occurrences of SQLVAR per column will be needed to accommodate the additional information. To specify this expansion of the SQLVAR array, set SQLN to 2*n* on the PREPARE statement (where *n* is the number of columns in the result table). Then, on any later FETCH statement, set SQLN to *n*. 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.

FROM
Introduces the statement string. The statement string is the value of the specified *string-expression* or the identified *host-variable*.

string-expression
Is any PL/I expression that yields a character string. In all other host languages you must use *host variable*.

host-variable
Must identify a host variable that is described in the program in accordance with the rules for declaring character string variables. In C, COBOL and Assembler language, the host variable must be a varying-length string variable.

Notes

Rules for statement strings: The statement string must be one of the following SQL statements: ALTER, COMMENT ON, COMMIT, CREATE, DELETE, DROP, EXPLAIN, GRANT, INSERT, LABEL ON, LOCK TABLE, REVOKE, ROLLBACK, SET, UPDATE, or a *select-statement*.

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 may 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 "OPEN" on page 215 and "EXECUTE" on page 184.

Rules for parameter markers:

- Parameter markers must not appear:
  - 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 date/time arithmetic expression
  - In a SET statement
- At least one of the operands of the BETWEEN or IN predicates 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 may be the operand of a unary minus. For example, `WHERE C = -?`.

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.

Prepared statements can be referred to in the following kinds of statements, with the restrictions shown:

In...	The prepared statement ...
DESCRIBE	has no restrictions
DECLARE CURSOR	must be SELECT
EXECUTE	must *not* be SELECT

A prepared statement can be executed many times. Indeed, if a prepared statement is not executed more than once and does not contain parameter markers, it is more efficient to use the EXECUTE IMMEDIATE statement rather than the PREPARE and EXECUTE statements.

All prepared statements created by a unit of recovery are destroyed when the unit of recovery is terminated.

Example

In this example an INSERT statement with parameter markers is prepared and executed. S1 is a structure that corresponds to the format of DSN8220.DEPT.

```
EXEC SQL PREPARE DEPT_INSERT FROM
  'INSERT INTO DSN8220.DEPT VALUES(?,?,?,?)';

(Check for successful execution and read values into S1)

EXEC SQL EXECUTE DEPT_INSERT USING S1;
```

REVOKE

The REVOKE statement revokes privileges from authorization IDs. There is a separate form of the statement for each of these classes of privilege:

- Database
- Plan
- System
- Table
- Use.

The applicable objects are always local.

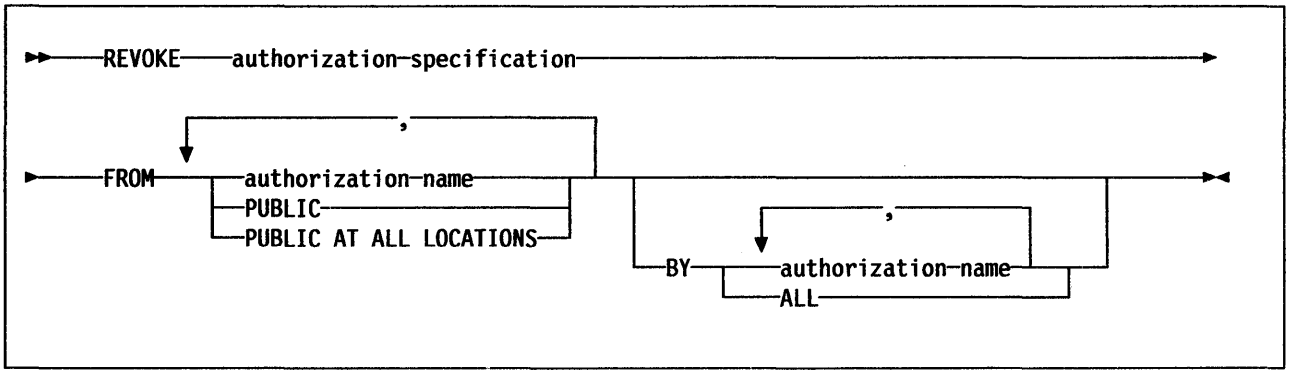
Invocation

This statement can be embedded in an application program or issued interactively. It is an executable statement that can 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 privilege set defined below must have been used to grant 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 privilege set must include SYSADM authority.

If the statement is embedded in a program, the privilege set is the privileges designated by the authorization ID of the owner of the plan. If the statement is dynamically prepared, the privilege set is the privileges designated by the SQL authorization ID of the process.



Description

authorization-specification
Names one or more privileges, in one of the formats described on the following pages. The same privilege must not be specified more than once.

FROM
Specifies from whom the privileges are revoked.

authorization-name
Lists one or more authorization IDs. Do not use the same authorization ID more than once.

You cannot use the ID of the REVOKE statement itself. (You cannot revoke privileges from yourself.)

PUBLIC
Revokes a grant of privileges to PUBLIC.

PUBLIC AT ALL LOCATIONS
Revokes a grant of privileges to PUBLIC ALL AT LOCATIONS.

BY
Lists others (grantors) who have granted privileges. REVOKE with BY revokes each named privilege that was explicitly granted to some named user by one of the named grantors. Only an authorization ID with SYSADM authority can use BY, even if the authorization ID names only itself in the BY clause.

If you omit BY, you must have granted each designated privilege to each of the designated 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.

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

Cascade Revoke: Revoking a privilege from a user may also cause that privilege to be revoked from other users. When Steve revokes privilege P from Randy, privilege P' is revoked from Claire if P' is exclusively dependent on Steve. P' is exclusively dependent on P if all of the following are true:

- P and P' are the same privilege.
- Randy granted that privilege to Claire.
- The grant could not have been made if Steve had not granted the privilege to Randy.
- Randy does not have install SYSADM authority.

Thus, when Steve revokes the privilege from Randy, that privilege is not revoked from Claire if, at the time that Randy granted it to Claire, Randy also held that privilege with the GRANT option from some other source, or had install SYSADM authority.

If the privilege is revoked from Claire the above rules apply, in turn, to grants that were made by Claire to any other users. The rules also apply to the implicit grants that are made as a result of a CREATE VIEW statement.

See Section 4 of *System and Database Administration Guide* for another example that describes cascading revokes.

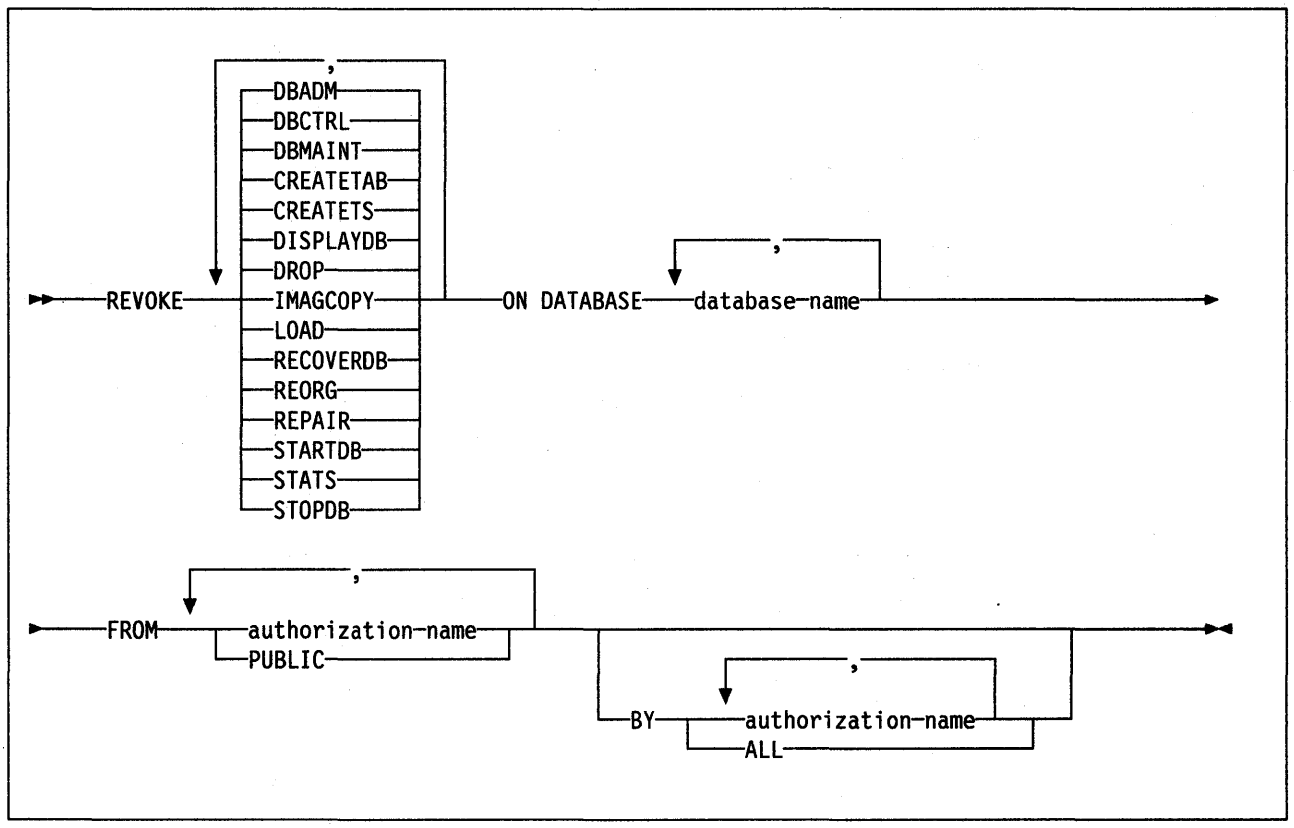
Revoking a SELECT or SYSADM privilege that was exercised to create a view causes the view to be dropped. Revoking any privilege that was exercised to create a plan invalidates the plan.

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.

Privileges belonging to an authority: You can revoke an administrative authority, but you cannot separately revoke the specific privileges inherent in that administrative authority.

REVOKE (DATABASE PRIVILEGES)

This form of the REVOKE statement revokes database privileges.



Description

Each keyword listed revokes the privilege described, but only as it applies to or within the databases named in the statement. The same keyword must not be specified more than once.

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 drop 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 to delete records from the SYSIBM.SYSCOPY catalog table and the SYSIBM.SYSGRNG directory table.

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*
Lists one or more 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). A database must not be identified more than once.

FROM
Refer to "REVOKE" on page 221 for a description of the FROM clause.

BY
Refer to "REVOKE" on page 221 for a description of the BY clause.

Examples

Example 1: Revoke drop privileges on database DSN8D22A from user PEREZ.

```
REVOKE DROP
  ON DATABASE DSN8D22A
  FROM PEREZ;
```

Example 2: Revoke repair privileges on database DSN8D22A from all local users. (Grants to specific users will not be affected.)

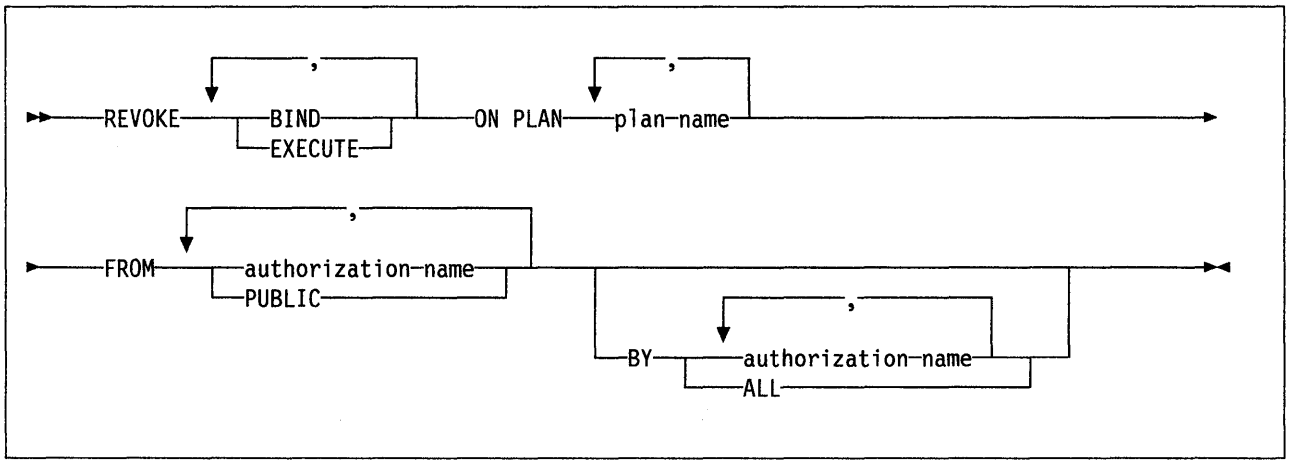
```
REVOKE REPAIR
  ON DATABASE DSN8D22A
  FROM PUBLIC;
```

Example 3: Revoke authority to create new tables and load tables in database DSN8D22A from users WALKER, PIANKA, and YOSHIMURA.

```
REVOKE CREATETAB,LOAD
  ON DATABASE DSN8D22A
  FROM WALKER,PIANKA,YOSHIMURA;
```

REVOKE (PLAN PRIVILEGES)

This form of the REVOKE statement revokes authority to bind or execute an application plan. Application plans are discussed in *Command and Utility Reference*.



Description

BIND
Revokes the privilege to use the BIND, REBIND, and FREE subcommands against the application plans named. BIND must not be specified more than once.

EXECUTE
Revokes the privilege to run programs that use the named application plans. EXECUTE must not be specified more than once.

ON PLAN *plan-name*
Lists one or more application plans on 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 221 for a description of the FROM clause.

BY
Refer to "REVOKE" on page 221 for a description of the BY clause.

Examples

Example 1: Revoke authority to bind plan DSN8CP22 from user JONES.

```
REVOKE BIND
  ON PLAN DSN8CP22
  FROM JONES;
```

Example 2: Revoke authority previously granted to all local users to bind and execute plan DSN8IP22. (Grants to specific users will not be affected.)

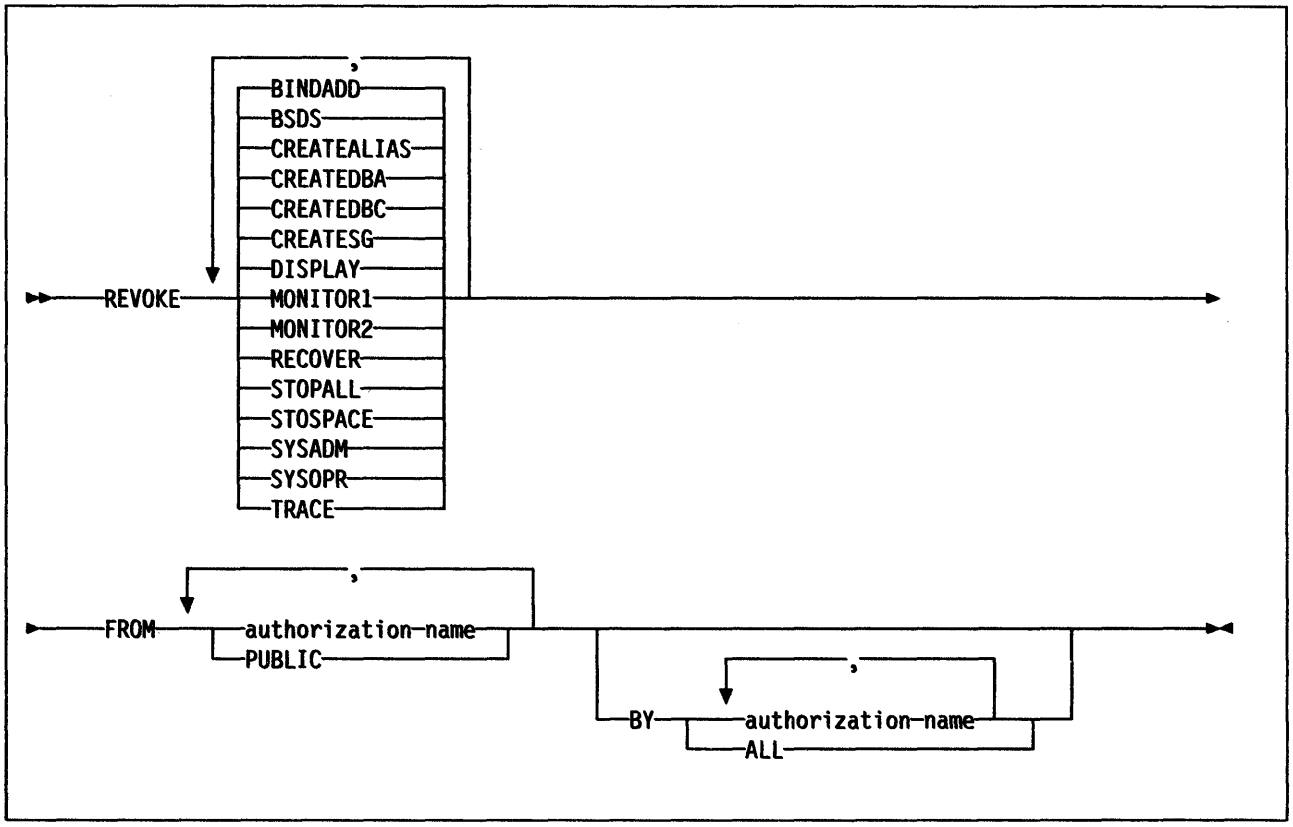
```
REVOKE BIND,EXECUTE
  ON PLAN DSN8IP22
  FROM PUBLIC;
```

Example 3: Revoke authority to execute plan DSN8CP22 from users ADAMSON and BROWN.

```
REVOKE EXECUTE
  ON PLAN DSN8CP22
  FROM ADAMSON,BROWN;
```

REVOKE (SYSTEM PRIVILEGES)

This form of the REVOKE statement revokes system privileges.



Description

Each keyword listed revokes the privilege described. The same keyword must not be specified more than once.

BINDADD
Revokes the privilege to create application plans using the BIND subcommand with the ADD option.

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 create new databases and acquire DBADM authority over those databases.

CREATEDBC
Revokes the privilege to create new databases and acquire DBCTRL authority over those databases.

CREATESG
Revokes the privilege to create new storage groups.

DISPLAY
Revokes the privilege to do the following:

- Use the -DISPLAY THREAD command for information on active threads within DB2
- Use the -DISPLAY DATABASE command for the status of all databases.
- Use the -DISPLAY LOCATION and -DISPLAY TRACE commands.

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 MONITOR2 privilege also implies having MONITOR1 privileges.)

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 privilege to have system administrator authority.

SYSOPR
Revokes the privilege to have system operator authority.

TRACE
Revokes the privilege to use the -MODIFY TRACE, -START TRACE, and -STOP TRACE commands.

FROM
Refer to "REVOKE" on page 221 for a description of the FROM clause.

BY
Refer to "REVOKE" on page 221 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.

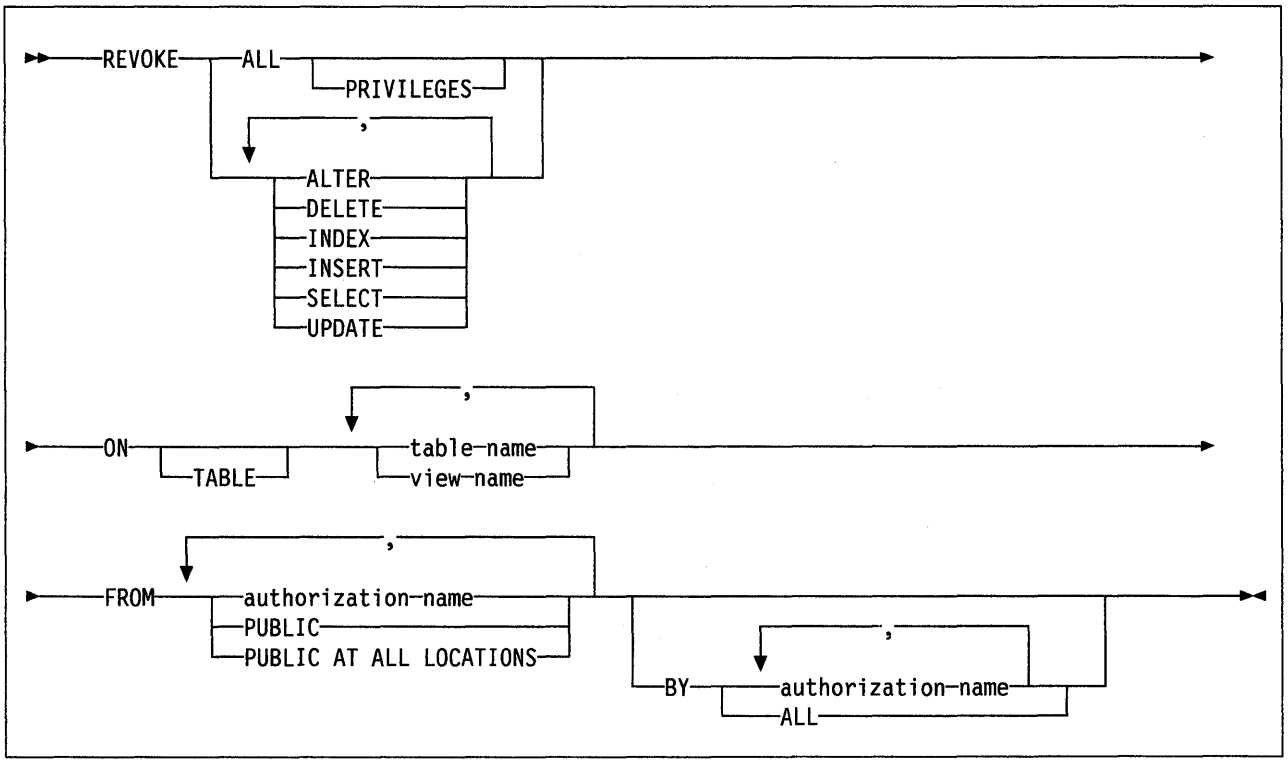
```
REVOKE BSDS,RECOVER
  FROM PARKER,SETRIGHT;
```

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.



ALL or **ALL PRIVILEGES**
Revokes all privileges held by an *authorization-name* for the specified tables or views.

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. The same keyword must not be specified more than once.

ALTER
Revokes the privilege to use the ALTER statement. ALTER cannot be revoked from PUBLIC AT ALL LOCATIONS.

DELETE
Revokes the privilege to use the DELETE statement.

INDEX
Revokes the privilege to use the CREATE INDEX statement. INDEX cannot be revoked from PUBLIC AT ALL LOCATIONS.

INSERT
Revokes the privilege to use the INSERT statement.

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.

UPDATE
Revokes the privilege to use the UPDATE statement. Note that a list of column names may 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 may consist of table names, view names, or a combination of the two. A table or view must not be identified more than once.

If BY is not specified, the authorization ID of the REVOKE statement must have been used to grant at least one of the designated privileges on each of the designated tables and views. (No single privilege need be granted on all tables and views.) If BY is specified, each designated grantor must satisfy the above requirement. In this case, the authorization ID of the statement need not satisfy the requirement unless it is one of the designated grantors.

FROM
Refer to "REVOKE" on page 221 for a description of the FROM clause.

BY
Refer to "REVOKE" on page 221 for a description of the BY clause.

Notes

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 will certainly occur if a table or view is specified twice after the keyword ON. One could also occur 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.

Examples

Example 1: Revoke SELECT privileges on table DSN8220.EMP from user PULASKI.

```
REVOKE SELECT
  ON TABLE DSN8220.EMP
  FROM PULASKI
```

Example 2: Revoke update privileges on table DSN8220.EMP, previously granted to all local users. Note that grants to specific users are not affected.

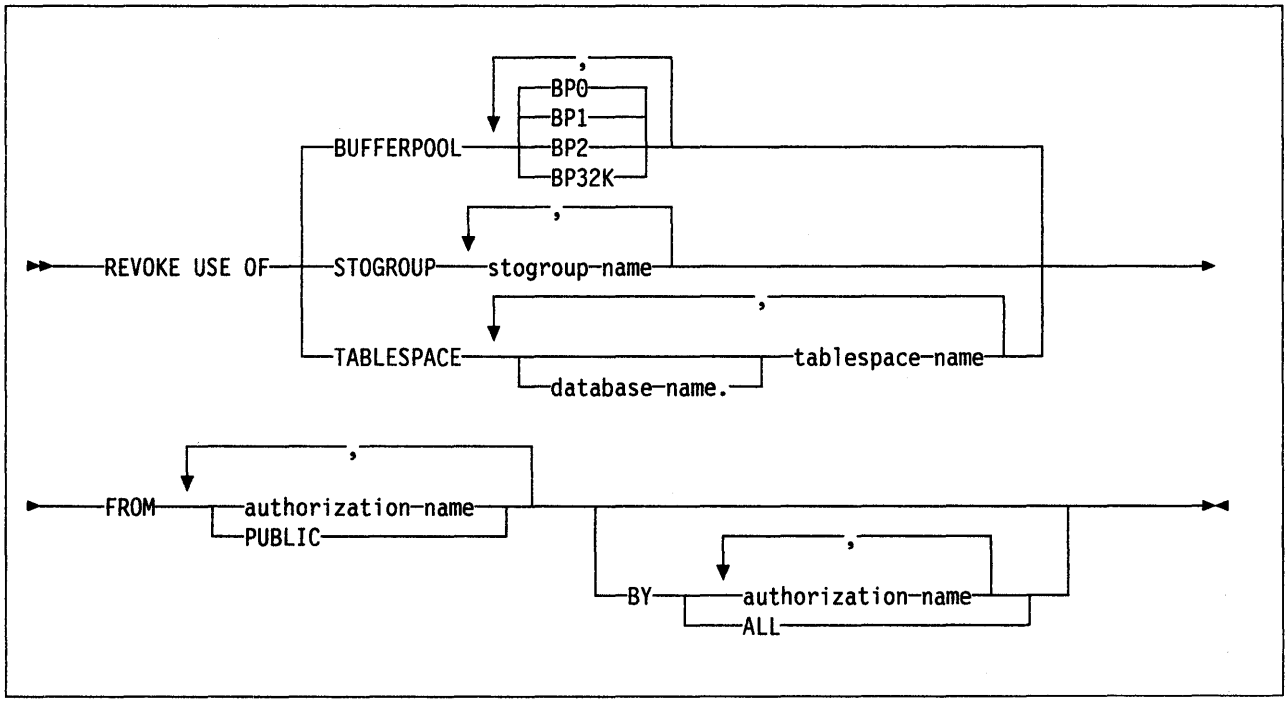
```
REVOKE UPDATE
  ON TABLE DSN8220.EMP
  FROM PUBLIC
```

Example 3: Revoke all privileges on table DSN8220.EMP, from users KWAN and THOMPSON.

```
REVOKE ALL
  ON TABLE DSN8220.EMP
  FROM KWAN,THOMPSON
```

REVOKE (USE PRIVILEGES)

This form of the REVOKE statement revokes authority to use particular buffer pools, storage groups, or table spaces.



Description

You can revoke privileges for only one type of object with each statement. Thus you may revoke the use of several table spaces with one statement, but not the use of a table space and a storage group. The same object must not be specified more than once.

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).

BUFFERPOOL BP*n*
Revokes the USE privilege on one or more buffer pools. The USE privilege for a buffer pool allows a user to name that buffer pool in a CREATE INDEX, CREATE TABLESPACE, ALTER INDEX, or ALTER TABLESPACE statement.

STOGROUP *stogroup-name*
Revokes the USE privilege on one or more storage groups. The USE privilege for a storage group allows a user to name that storage group in a CREATE INDEX, CREATE TABLESPACE, ALTER INDEX, or ALTER TABLESPACE statement.

TABLESPACE *database-name.tablespace-name*
Revokes the USE privilege on one or more table spaces. The USE privilege for a table space allows a user to name that table space in a CREATE TABLE statement. The default for *database-name* is DSNDB04.

FROM
Refer to "REVOKE" on page 221 for a description of the FROM clause.

BY

Refer to "REVOKE" on page 221 for a description of the BY clause.

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 DSN8S22D in the database DSN8D22A. The grant is to PUBLIC, that is, to everyone at the local DB2 subsystem. (Grants to specific users will not be affected.)

```
REVOKE USE OF TABLESPACE DSN8D22A.DSN8S22D
  FROM PUBLIC;
```

ROLLBACK

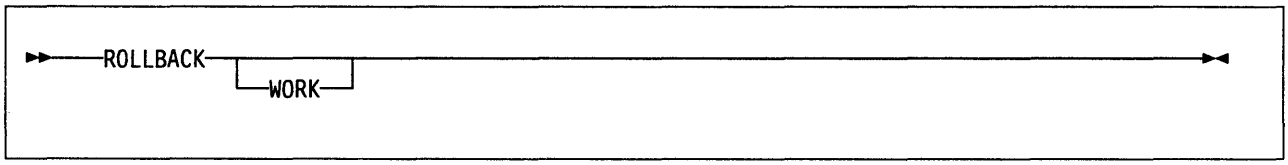
The ROLLBACK statement is used to terminate a unit of recovery and back out the database changes that were made by that unit of recovery.

Invocation

This statement can be embedded in an application program or it can be 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.



Description

The unit of recovery in which the ROLLBACK statement is executed is terminated and a new unit of recovery is initiated. All changes made by ALTER, COMMENT ON, CREATE, DELETE, DROP, EXPLAIN, GRANT, INSERT, LABEL ON, 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 the description of the LOCK TABLE statement for an explanation of the duration of those locks.) All cursors opened during the unit of recovery are closed. All statements prepared during the unit of recovery are destroyed, and any cursors associated with the prepared statements are invalidated.

ROLLBACK WORK has the same effect as ROLLBACK.

Notes

ROLLBACK WORK, rather than ROLLBACK, should be used to conform to the SAA definition of SQL. However, neither form of the statement can be used in the IMS or CICS environment. To execute 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 ROLLBACK statement.

In all DB2 environments, the abnormal termination of a process is an implicit rollback operation.

Example

Rollback all DB2 database changes made since the unit of recovery was initiated.

```
ROLLBACK WORK;
```

SELECT INTO

The SELECT INTO statement produces a result table consisting of 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 and does not assign values to the host variables. The tables or views identified in the statement can be local or remote.

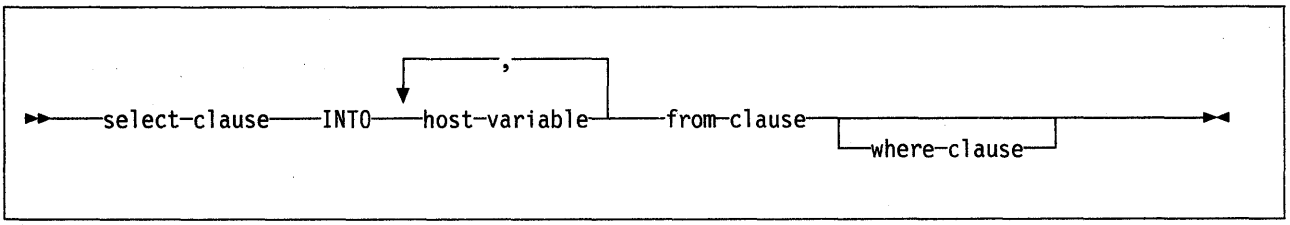
Invocation

This statement can only be embedded in an application program. It is an executable statement that cannot be dynamically prepared.

Authorization

The privileges held by the authorization ID of the owner of the plan 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.



Description

See Chapter 5, "Queries" on page 83 for a description of the *select-clause*, *from-clause*, and *where-clause*.

The *from-clause* must not identify a view that includes a *group-by-clause* or a *having-clause*.

INTO
Introduces a list of host variables.

host variable
Names a structure or variable that is described in the program under the rules for declaring host structures and variables. A reference to a structure is replaced by a reference to each of its variables before the statement is executed.

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 249.)

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/time value, the variable must be a character string variable of a minimum length as defined in Chapter 3. 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. 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 are assigned to all host variables, but the row that is the source of the values is undefined and not predictable.

Examples

Example 1: Put the maximum salary in DSN8220.EMP into the host variable MAXSALARY.

```
EXEC SQL SELECT MAX(SALARY)
  INTO :MAXSALRY
  FROM DSN8220.EMP;
```

Example 2: Put the row for employee 528671, from DSN8220.EMP, into the host structure EMPREC.

```
EXEC SQL SELECT * INTO :EMPREC
  FROM DSN8220.EMP
  WHERE EMPNO = '528671'
END-EXEC.
```

SET CURRENT SQLID

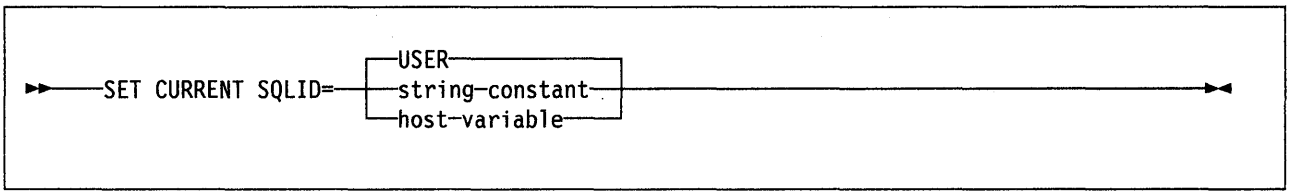
The SET CURRENT SQLID statement changes the value of the SQL authorization ID.

Invocation

The statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared.

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 process.



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 SQL authorization ID is:

- The authorization ID used for authorization checking on dynamically prepared CREATE, GRANT, and REVOKE SQL statements.
- The owner of a tablespace, 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.

The initital 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 reSIGNON request is received from a CICS transaction subtask or an IMS independent region
- The DB2 connection is terminated.

Example

Set the CURRENT SQLID to the primary authorization ID.

```
SET CURRENT SQLID=USER;
```

UPDATE

The UPDATE statement updates the values of specified columns in rows of a local or remote table or view. Updating a row of a view updates a row of its base table.

The forms of this statement are:

- 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 or not a view is used for the update:

If a view is not used, the privilege set defined below must include at least one of the following:

- The UPDATE privilege on the table
- Ownership of the table
- DBADM authority on the database containing the table
- SYSADM authority.

If a view is used, the privilege set must contain at least one of the following:

- The UPDATE privilege on the view
- SYSADM authority.

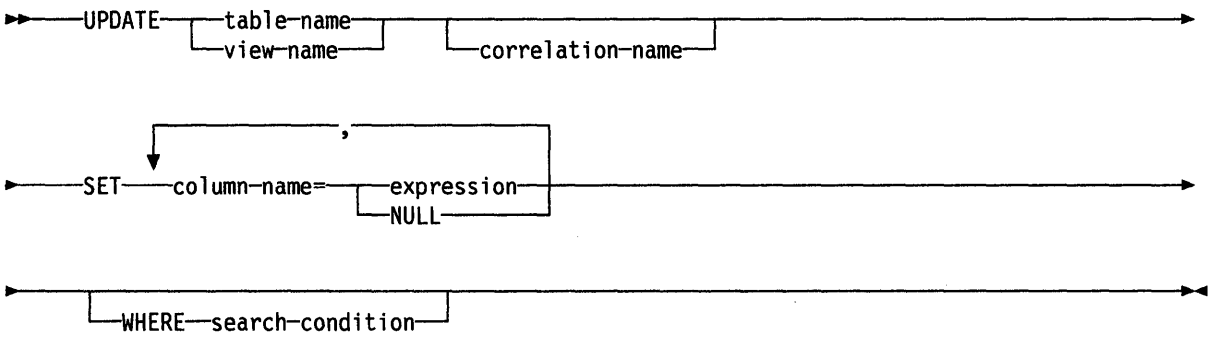
Note that the owner of a view, unlike the owner of a table, might not have UPDATE authority on the view (or may have UPDATE authority without being able to grant it to others). Indeed, the nature of the view itself may preclude its use for UPDATE. For more on all this, see the discussion of authority under "CREATE VIEW" on page 161.

For local execution of the statement: If the statement is embedded in a program, the privilege set is the privileges designated by the authorization ID of the owner of the plan. If the statement is dynamically prepared, the privilege set is the union of the privileges designated by each authorization ID of the process.

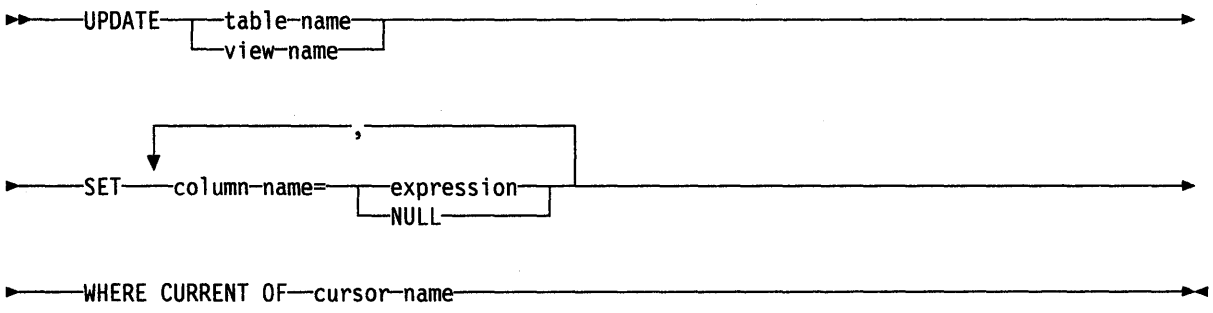
For remote execution of the statement: The privilege set consists of all those privileges recorded in the remote subsystem's catalog for a certain authorization ID. For a statement embedded in a program, this authorization ID is derived from the one that owns the plan. For a dynamically prepared statement, it is derived from the primary authorization ID of the process. The derived authorization ID may, of course, be equal to the original.

The derivation, known as "translation," is described in Section 5 (Volume 2) of *Administration Guide*. Controlling the procedure are the SYSIBM.SYSUSERNAMES tables in the local and remote communication databases. For a description of commmunication databases, see Appendix D, "The Communications Database" on page 295.

Searched UPDATE



Positioned UPDATE



Description

table-name or *view-name*
Is the name of the table or view to be updated. The name must identify a table or view described in the catalog, of the DB2 subsystem identified by the implicitly or explicitly specified *location-name*. But it must not identify a read-only view. A catalog table or a view of a catalog table may be identified if every column identified in the SET clause is an updatable column. If a column of a catalog table is updatable, then its description under Appendix C, "DB2 Catalog Tables" on page 257 will indicate that the column can be updated.

The referenced table or view can be remote. But it must be local if the process is attached to DB2 through the CICS or IMS/VS attachment facilities.

correlation-name
May be used within *search-condition* to designate the table or view. (For an explanation of *correlation-name*, see "Correlation Names" on page 44.)

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. The columns named must be present in the table or view to be updated.

column-name
Identifies a column to be updated. The *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. It must not identify

a column that is part of the key of a partitioned index or a view column that is derived from any part of the key of a partitioned index.

For a positioned update, allowable column names may 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 may be omitted under the conditions described in "The NOFOR Option: FOR UPDATE OF" on page 40.

expression or **NULL**
Indicates the new value of the column. The *expression* is any expression of the type described in Chapter 3. It must not include a column function. NULL specifies the null value.

A *column-name* in an expression must name a column of the named 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
Introduces a condition that indicates what rows are 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 as described in Chapter 3. Each *column-name* in the search condition, other than in a subquery, must name a column of the table or view. The search condition must not include a subquery so 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 a column to be updated is part of a primary key, the number of rows selected for update must not be greater than one.

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 a subquery with a correlated reference may have to 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 165.

If the UPDATE statement is embedded in a program, the DECLARE CURSOR statement must include a *select-statement* rather than a *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 165.)

When the UPDATE statement is executed, the cursor must be positioned on a row; that row is updated.

If a column to be updated is part of a primary key, WHERE CURRENT OF must not be used.

Notes

The successful or unsuccessful execution of a cursor-controlled 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.

Update values are assigned to columns under the assignment rules described in Chapter 3.

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 may 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 date/time value	Be a DATE, TIME, or TIMESTAMP column with the same date type or a character string column of an appropriate length as specified in Chapter 3

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
- Validation procedure
- Field procedure on an updated column.

The value of a primary key in a parent row must not be changed. A non-null update value of a foreign key must be equal to some value of the primary key of the parent table of the relationship.

If a view is used that is not defined using WITH CHECK OPTION, rows can be changed 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.

If a view is used that is defined using 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 161.

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.

If an update value violates any constraints, or if any other error occurs during the execution of the UPDATE statement, no rows are updated. The order which multiple rows are updated is undefined.

When 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 249.)

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, the updated row can only be accessed by the application process that performed the update.

Examples

The following examples refer to the sample table DSN8220.EMP. Each row in this table represents an employee of a fictional enterprise.

Example 1: Change employee 000190's telephone number to 3565 in DSN8220.EMP.

```
UPDATE DSN8220.EMP
  SET PHONENO='3565'
  WHERE EMPNO='000190'
```

Example 2: Give each member of department D11 a 100-dollar raise.

```
UPDATE DSN8220.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 DSN8220.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.

```
UPDATE DSN8220.EMP SET SALARY = 2 * SALARY WHERE CURRENT OF C1;
```

WHENEVER

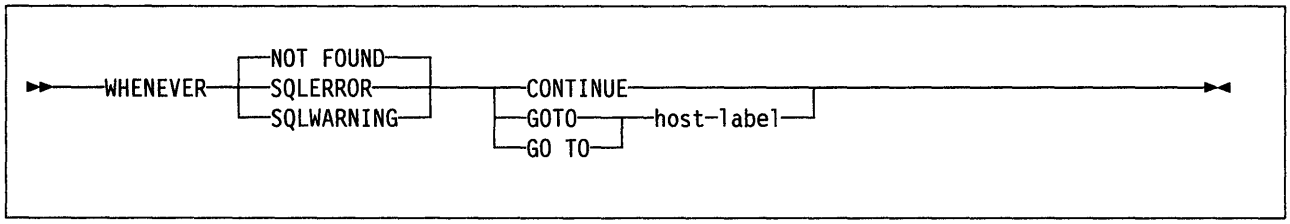
The WHENEVER statement specifies the host language statement to be executed when a specified exception condition occurs.

Invocation

This statement can only be embedded in an application program. It is not an executable statement.

Authorization

None required.



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 SQL return code of +100.

SQLERROR
Identifies any condition that results in a negative SQL return code.

SQLWARNING
Identifies any condition that results in a warning condition (SQLWARN0 is 'W'), or that results in a positive SQL return code other than +100.

The CONTINUE or GO TO clause is used to specify what is to happen 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 a *section-name* or an unqualified *paragraph-name*.

Notes

There are three types of WHENEVER statements:

WHENEVER NOT FOUND
WHENEVER SQLERROR
WHENEVER SQLWARNING

Every executable SQL statement in a 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 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.

Example

If an error is produced, go to HANDLERR. If a warning code is produced, continue with the normal flow of the program. If no results are found, go to ENDDATA.

```
EXEC SQL WHENEVER SQLERROR GOTO HANDLERR END-EXEC.
EXEC SQL WHENEVER SQLWARNING CONTINUE END-EXEC.
EXEC SQL WHENEVER NOT FOUND GO TO ENDDATA END-EXEC.
```

Appendix A. SQL Limits

The table below describes limits imposed by SQL.

Table 6 (Page 1 of 2). SQL Limits

ITEM	SQL LIMIT
Longest synonym, *correlation-name*, or name of a column, table, view, alias, or index.	18
Longest *authorization-name* or name of a plan, database, table space, storage group, or referential constraint.	8
Maximum number of columns in a table or view (the value depends on the complexity of the CREATE VIEW statement)	300 or fewer
Maximum row and record sizes for a table.	See "Notes" on page 151.
Maximum size of a VARCHAR or VARGRAPHIC column, in bytes	for 4K pages, 4046 for 32K pages, 32704
Largest INTEGER value	2147483647
Smallest INTEGER value	-2147483648
Largest SMALLINT value	32767
Smallest SMALLINT value	-32768
Largest FLOAT value	Approximately 7.2×10^{75}
Smallest FLOAT value	Approximately -7.2×10^{75}
Smallest positive FLOAT value	Approximately 5.4×10^{-79}
Largest negative FLOAT value	Approximately -5.4×10^{-79}
Largest DECIMAL value	999999999999999
Smallest DECIMAL value	-999999999999999
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
Maximum number of table names in an SQL statement. (In a complex SELECT, the number of tables that can be joined may be significantly less.)	15 or fewer, depending on the SQL statement.
Maximum total length of host and indicator variables pointed to in an SQLDA	32767 bytes

Table 6 (Page 2 of 2). SQL Limits	
ITEM	**SQL LIMIT**
Longest host variable used for insert or update	32704 bytes
Longest SQL statement	32765 bytes
Maximum number of elements in a select list	300
Maximum number of base tables in a view	16
Maximum number of predicates in a WHERE or HAVING clause	300
Maximum total length of columns in a GROUP BY clause	4000
Maximum total length of columns in an ORDER BY clause	4000
Maximum length of a string concatenation	32754 SBCS characters 16382 DBCS characters
Maximum number of columns in an index key	16
Longest index key for a non-partitioned table space, in bytes	254 less the number of key columns that allow nulls
Longest index key for a partitioned table space, in bytes	40 less the number of key columns that allow nulls
Maximum number of partitions in a partitioned table space	64
Maximum size of a partition, in gigabytes	4 for 1 to 16 partitions 2 for 17 to 32 partitions 1 for 33 to 64 partitions
Maximum number of volume IDs in a storage group	133

The limit for items not mentioned above is system storage. Examples of such items are most host variables in a source program, most indexes on a table, and most host variables, tokens, or functions in an SQL statement.

Appendix B. SQLCA and SQLDA

SQL Communication Area (SQLCA)

An SQLCA is a structure or collection of variables that is updated after each SQL statement executes. A program that contains executable SQL statements must provide exactly one SQLCA. In all host languages, 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.

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.

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 7 (Page 1 of 2). 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 'SQLCA'.
SQLCABC	sqlcabc	Not used.	INTEGER	Contains the length of the SQLCA: 136.
SQLCODE (See note 1)	sqlcode	SQLCOD	INTEGER	Contains the SQL return code. (See note 2) **Code** **Means** 0 Successful execution (though there may have been warning messages). positive Successful execution, but with an exception condition. negative Error condition.
SQLERRML	sqlerrml (See note 2.)	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	SQLTXT	VARCHAR (70)	Contains one or more tokens, separated by X'FF', that are substituted for variables in the descriptions of error conditions. (See note 2.)
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.
SQLERRD(1)	sqlerrd[1]	SQLERRD(1)	INTEGER	Contains a Relational Data System error code.
SQLERRD(2)	sqlerrd[2]	SQLERRD(2)	INTEGER	Contains a Data Manager error code.

Table 7 (Page 2 of 2). Fields of SQLCA				
Assembler, COBOL, or PL/I Name	**C Name**	**FORTRAN Name**	**Data Type**	**Purpose**
SQLERRD(3)	sqlerrd[3]	SQLERRD(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 -1 for a mass delete from a table in a segmented table space.
SQLERRD(4)	sqlerrd[4]	SQLERRD(4)	INTEGER	Contains timerons, a short floating point value that indicates a rough relative estimate of resources required. 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 may vary with changes to the statistics in the catalog. It is also subject to change between releases of DB2.
SQLERRD(5)	sqlerrd[5]	SQLERRD(5)	INTEGER	Contains the position or column of a syntax error for a PREPARE, or EXECUTE IMMEDIATE statement.
SQLERRD(6)	sqlerrd[6]	SQLERRD(6)	INTEGER	Contains a Buffer Manager error code.
SQLWARN0	sqlwarn0	SQLWARN(0)	CHAR(1)	Blank if all other indicators are blank; contains "W" if at least one other indicator contains "W."
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 columns is larger than the number of host variables.
SQLWARN4	sqlwarn4	SQLWRN(4)	CHAR(1)	Contains "W" if a prepared UPDATE or DELETE statement does not include a WHERE clause.
SQLWARN5	sqlwarn5	SQLWRN(5)	CHAR(1)	Contains "W" if the SQL statement was not executed because it is an SQL/DS statement that is not valid in DB2.
SQLWARN6	sqlwarn6	SQLWRN(6)	CHAR(1)	Contains "W" if an adjustment was made to a DATE or TIMESTAMP value to correct an invalid date resulting from an arithmetic operation.
SQLWARN7	sqlwarn7	SQLWRN(7)	CHAR(1)	Reserved for future use.
SQLEXT	sqlext	Not applicable	CHAR(8)	Reserved for future use.

Notes to Table 7:

1. With the precompiler option STDSQL(86) in effect, SQLCODE is replaced by SQLCADE in SQLCA. For more on this, see "Standard SQL Language" on page 39.
2. For the specific meanings of SQL return codes, see Section 2 of *Messages and Codes*. For the specific meaning of variables in error messages, see Section 3 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.

The Included SQLCA

The description of the SQLCA that is given by INCLUDE SQLCA is shown for each of the host languages.

In **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          IF ANY
SQLWARN1 DS   C          IF ANY
SQLWARN2 DS   C          IF ANY
SQLWARN3 DS   C          IF ANY
SQLWARN4 DS   C          IF ANY
SQLWARN5 DS   C          IF ANY
SQLWARN6 DS   C          IF ANY
SQLWARN7 DS   C          IF ANY
SQLEXT   DS   CL8
```

In **PL/I**:

```
DCL 1 SQLCA,
      2 SQLCAID      CHAR(8),
      2 SQLCABC      BIN FIXED(31),
      2 SQLCODE      BIN FIXED(31),
      2 SQLERRM      CHAR(70) VAR,
      2 SQLERRP      CHAR(8),
      2 SQLERRD(6)   BIN FIXED(31),
      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       CHAR(8);
```

In **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
      COMMON /SQLCA2/SQLERP,SQLWRN,SQLTXT,SQLEXT
*
```

In **COBOL**:

```
01 SQLCA.
   05 SQLCAID      PIC X(8).
   05 SQLCABC      PIC S9(9) COMPUTATIONAL.
   05 SQLCODE      PIC S9(9) COMPUTATIONAL.
   05 SQLERRM.
      49 SQLERRML  PIC S9(4) COMPUTATIONAL.
      49 SQLERRMC  PIC X(70).
   05 SQLERRP      PIC X(8).
   05 SQLERRD      OCCURS 6 TIMES
                   PIC S9(9) COMPUTATIONAL.
   05 SQLWARN.
      10 SQLWARN0  PIC X(1).
      10 SQLWARN1  PIC X(1).
      10 SQLWARN2  PIC X(1).
      10 SQLWARN3  PIC X(1).
      10 SQLWARN4  PIC X(1).
      10 SQLWARN5  PIC X(1).
      10 SQLWARN6  PIC X(1).
      10 SQLWARN7  PIC X(1).
   05 SQLEXT       PIC X(8).
```

In **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[8];
       unsigned char  sqlext[8];
};
#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]
#endif
struct sqlca sqlca;
```

SQL Descriptor Area (SQLDA)

An SQLDA is a collection of variables that is required for execution of the SQL DESCRIBE statement, and may optionally be used by the PREPARE, OPEN, FETCH, and EXECUTE statements. An SQLDA is used for dynamic SQL; it can be used in a DESCRIBE statement, modified with the addresses of host variables, and then reused in a FETCH statement. Section 2 of *Application Programming and SQL Guide* discusses ways to use the SQLDA.

The meaning of the information in an SQLDA depends on its use. In DESCRIBE and PREPARE, an SQLDA provides information to an application program about a prepared statement. In OPEN, EXECUTE, and FETCH, an SQLDA provides information to DB2 about host variables.

Description of Fields

The source language description of an SQLDA is different for PL/I, C and Assembler. The following description is based on the PL/I structure provided by the SQL INCLUDE statement.

An SQLDA consists of four variables followed by an arbitrary number of occurrences of a sequence of five variables collectively named SQLVAR. In OPEN, FETCH, and EXECUTE, each occurrence of SQLVAR describes a host variable. In DESCRIBE and PREPARE, they describe columns of a result table.

Table 8. Fields of SQLDA

PL/I Name	Data Type	Usage in DESCRIBE and PREPARE	Usage in FETCH, OPEN, or EXECUTE
SQLDAID	CHAR(8)	An "eye catcher" for storage dumps, containing 'SQLDA '.	Not used.
SQLDABC	INTEGER	Length of the SQLDA, equal to SQLN*44+16.	Same.
SQLN	SMALLINT	Total number of occurrences of SQLVAR.	Same.
SQLD	SMALLINT	The number of columns described by occurrences of SQLVAR.	The number of host variables described by occurrences of SQLVAR.

Fields in an Occurrence of SQLVAR

Table 9. Fields in SQLVAR

Name	Data Type	Usage in DESCRIBE and PREPARE	Usage in FETCH, OPEN, and EXECUTE
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 SQLTYPE on page 177	Tells the data type of the host variable and whether an indicator variable is provided. For a description of the type codes, see "Values of SQLTYPE" on page 254.
SQLLEN	SMALLINT	Defines the external length of a value from the column, as follows: **Data Type** — **Content** Character — Length attribute in bytes Graphic — Length attribute in *double-byte characters* Decimal — byte 1 = precision; byte 2 = scale Float — 4 (bytes) for single precision; 8 for double precision Smallint — 2 (bytes) Integer — 4 (bytes) Date — 10 (bytes) or LOCAL value Time — 8 (bytes) or LOCAL value Timestamp — 26 (bytes)	Same, for host variable.
SQLDATA	pointer	Undefined	Contains the address of the host variable.
SQLIND	pointer	Undefined	Contains the address of an associated indicator variable, if there is one; otherwise, not used.
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.	Not used.

Values of SQLTYPE

The table below lists allowable values of the SQLTYPE field of an SQLDA, and their meanings for FETCH, OPEN, and EXECUTE. See "DESCRIBE" on page 176 for a table that applies to the DESCRIBE statement. There are two values for each data type. The first value means that an indicator variable is not provided. The second value means an indicator variable is provided.

Value	Data Type	Indicator Variable
384/385	fixed-length character string	no/yes
388/389	fixed-length character string	no/yes
392/393	fixed-length character string	no/yes
448/449	varying-length character string	no/yes
452/453	fixed-length character string	no/yes
456/457	long character string	no/yes
460/461	varying-length, optionally null terminated, character string (C)	no/yes
464/465	varying-length graphic string	no/yes
468/469	fixed-length graphic string	no/yes
472/473	long graphic string	no/yes
480/481	floating-point	no/yes
484/485	decimal	no/yes
496/497	large integer	no/yes
500/501	small integer	no/yes
504/505	COBOL DISPLAY SIGN LEADING SEPARATE	no/yes

Note: On DESCRIBE, the type codes 384/385, 388/389, and 392/393 denote date, time, and timestamp, respectively. Host variables do not have date/time data types, so character string variables must be used to retrieve date/time values. Thus, when the SQLDA describes host variables, these type codes denote fixed-length character string variables.

The Included SQLDA

The description of the SQLDA that is given by INCLUDE SQLDA is shown for C, assembler and PL/I. 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 an appendix of *Application Programming and SQL Guide*.

In **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
```

In **PL/I**:

```
DCL 1 SQLDA BASED(SQLDAPTR),
      2 SQLDAID     CHAR(8),
      2 SQLDABC     BIN FIXED(31),
      2 SQLN        BIN FIXED,
      2 SQLD        BIN FIXED,
      2 SQLVAR      (SQLSIZE REF(SQLN)),
        3 SQLTYPE   BIN FIXED,
        3 SQLLEN    BIN FIXED,
        3 SQLDATA   PTR,
        3 SQLIND    PTR,
        3 SQLNAME   CHAR(30) VAR;
DCL SQLSIZE BIN FIXED;
DCL SQLDAPTR PTR;
```

In **C**, INCLUDE SQLDA specifies:

```
#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
```

Appendix C. DB2 Catalog Tables

This appendix is intended to help you to use the DB2 catalog. That interface is product-sensitive, as defined in "Statement of Purpose" on page 1.

DB2 maintains a set of tables called the DB2 catalog (database DSNDB06). The catalog tables describe such things as table spaces, tables, columns, indexes, privileges, and application plans. Data in the catalog tables is available to authorized users of DB2 through normal SQL query facilities; however, the catalog is primarily intended for use by DB2, and is therefore subject to change.

The catalog tables are updated by DB2 during normal operations in response to SQL data definition statements, SQL control statements, and certain commands and utilities. All columns of the catalog tables are defined as NOT NULL or NOT NULL WITH DEFAULT. For details about the record headers and column offsets, see Section 6 of *Diagnosis Guide and Reference*.

Links in the DB2 Catalog

The catalog contains objects called *links*.

A link connects a *parent table* to a *child table*. A row in the child table may be linked to only one row in the parent table, and the information recorded in the child row is dependent on the existence of the information in the parent row. On the other hand, a parent row may be linked to more than one child row.

For example, SYSIBM.SYSDATABASE records the existence of databases. It is linked to SYSIBM.SYSDBAUTH, which records the privileges users hold over databases. The information in SYSDBAUTH is dependent on the information in SYSDATABASE (no one can have a privilege on a database that does not exist); and a record in SYSDBAUTH is linked to only one record in SYSDATABASE, though a record in SYSDATABASE may be linked to many records in SYSDBAUTH. Hence SYSDATABASE is the parent and SYSDBAUTH is the child.

A table can be both a parent and a child. For example, SYSTABLES is a child of SYSTABLESPACE and a parent of SYSCOLUMNS.

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.

TABLE SPACE DSNDB06. ...	TABLE SYSIBM. ...	Refer to page	INDEX SYSIBM. ...	INDEX FIELDS
SYSCOPY	SYSCOPY	262	DSNUCH01	DBNAME.TSNAME.START_RBA
		262	DSNUCX01	DSNAME
SYSDBASE	SYSCOLAUTH	259		
	SYSCOLUMNS	260	DSNDCX01	TBCREATOR.TBNAME.NAME
	SYSFIELDS	267		
	SYSFOREIGNKEYS	268		
	SYSINDEXES	269	DSNDXX01	CREATOR.NAME
			DSNDXX02	DBNAME.INDEXSPACE
	SYSINDEXPART	271		
	SYSKEYS	272		
	SYSLINKS	273		
	SYSRELS	277	DSNDLX01	REFTBCREATOR.REFTBNAME
	SYSSYNONYMS	281	DSNDYX01	CREATOR.NAME
	SYSTABAUTH	282	DSNATX01	GRANTOR
			DSNATX02	GRANTEE.TCREATOR.TTNAME. GRANTEETYPE.UPDATECOLS. ALTERAUTH.DELETEAUTH. INDEXAUTH.INSERTAUTH. SELECTAUTH.UPDATEAUTH
	SYSTABLEPART	284		
	SYSTABLES	285	DSNDTX01	CREATOR.NAME
	SYSTABLESPACE	287	DSNDSX01	DBNAME.NAME
SYSDBAUT	SYSDATABASE	263	DSNDDH01	NAME
	SYSDBAUTH	264	DSNADH01	GRANTEE.NAME
			DSNADX01	GRANTOR.NAME
SYSGPAUT	SYSRESAUTH	278	DSNAGH01	GRANTEE.QUALIFIER. NAME.OPBTYPE
			DSNAGX01	GRANTOR.QUALIFIER. NAME.OPBTYPE
SYSGROUP	SYSSTOGROUP	280	DSNSSH01	NAME
	SYSVOLUMES	293		
SYSPLAN	SYSDBRM	266		
	SYSPLAN	274	DSNPPH01	NAME
	SYSPLANAUTH	275	DSNAPH01	GRANTEE.NAME.EXECUTEAUTH
			DSNAPX01	GRANTOR
	SYSPLANDEP	276	DSNGGX01	BCREATOR.BNAME.BTYPE
	SYSSTMT	279		
SYSUSER	SYSUSERAUTH	288	DSNAUH01	GRANTEE
			DSNAUX02	GRANTOR
SYSVIEWS	SYSVIEWDEP	290	DSNGGX02	BCREATOR.BNAME.BTYPE
	SYSVIEWS	291		
	SYSVLTREE	292		
	SYSVTREE	294	DSNVTH01	CREATOR.NAME

SYSIBM.SYSCOLAUTH Table

The SYSIBM.SYSCOLAUTH table records the UPDATE privileges held by users on individual columns of a table or view.

Column Name	Data Type	Description
GRANTOR	CHAR(8)	Authorization ID of the user who granted the privileges. Could also be PUBLIC or PUBLIC followed by an asterisk.[8]
GRANTEE	CHAR(8)	Authorization ID of the user who holds the privilege or the name of an application plan that uses the privilege. PUBLIC for a grant to PUBLIC. PUBLIC followed by an asterisk for a grant to PUBLIC AT ALL LOCATIONS.
GRANTEETYPE	CHAR(1)	Meaning: blank GRANTEE is an authorization ID P GRANTEE is an application plan
CREATOR	CHAR(8)	The authorization ID of the owner of the table or view on which the update privilege is held.
TNAME	VARCHAR(18)	The name of the table or view.
TIMESTAMP	CHAR(12)	Time at which the privilege was granted (internal timestamp format).
DATEGRANTED	CHAR(6)	Date the privilege was granted, in the form *yymmdd*.
TIMEGRANTED	CHAR(8)	Time the privilege was granted, in the form *hhmmssth*.
COLNAME	VARCHAR(18)	Name of the column to which the UPDATE privilege applies.
IBMREQD	CHAR(1)	Whether the row came from the basic machine readable material (MRM) tape: N no Y yes

[8] PUBLIC followed by an asterisk (PUBLIC*) denotes PUBLIC AT ALL LOCATIONS. For conditions under which GRANTOR can be PUBLIC or PUBLIC*, see Section 5 (Volume 2) of *Administration Guide*.

SYSIBM.SYSCOLUMNS Table

The SYSIBM.SYSCOLUMNS table contains one row for every column of each table and view.

Column Name	Data Type	Description
NAME	VARCHAR(18)	Name of the column.
TBNAME	VARCHAR(18)	Name of the table or view which contains the column.
TBCREATOR	CHAR(8)	Authorization ID of the owner of the table or view that contains the column.
COLNO	SMALLINT	Numerical place of the column in the table or view; for example 4 (out of 10).
COLTYPE	CHAR(8)	Type of 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
LENGTH	SMALLINT	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
SCALE	SMALLINT	Scale of decimal data. Zero if not a decimal column.
NULLS	CHAR(1)	Whether the column can contain null values: N no Y yes
COLCARD	INTEGER	Number of distinct values in the column. For non-indexed columns, the value is estimated using a probabilistic counting method. -1 if statistics have not been gathered. This is an updateable column.
HIGH2KEY	CHAR(8)	Second highest value of the column. Blank if statistics have not been gathered. If the key has a non-character data type, the data may not be printable. This is an updateable column.
LOW2KEY	CHAR(8)	Second lowest value of the column. Blank if statistics have not been gathered. If the key has a non-character data type, the data may not be printable. This is an updateable column.

Column Name	Data Type	Description
UPDATES	CHAR(1)	Whether the column can be updated: N no Y yes (The value is N only if the column is part of the key of a partitioned index or is derived from a function or expression. Thus the value can be Y for columns of a read-only view. N also applies to catalog columns that are not updateable.)
IBMREQD	CHAR(1)	Whether the row came from the basic machine-readable material (MRM) tape: N no Y yes
REMARKS	VARCHAR(254)	A character string provided by the user with the COMMENT ON statement.
DEFAULT	CHAR(1)	Whether the column has a default value (null or nonnull): N no Y yes
KEYSEQ	SMALLINT	The column's numerical position within the table's primary key. 0 if it is not part of a primary key.
FOREIGNKEY	CHAR(1)	Contains B if the column contains bit data. All other values indicate that the column does not contain bit data. This is an updateable column.
FLDPROC	CHAR(1)	Whether the column has a field procedure: N no Y yes This field is blank for views.
LABEL	VARCHAR(30)	The column label as given by a LABEL ON statement; otherwise it is an empty string.

SYSIBM.SYSCOPY Table

The SYSIBM.SYSCOPY table contains information needed for recovery.

Column Name	Data Type	Description
DBNAME	CHAR(8)	Name of the database.
TSNAME	CHAR(8)	Name of the table space.
DSNUM	INTEGER	Data set number within the table space. For partitioned table spaces, this corresponds to partition number.
ICTYPE	CHAR(1)	Operation type: F full image copy I incremental image copy P partial recovery point Q QUIESCE R LOAD REPLACE LOG(YES) W REORG LOG(NO) X REORG LOG(YES) Y LOAD LOG(NO) Z LOAD LOG(YES)
ICDATE	CHAR(6)	Date of the entry in the form *yymmdd*.
START_RBA	CHAR(6)	A 48-bit positive integer containing the relative byte location of a point in the DB2 recovery log. The indicated point is: • 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 other values of ICTYPE, the end of the log before the start of the RELOAD phase of the LOAD or REORG utility.
FILESEQNO	INTEGER	Tape file sequence number of the copy.
DEVTYPE	CHAR(8)	Device type the copy is on.
IBMREQD	CHAR(1)	Whether the row came from the basic machine-readable material (MRM) tape: N no Y yes
DSNAME	CHAR(44)	The name of the data set.
ICTIME	CHAR(6)	The time at which this row was inserted, in the form *hhmmss*. 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 DB2.
SHRLEVEL	CHAR(1)	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.
DSVOLSER	VARCHAR(1784)	The volume serial numbers of the data set. A list of 6-byte numbers separated by commas.
TIMESTAMP	TIMESTAMP	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 be deleted in subsequent DB2 releases.

SYSIBM.SYSDATABASE Table

The SYSIBM.SYSDATABASE table contains one row for each database, except for database DSNDB01.

Column Name	Data Type	Description
NAME	CHAR(8)	Database name.
CREATOR	CHAR(8)	Authorization ID of the owner of the database.
STGROUP	CHAR(8)	Name of the default storage group of the database; blank for a system database.
BPOOL	CHAR(8)	Name of the default buffer pool of the database; blank for a system database.
DBID	SMALLINT	Internal identifier of the database.
IBMREQD	CHAR(1)	Whether the row came from the basic machine-readable material (MRM) tape: N no Y yes
CREATEDBY	CHAR(8)	Primary authorization ID of the user who created the database.

SYSIBM.SYSDBAUTH Table

The SYSIBM.SYSDBAUTH table records the privileges held by users over databases.

Column Name	Data Type	Description
GRANTOR	CHAR(8)	Authorization ID of the user who granted the privileges.
GRANTEE	CHAR(8)	Authorization ID of the user who holds the privileges, or the name of an application plan that uses the privileges.
NAME	CHAR(8)	Database name.
TIMESTAMP	CHAR(12)	Time at which the privileges were granted (internal timestamp format).
DATEGRANTED	CHAR(6)	Date the privileges were granted; in the form *yymmdd*.
TIMEGRANTED	CHAR(8)	Time the privileges were granted; in the form *hhmmssth*.
GRANTEETYPE	CHAR(1)	Meaning: blank GRANTEE is an authorization ID P GRANTEE is an application plan
AUTHHOWGOT	CHAR(1)	Authorization level of the user from whom the privileges were received. **Note:** This is the authorization level used by the authorization subcomponent. It is not necessarily the highest authorization level of the grantor. blank not applicable C DBCTL D DBADM M DBMAINT S SYSADM
CREATETABAUTH	CHAR(1)	Whether the GRANTEE can create tables within the database: blank privilege is not held G privilege held with the GRANT option Y privilege is held without the GRANT option
CREATETSAUTH	CHAR(1)	Whether the GRANTEE can create table spaces within the database: blank privilege is not held G privilege held with the GRANT option Y privilege is held without the GRANT option
DBADMAUTH	CHAR(1)	Whether the GRANTEE has DBADM authority over the database: blank privilege is not held G privilege held with the GRANT option Y privilege is held without the GRANT option
DBCTRLAUTH	CHAR(1)	Whether the GRANTEE has DBCTRL authority over the database: blank privilege is not held G privilege held with the GRANT option Y privilege is held without the GRANT option
DBMAINTAUTH	CHAR(1)	Whether the GRANTEE has DBMAINT authority over the database: blank privilege is not held G privilege held with the GRANT option Y privilege is held without the GRANT option
DISPLAYDBAUTH	CHAR(1)	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
DROPAUTH	CHAR(1)	Whether the GRANTEE can drop the database: blank privilege is not held G privilege held with the GRANT option Y privilege is held without the GRANT option
IMAGCOPYAUTH	CHAR(1)	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

Column Name	Data Type	Description
LOADAUTH	CHAR(1)	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
REORGAUTH	CHAR(1)	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
RECOVERDBAUTH	CHAR(1)	Whether the GRANTEE can use the RECOVER and REPORT utilities on table spaces of the database: blank privilege is not held G privilege held with the GRANT option Y privilege is held without the GRANT option
REPAIRAUTH	CHAR(1)	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
STARTDBAUTH	CHAR(1)	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
STATSAUTH	CHAR(1)	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
STOPAUTH	CHAR(1)	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
IBMREQD	CHAR(1)	Whether the row came from the basic machine-readable material (MRM) tape: N no Y yes

SYSIBM.SYSDBRM Table

The SYSIBM.SYSDBRM table contains one row for each DBRM (database request module) of each application plan.

Column Name	Data Type	Description
NAME	CHAR(8)	Name of the DBRM.
TIMESTAMP	CHAR(8)	Time of precompilation in internal format.
PDSNAME	CHAR(44)	Name of the partitioned data set of which the DBRM is a member.
PLNAME	CHAR(8)	Name of the application plan of which this DBRM is a part.
PLCREATOR	CHAR(8)	Authorization ID of the owner of the application plan.
PRECOMPTIME	CHAR(8)	Time of precompilation in the form *hhmmssth*.
PRECOMPDATE	CHAR(6)	Date of precompilation in the form *yymmdd*.
QUOTE	CHAR(1)	Whether the SQL escape character is the apostrophe or the quotation mark: N apostrophe Y quotation mark
COMMA	CHAR(1)	Whether the decimal point is the period or the comma: N period Y comma
HOSTLANG	CHAR(1)	The host language used: B assembler language C COBOL D C F FORTRAN P PL/I 2 VS COBOL II
IBMREQD	CHAR(1)	Whether the row came from the basic machine-readable material (MRM) tape: N no Y yes
CHARSET	CHAR(1)	Indicates the character set in use when the program was precompiled: K Katakana A Alphanumeric
MIXED	CHAR(1)	Indicates if the MIXED option was in effect when the program was precompiled: N no Y yes

SYSIBM.SYSFIELDS Table

The SYSIBM.SYSFIELDS table contains one row for every column that has a field procedure. It may also contain up to ten additional rows for every column that serves as the first column in an index key. The additional rows, which contain statistics, may be added when the index is scanned by the RUNSTATS utility. The column for the added rows need not have a field procedure.

Column Name	Data Type	Description
TBCREATOR	CHAR(8)	Authorization ID of the owner of the table that contains the column.
TBNAME	VARCHAR(18)	Name of the table that contains the column.
COLNO	SMALLINT	Numerical place of this column in the table.
NAME	VARCHAR(18)	Name of the column.
FLDTYPE	CHAR(8)	Data type of the encoded values in the field. 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
LENGTH	SMALLINT	The length attribute of the field; or, for a decimal field, its precision. The number does not include the internal prefixes that may 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
SCALE	SMALLINT	Scale if FLDTYPE is DECIMAL; otherwise, 0.
FLDPROC	CHAR(8)	For a row describing a field procedure, the name of the procedure. Blank for a statistical row.
WORKAREA	SMALLINT	For a row describing a field procedure, the size, in bytes, of the work area required for the encoding and decoding of the procedure. 0 for a statistical row.
IBMREQD	CHAR(1)	Whether the row came from the basic machine-readable material (MRM) tape: N no Y yes
EXITPARML	SMALLINT	For a row describing a field procedure, the length of the field procedure parameter value block. For a statistical row, the percentage of times, times 100 that the column has the value contained in EXITPARM.
PARMLIST	VARCHAR(254)	For a row describing a field procedure, the parameter list following FIELDPROC in the statement that created the column, with nonsignificant blanks removed. For a statistical row, a string of length zero.
EXITPARM	VARCHAR(1530)	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). For a statistical row, the column value whose frequency appears in EXITPARML.

SYSIBM.SYSFOREIGNKEYS Table

The SYSIBM.SYSFOREIGNKEYS table contains one row for every column of every foreign key.

Column Name	Data Type	Description
CREATOR	CHAR(8)	Authorization ID of the owner of the table that contains the column.
TBNAME	VARCHAR(18)	Name of the table that contains the column.
RELNAME	CHAR(8)	Constraint name for the constraint for which the column is part of the foreign key.
COLNAME	VARCHAR(18)	Name of the column.
COLNO	SMALLINT	Numerical place of the column in its table.
COLSEQ	SMALLINT	Numerical place of the column in the foreign key.
IBMREQD	CHAR(1)	Whether the row came from the basic machine-readable material (MRM) tape: Y yes N no

SYSIBM.SYSINDEXES Table

The SYSIBM.SYSINDEXES table contains one row for every index.

Column Name	Data Type	Description
NAME	VARCHAR(18)	Name of the index.
CREATOR	CHAR(8)	Authorization ID of the owner of the index.
TBNAME	VARCHAR(18)	Name of the table on which the index is defined.
TBCREATOR	CHAR(8)	Authorization ID of the owner of the table.
UNIQUERULE	CHAR(1)	Whether the index is unique: D no (duplicates are allowed) U yes P primary index (unique)
COLCOUNT	SMALLINT	The number of columns in the key.
CLUSTERING	CHAR(1)	Whether CLUSTER was specified when the index was created: N no Y yes
CLUSTERED	CHAR(1)	Whether the table is actually clustered by the index: N no: 95% or fewer of the rows are in clustering order. Y yes: More than 95% of the rows are in clustering order. The entry can be changed by the RUNSTATS utility.
DBID	SMALLINT	Internal identifier of the database.
OBID	SMALLINT	Internal identifier of the index fan set descriptor.
ISOBID	SMALLINT	Internal identifier of the index page set descriptor.
DBNAME	CHAR(8)	Name of the database that contains the index.
INDEXSPACE	CHAR(8)	Name of the index space.
FIRSTKEYCARD	INTEGER	Number of distinct values of the first key column. -1 before statistics are gathered. This is an updateable column.
FULLKEYCARD	INTEGER	Number of distinct values of the key. -1 before statistics are gathered. This is an updateable column.
NLEAF	INTEGER	Number of active leaf pages in the index. -1 before statistics are gathered. This is an updateable column.
NLEVELS	SMALLINT	Number of levels in the index tree. If the index is partitioned, it is the number of levels in the index tree. -1 before statistics are gathered. This is an updateable column.
BPOOL	CHAR(8)	Name of the buffer pool used for the index.
PGSIZE	SMALLINT	Size of subpages in the index: 256, 512, 1024, 2048, or 4096
ERASERULE	CHAR(1)	Whether the data sets are erased when dropped. The value is meaningless if the index is partitioned. N no Y yes
DSETPASS	CHAR(8)	The password for the data sets of the index.
CLOSERULE	CHAR(1)	Whether the data sets are closed when the index is not in use: N no Y yes
SPACE	INTEGER	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, and different storage groups have been specified, this is the number of kilobytes of DASD storage allocated to the partition as determined by the last execution of the STOSPACE utility.
IBMREQD	CHAR(1)	Whether the row came from the basic machine-readable material (MRM) tape: N no Y yes C V2R1 dependency indicator; not from MRM tape.

Column Name	Data Type	Description
CLUSTERRATIO	SMALLINT	Percentage of rows that are in clustering order. 0 before statistics are gathered. This column is updateable.
CREATEDBY	CHAR(8)	Primary authorization ID of the user who created the index.

SYSIBM.SYSINDEXPART Table

The SYSIBM.SYSINDEXPART table contains one row for each unpartitioned index and one row for each partition of a partitioned index.

Column Name	Data Type	Description
PARTITION	SMALLINT	Partition number; 0 if index is not partitioned.
IXNAME	VARCHAR(18)	Name of the index.
IXCREATOR	CHAR(8)	Authorization ID of the owner of the index.
PQTY	INTEGER	Primary space allocation in units of 4K-byte storage blocks. Zero if a storage group is not used.
SQTY	SMALLINT	Secondary space allocation in units of 4K-byte storage blocks. Zero if a storage group is not used.
STORTYPE	CHAR(1)	Type of storage allocation: E explicit and STORNAME names a ICF catalog I implicit and STORNAME names a storage group
STORNAME	CHAR(8)	Name of storage group or ICF catalog used for space allocation.
VCATNAME	CHAR(8)	Name of ICF catalog used for space allocation.
CARD	INTEGER	Number of rows referenced by the index or partition. -1 if statistics not gathered.
FAROFFPOS	INTEGER	Number of referenced rows far from optimal position because of an insert into a full page. -1 if statistics not gathered.
LEAFDIST	INTEGER	100 times the average number of pages between successive leaf pages of the index. -1 if statistics not gathered.
NEAROFFPOS	INTEGER	Number of referenced rows near, but not at optimal position, because of an insert into a full page.
IBMREQD	CHAR(1)	Whether the row came from the basic machine-readable material (MRM) tape: N no Y yes
LIMITKEY	VARCHAR(512)	The limit key of the partition in an internal format. 0 if the index is not partitioned.
FREEPAGE	SMALLINT	The number of pages that are loaded before a page is left as free space.
PCTFREE	SMALLINT	The percentage of each subpage or nonleaf page that is left as free space.

SYSIBM.SYSKEYS Table

The SYSIBM.SYSKEYS table contains one row for each column of an index key.

Column Name	Data Type	Description
IXNAME	VARCHAR(18)	Name of the index.
IXCREATOR	CHAR(8)	Authorization ID of the owner of the index.
COLNAME	VARCHAR(18)	Name of the column of the key.
COLNO	SMALLINT	Numerical position of the column in the row; for example 4 (out of 10).
COLSEQ	SMALLINT	Numerical position of the column in the key; for example 4 (out of 10).
ORDERING	CHAR(1)	Order of the column in the key: A ascending D descending
IBMREQD	CHAR(1)	Whether the row came from the basic machine-readable material (MRM) tape: N no Y yes

SYSIBM.SYSLINKS Table

The SYSIBM.SYSLINKS table contains one row for every link in the DB2 catalog.

Column Name	Data Type	Description
CREATOR	CHAR(8)	Authorization ID of the owner of the child table of the link.
TBNAME	VARCHAR(18)	Name of the child table in the link.
LINKNAME	CHAR(8)	Name of the link.
PARENTNAME	VARCHAR(18)	Name of the parent table in the link.
PARENTCREATOR	CHAR(8)	Authorization ID of the owner of the parent table of the link.
CHILDSEQ	SMALLINT	Cluster order of the child table within its parent table.
DBNAME	CHAR(8)	Name of the database containing the link.
DBID	SMALLINT	Internal DB2 identifier of the database.
OBID	SMALLINT	Internal DB2 identifier of the link.
COLCOUNT	SMALLINT	Number of columns in the ordering key for the link. 0 if there is no ordering key.
INSERTRULE	CHAR(1)	Type of insert rule for the link: F FIRST L LAST O ONE U UNIQUE
IBMREQD	CHAR(1)	Whether the row came from the basic machine-readable material (MRM) tape: N no Y yes

SYSIBM.SYSPLAN Table

The SYSIBM.SYSPLAN table contains one row for each application plan.

Column Name	Data Type	Description
NAME	CHAR(8)	Name of the application plan.
CREATOR	CHAR(8)	Authorization ID of the owner of the application plan.
BINDDATE	CHAR(6)	Date on which the most recent BIND or REBIND was performed in the form *yymmdd*.
VALIDATE	CHAR(1)	Whether validity checking can be deferred until run time: B all checking must be performed during BIND R checking is deferred if tables, views, or privileges do not exist at bind time
ISOLATION	CHAR(1)	The isolation level: R repeatable read S cursor stability
VALID	CHAR(1)	Whether the application plan is valid (whether it can be run without rebinding): N no Y yes A table or table space has been altered, but no rebinding is needed.
OPERATIVE	CHAR(1)	Whether the application plan can be allocated: N no; an explicit BIND or REBIND is required before the plan can be allocated Y yes
BINDTIME	CHAR(8)	Time of the BIND in the form *hhmmssth*.
PLSIZE	INTEGER	Size of the base section[9] of the plan, in bytes. Used by DB2 to allocate storage for the control structure.
IBMREQD	CHAR(1)	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
AVGSIZE	INTEGER	Average size, in bytes, of those sections[9] of the plan that contain DML statements processed at bind time.
ACQUIRE	CHAR(1)	When resources are acquired: A at allocation U at first use
RELEASE	CHAR(1)	When resources are released: C at commit D at deallocation
EXREFERENCE	CHAR(1)	Not used. All values are N.
EXSTRUCTURE	CHAR(1)	Not used. All values are N.
EXCOST	CHAR(1)	Not used. All values are N.
EXPLAN	CHAR(1)	Whether the plan was bound with EXPLAIN YES: N no Y yes
EXPREDICATE	CHAR(1)	Not used. All values are N.
BOUNDBY	CHAR(8)	The primary authorization ID of the binder of the plan.

[9] Plans are divided into *sections*. The base section of the plan must be in the EDM pool during the entire time the application program is executing. Other sections of the plan, corresponding roughly to sets of related SQL statements, are brought into the pool as needed.

SYSIBM.SYSPLANAUTH Table

The SYSIBM.SYSPLANAUTH table records the privileges held by users over application plans.

Column Name	Data Type	Description
GRANTOR	CHAR(8)	Authorization ID of the user who granted the privileges.
GRANTEE	CHAR(8)	Authorization ID of the user who holds the privileges, or name of a plan that uses the privileges.
NAME	CHAR(8)	Name of the application plan on which the privileges are held.
TIMESTAMP	CHAR(12)	Time at which the privileges were granted (internal timestamp format).
DATEGRANTED	CHAR(6)	Date the privileges were granted; in the form *yymmdd*.
TIMEGRANTED	CHAR(8)	Time the privileges were granted; in the form *hhmmssth*.
GRANTEETYPE	CHAR(1)	Meaning: blank GRANTEE is an authorization ID P GRANTEE is an application plan
AUTHHOWGOT	CHAR(1)	Authorization level of the user from whom the privileges were received. **Note:** This is the authorization level used by the authorization subcomponent. It is not necessarily the highest authorization level of the grantor. blank not applicable C DBCTL D DBADM M DBMAINT S SYSADM
BINDAUTH	CHAR(1)	Whether the GRANTEE can use the BIND, REBIND, or FREE commands against the plan: blank privilege is not held G privilege is held with the GRANT option Y privilege is held without the GRANT option
EXECUTEAUTH	CHAR(1)	Whether the GRANTEE can run programs that use the application plan: blank privilege is not held G privilege is held with the GRANT option Y privilege is held without the GRANT option
IBMREQD	CHAR(1)	Whether the row came from the basic machine-readable material (MRM) tape: N no Y yes

SYSIBM.SYSPLANDEP Table

The SYSIBM.SYSPLANDEP table records the dependencies of plans on tables, views, aliases, synonyms, table spaces, and indexes.

Column Name	Data Type	Description
BNAME	VARCHAR(18)	Name of an object the plan is dependent on.
BCREATOR	CHAR(8)	If BNAME is a table space, its database. Otherwise, the authorization ID of the owner of BNAME.
BTYPE	CHAR(1)	Type of object BNAME: A alias I index R table space S synonym T table V view
DNAME	CHAR(8)	Name of the plan.
IBMREQD	CHAR(1)	Whether the row came from the basic machine-readable material (MRM) tape: N no Y yes

SYSIBM.SYSRELS Table

The SYSIBM.SYSRELS table contains one row for every referential constraint.

Column Name	Data Type	Description
CREATOR	CHAR(8)	Authorization ID of the owner of the dependent table of the referential constraint.
TBNAME	VARCHAR(18)	Name of the dependent table of the referential constraint.
RELNAME	CHAR(8)	Constraint name
REFTBNAME	VARCHAR(18)	Name of the parent table of the referential constraint
REFTBCREATOR	CHAR(8)	Authorization ID of the owner of the parent table.
COLCOUNT	SMALLINT	Number of columns in the foreign key.
DELETERULE	CHAR(1)	Type of delete rule for the referential constraint. C CASCADE R RESTRICT N SET NULL
IBMREQD	CHAR(1)	Whether the row came from the basic machine-readable material (MRM) tape: N no Y yes
RELOBID1	SMALLINT	Internal identifier of the constraint with respect to the database that contains the parent table.
RELOBID2	SMALLINT	Internal identifier of the constraint with respect to the database that contains the dependent table.
TIMESTAMP	TIMESTAMP	The date and time the constraint was defined. If the constraint is between catalog tables, the value is 1985-04-01-00.00.00.000000.

SYSIBM.SYSRESAUTH Table

The SYSIBM.SYSRESAUTH table records the privileges held by users over buffer pools, storage groups, and table spaces.

Column Name	Data Type	Description
GRANTOR	CHAR(8)	Authorization ID of the user who granted the privilege.
GRANTEE	CHAR(8)	Authorization ID of the user who holds the privilege, or name of an application plan that uses the privilege.
QUALIFIER	CHAR(8)	This column contains blanks if the row describes a privilege over a buffer pool or storage group. It contains the qualifier of the table space name (the database name), if the row describes a privilege over a table space.
NAME	CHAR(8)	Name of the storage group, table space, or buffer pool.
GRANTEETYPE	CHAR(1)	Meaning: blank GRANTEE is an authorization ID P GRANTEE is an application plan
AUTHHOWGOT	CHAR(1)	Authorization level of the user from whom the privileges were received. **Note:** This is the authorization level used by the authorization subcomponent. It is not necessarily the highest authorization level of the grantor. blank not applicable C DBCTL D DBADM M DBMAINT S SYSADM
OBTYPE	CHAR(1)	Object type: B buffer pool S storage group R table space
TIMESTAMP	CHAR(12)	Time at which the privilege was granted (internal timestamp format).
DATEGRANTED	CHAR(6)	Date the privilege was granted; in the form *yymmdd*.
TIMEGRANTED	CHAR(8)	Time the privilege was granted; in the form *hhmmssth*.
USEAUTH	CHAR(1)	Whether the privilege is held with the GRANT option: G the privilege is held with the GRANT option Y the privilege is held without the GRANT option
IBMREQD	CHAR(1)	Whether the row came from the basic machine-readable material (MRM) tape: N no Y yes

SYSIBM.SYSSTMT Table

The SYSIBM.SYSSTMT table contains one or more rows for each SQL statement of each DBRM.

Column Name	Data Type	Description
NAME	CHAR(8)	Name of the DBRM.
PLNAME	CHAR(8)	Name of the application plan.
PLCREATOR	CHAR(8)	Authorization ID of the owner of the application plan.
SEQNO	SMALLINT	The sequence number of this row; the first portion of the SQL text is stored on row one and successive rows have increasing values for SEQNO.
STMTNO	SMALLINT	Statement number of the SQL statement in the source program.
SECTNO	SMALLINT	The section number of the section within the DBRM identified in the NAME column.
IBMREQD	CHAR(1)	Whether the row came from the basic machine-readable material (MRM) tape: N no Y yes
TEXT	VARCHAR(254)	The text or portion of the text of the SQL statement.

SYSIBM.SYSSTOGROUP Table

The SYSIBM.SYSSTOGROUP table contains one row for each storage group.

Column Name	Data Type	Description
NAME	CHAR(8)	Name of the storage group.
CREATOR	CHAR(8)	Authorization ID of the owner of the storage group.
VCATNAME	CHAR(8)	Name of the ICF catalog.
VPASSWORD	CHAR(8)	Password for the ICF catalog.
SPACE	INTEGER	Number of kilobytes of DASD storage allocated to the storage group as determined by the last execution of the STOSPACE utility.
SPCDATE	CHAR(5)	Date when the SPACE column was last updated, in the form *yyddd*.
IBMREQD	CHAR(1)	Whether the row came from the basic machine-readable material (MRM) tape: N no Y yes
CREATEDBY	CHAR(8)	Primary authorization ID of the user who created the storage group.

SYSIBM.SYSSYNONYMS Table

The SYSIBM.SYSSYNONYMS table contains one row for each synonym of a table or view.

Column Name	Data Type	Description
NAME	VARCHAR(18)	Synonym for the table or view.
CREATOR	CHAR(8)	Authorization ID of the owner of the synonym.
TBNAME	VARCHAR(18)	Name of the table or view.
TBCREATOR	CHAR(8)	Authorization ID of the owner of the table or view.
IBMREQD	CHAR(1)	Whether the row came from the basic machine-readable material (MRM) tape: N no Y yes
CREATEDBY	CHAR(8)	Primary authorization ID of the user who created the synonym.

SYSIBM.SYSTABAUTH Table

The SYSIBM.SYSTABAUTH table records the privileges held by users on tables and views.

Column Name	Data Type	Description
GRANTOR	CHAR(8)	Authorization ID of the user who granted the privileges. Could also be PUBLIC, or PUBLIC followed by an asterisk.[10]
GRANTEE	CHAR(8)	Authorization ID of the user who holds the privileges or the name of a plan that uses the privileges. PUBLIC for a grant to PUBLIC. PUBLIC followed by an asterisk for a grant to PUBLIC AT ALL LOCATIONS.
GRANTEETYPE	CHAR(1)	Meaning: blank GRANTEE is an authorization ID P GRANTEE is an application plan
DBNAME	CHAR(8)	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.
SCREATOR	CHAR(8)	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 referenced in the CREATE VIEW statement. Otherwise, SCREATOR is the same as TCREATOR.
STNAME	VARCHAR(18)	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 referenced in the CREATE VIEW statement. Otherwise, STNAME is the same as TTNAME.
TCREATOR	CHAR(8)	Authorization ID of the owner of the table or view.
TTNAME	VARCHAR(18)	Name of the table or view.
AUTHHOWGOT	CHAR(1)	Authorization level of the user from whom the privileges were received. **Note:** This is the authorization level used by the authorization subcomponent. It is not necessarily the highest authorization level of the grantor. blank not applicable S SYSADM D DBADM C DBCTL M DBMAINT
TIMESTAMP	CHAR(12)	Time at which the privileges were granted (internal timestamp format).
DATEGRANTED	CHAR(6)	Date the privileges were granted, in the form *yymmdd*.
TIMEGRANTED	CHAR(8)	Time the privileges were granted, in the form *hhmmssth*.
UPDATECOLS	CHAR(1)	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 which list the columns on which update privileges have been granted.
ALTERAUTH	CHAR(1)	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
DELETEAUTH	CHAR(1)	Whether the GRANTEE can delete rows from the table or view: blank not applicable, or privilege not held G privilege is held with the GRANT option Y privilege is held without the GRANT option

[10] PUBLIC followed by an asterisk (PUBLIC*) denotes PUBLIC AT ALL LOCATIONS. For conditions under which GRANTOR can be PUBLIC or PUBLIC* see Section 5 (Volume 2) of *Administration Guide*.

Column Name	Data Type	Description
INDEXAUTH	CHAR(1)	Whether the GRANTEE can create indexes on the table: blank not applicable, or privilege not held G privilege is held with the GRANT option Y privilege is held without the GRANT option
INSERTAUTH	CHAR(1)	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
SELECTAUTH	CHAR(1)	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
UPDATEAUTH	CHAR(1)	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
IBMREQD	CHAR(1)	Whether the row came from the basic machine-readable material (MRM) tape: N no Y yes
GRANTEELOCATION	CHAR(16)	Not used.

SYSIBM.SYSTABLEPART Table

The SYSIBM.SYSTABLEPART table contains one row for each unpartitioned table space and one row for each partition of a partitioned table space.

Column Name	Data Type	Description
PARTITION	SMALLINT	Partition number; 0 if table space is not partitioned.
TSNAME	CHAR(8)	Name of the table space.
DBNAME	CHAR(8)	Name of the database containing the table space.
IXNAME	VARCHAR(18)	Name of the partitioned index. This column is blank if the table space is not partitioned.
IXCREATOR	CHAR(8)	Authorization ID of the owner of the index. This column is blank if the table space is not partitioned.
PQTY	INTEGER	Primary space allocation in units of 4K-byte storage blocks. The value of this column is 0 if a storage group is not used.
SQTY	SMALLINT	Secondary space allocation in units of 4K-byte blocks. The value of this column is 0 if a storage group is not used.
STORTYPE	CHAR(1)	Type of storage allocation: E explicit (storage group not used) I implicit (storage group used)
STORNAME	CHAR(8)	Name of storage group used for space allocation. Blank if storage group not used.
VCATNAME	CHAR(8)	Name of ICF catalog used for space allocation.
CARD	INTEGER	Number of rows in the table space or partition. -1 if statistics not gathered.
FARINDREF	INTEGER	Number of rows that have been relocated far from their original page. -1 if statistics not gathered.
NEARINDREF	INTEGER	Number of rows that have been relocated near their original page. -1 if statistics not gathered.
PERCACTIVE	SMALLINT	Percentage of space occupied by rows of data from active tables. -1 if statistics not gathered.
PERCDROP	SMALLINT	Percentage of space occupied by rows of dropped tables. -1 if statistics have not been gathered. 0 for segmented table spaces.
IBMREQD	CHAR(1)	Whether the row came from the basic machine-readable material (MRM) tape: N no Y yes
LIMITKEY	VARCHAR(512)	The limit key of the partition in an external format. 0 if the table space is not partitioned.
FREEPAGE	SMALLINT	The number of pages loaded before a page is left as free space.
PCTFREE	SMALLINT	The percentage of each page left as free space.
CHECKFLAG	CHAR(1)	C The table space partition is in CHECK PENDING mode and there are rows that may violate referential constraints. blank Blank if the table space is not a partition or does not contain rows that may violate referential constraints.
CHECKRID	CHAR(4)	CHECKRID is blank if the table or partition is not in a check pending state (the CHECKFLAG is blank) or if the table space is not partitioned. Otherwise, it is the RID of the first row of the table space partition that may violate referential constraints or the value is X'00000000' indicating that any row may violate referential constraints.

SYSIBM.SYSTABLES Table

The SYSIBM.SYSTABLES table contains one row for each table, view, or alias.

Column Name	Data Type	Description
NAME	VARCHAR(18)	Name of the table, view, or alias.
CREATOR	CHAR(8)	Authorization ID of the owner of the table, view, or alias.
TYPE	CHAR(1)	Type of object: A alias T table V view
DBNAME	CHAR(8)	For a table, or a view of tables, the name of the database that contains the table space named in TSNAME. For an alias, or a view of a view, the value is DSNDB06.
TSNAME	CHAR(8)	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 view of a view, the value is SYSVIEWS. For an alias, it is SYSDBAUT.
DBID	SMALLINT	Internal identifier of the database; 0 if the row describes a view or alias.
OBID	SMALLINT	Internal identifier of the table; 0 if the row describes a view or alias.
COLCOUNT	SMALLINT	Number of columns in the table or view. 0 if the row describes an alias.
EDPROC	CHAR(8)	Name of the edit procedure; blank if the row describes a view or alias or a table without an edit procedure.
VALPROC	CHAR(8)	Name of the validation procedure; blank if the row describes a view or alias or a table without a validation procedure.
CLUSTERTYPE	CHAR(1)	Not used; all values are blank.
CLUSTERRID	INTEGER	Not used; all values are 0.
CARD	INTEGER	Total number of rows in the table. -1 if statistics not gathered or the row describes a view or alias. This is an updateable column.
NPAGES	INTEGER	Total number of pages on which rows of the table appear. -1 if statistics not gathered or the row describes a view or alias. This is an updateable column.
PCTPAGES	SMALLINT	Percentage of total pages of the table space that contain rows of the table. If the table space is segmented, the percentage of total pages in the set of segments assigned to the table. -1 if statistics not gathered or the row describes a view or alias. This is an updateable column.
IBMREQD	CHAR(1)	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
REMARKS	VARCHAR(254)	A character string provided by the user with the COMMENT statement.
PARENTS	SMALLINT	The number of relationships in which the table is a dependent. 0 if the row describes a view or alias.
CHILDREN	SMALLINT	The number of relationships in which the table is a parent. 0 if the row describes a view or alias.
KEYCOLUMNS	SMALLINT	The number of columns in the table's primary key. 0 if the row describes a view or alias.

Column Name	Data Type	Description
RECLENGTH	SMALLINT	The maximum length of any record in the table. Length is 8+N+L, where: • 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 25. The value is 0 if the row describes a view or alias. For maximum row and record sizes, see "Notes" on page 151.
STATUS	CHAR(1)	I The table's definition is incomplete because it lacks a primary index. X The table has a primary index. blank Table has no primary key, or is a catalog table, or the row describes a view or alias.
KEYOBID	SMALLINT	Internal DB2 identifier of the index that enforces uniqueness of the table's primary key; 0 if not applicable.
LABEL	VARCHAR(30)	The label as given by a LABEL ON statement; otherwise an empty string.
CHECKFLAG	CHAR(1)	C The table space containing the table is in CHECK PENDING mode and there are rows in the table that may violate referential constraints. blank The table contains no rows that violate referential constraints, or the row describes a view or alias.
CHECKRID	CHAR(4)	Blank if the table is not in a check pending state (the CHECKFLAG is blank), if the table space is partitioned, or if the row describes a view or alias. Otherwise, it is the RID of the first row of the table that may violate referential constraints or the value is X'00000000' indicating that any row may violate referential constraints.
AUDITING	CHAR(1)	Value of the audit option: A AUDIT ALL C AUDIT CHANGE blank AUDIT NONE
CREATEDBY	CHAR(8)	Primary authorization ID of the user who created the table, view, or alias.
LOCATION	CHAR(16)	Blank for a table or view, and for an alias defined on a local object . Location name of the object for an alias defined on a remote object.
TBCREATOR	CHAR(8)	For an alias, the authorization ID of the owner of the referenced table or view; blank otherwise.
TBNAME	VARCHAR(18)	For an alias, the name for the referenced table or view; blank otherwise.

SYSIBM.SYSTABLESPACE Table

The SYSIBM.SYSTABLESPACE table contains one row for each table space.

Column Name	Data Type	Description
NAME	CHAR(8)	Name of the table space.
CREATOR	CHAR(8)	Authorization ID of the owner of the table space.
DBNAME	CHAR(8)	Name of the database containing the table space.
DBID	SMALLINT	Internal identifier of the database which contains the table space.
OBID	SMALLINT	Internal identifier of the table space file descriptor.
PSID	SMALLINT	Internal identifier of the table space page set descriptor.
BPOOL	CHAR(8)	Name of the buffer pool used for the table space.
PARTITION	SMALLINT	Number of partitions of the table space; 0 if the table space is not partitioned.
LOCKRULE	CHAR(1)	Lock size of the table space: A any P page S table space T table
PGSIZE	SMALLINT	Size of pages in the table space in kilobytes.
ERASERULE	CHAR(1)	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
STATUS	CHAR(1)	Availability status of the table space: A available C definition is incomplete because no partitioned index has been created P table space is in CHECK PENDING mode S table space is in CHECK PENDING mode with the scope less than the entire table space. T definition is incomplete because no table has been created
IMPLICIT	CHAR(1)	Whether the table space was created implicitly: Y yes N no
NTABLES	SMALLINT	Number of tables defined in the table space.
NACTIVE	INTEGER	Number of active pages in the table space. 0 if statistics are not gathered.
DSETPASS	CHAR(8)	The password for the data sets of the table space.
CLOSERULE	CHAR(1)	Whether the data sets are to be closed when the table space is not in use: Y yes N no
SPACE	INTEGER	Number of kilobytes of DASD storage allocated to the table space, as determined by the last execution of the STOSPACE utility. 0 if the table space is not related to a storage group. If the table space is partitioned, and different storage groups have been specified, this is the number of kilobytes of DASD storage allocated to the partition as determined by the last execution of the STOSPACE utility.
IBMREQD	CHAR(1)	Whether the row came from the basic machine-readable material (MRM) tape: N no Y yes C V2R1 dependency indicator; not from MRM tape
ROOTNAME	VARCHAR(18)	For catalog use only.
ROOTCREATOR	CHAR(8)	For catalog use only.
SEGSIZE	SMALLINT	The number of pages in each segment of a segmented table space. Zero if the table space is not segmented.
CREATEDBY	CHAR(8)	Primary authorization ID of the user who created the table space.

SYSIBM.SYSUSERAUTH Table

The SYSIBM.SYSUSERAUTH table records the system privileges held by users.

Column Name	Data Type	Description
GRANTOR	CHAR(8)	Authorization ID of the user who granted the privileges.
GRANTEE	CHAR(8)	Authorization ID of the user who holds the privileges, or the name of a plan that uses the privileges.
TIMESTAMP	CHAR(12)	Time at which the privileges were granted (internal timestamp format).
DATEGRANTED	CHAR(6)	Date the privileges were granted; in the form *yymmdd*.
TIMEGRANTED	CHAR(8)	Time the privileges were granted; in the form *hhmmssth*.
GRANTEETYPE	CHAR(1)	Meaning: blank GRANTEE is an authorization ID P GRANTEE is an application plan
AUTHHOWGOT	CHAR(1)	Authorization level of the user from whom the privileges were received. **Note:** This is the authorization level used by the authorization subcomponent. It is not necessarily the highest authorization level of the grantor. blank not applicable S SYSADM D DBADM C DBCTL M DBMAINT
ALTERBPAUTH	CHAR(1)	Not used.
BINDADDAUTH	CHAR(1)	Whether the GRANTEE can use the BIND command with the ADD option: blank privilege is not held G privilege is held with the GRANT option Y privilege is held without the GRANT option
BSDSAUTH	CHAR(1)	Whether the GRANTEE can issue the -RECOVER BSDS command: blank privilege is not held G privilege is held with the GRANT option Y privilege is held without the GRANT option
CREATEDBAAUTH	CHAR(1)	Whether the GRANTEE can create databases and automatically receive DBADM authority over the new databases: blank privilege is not held G privilege is held with the GRANT option Y privilege is held without the GRANT option
CREATEDBCAUTH	CHAR(1)	Whether the GRANTEE can create new databases and automatically receive DBCTRL authority over the new databases: blank privilege is not held G privilege is held with the GRANT option Y privilege is held without the GRANT option
CREATESGAUTH	CHAR(1)	Whether the GRANTEE can create new storage groups: blank privilege is not held G privilege is held with the GRANT option Y privilege is held without the GRANT option
DISPLAYAUTH	CHAR(1)	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
RECOVERAUTH	CHAR(1)	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
STOPALLAUTH	CHAR(1)	Whether the GRANTEE can use the DB2 -STOP command: 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
STOSPACEAUTH	CHAR(1)	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
SYSADMAUTH	CHAR(1)	Whether the GRANTEE has system administration authority: blank privilege is not held G privilege is held with the GRANT option Y privilege is held without the GRANT option
SYSOPRAUTH	CHAR(1)	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
TRACEAUTH	CHAR(1)	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
IBMREQD	CHAR(1)	Whether the row came from the basic machine-readable material (MRM) tape: N no Y yes
MON1AUTH	CHAR(1)	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
MON2AUTH	CHAR(1)	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
CREATEALIASAUTH	CHAR(8)	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

SYSIBM.SYSVIEWDEP Table

The SYSIBM.SYSVIEWDEP table records the dependencies of views on tables and other views.

Column Name	Data Type	Description
BNAME	VARCHAR(18)	Name of a table or view on which the view is dependent.
BCREATOR	CHAR(8)	Authorization ID of the owner of BNAME.
BTYPE	CHAR(1)	Type of object BNAME: T table V view
DNAME	VARCHAR(18)	Name of the view.
DCREATOR	CHAR(8)	Authorization ID of the owner of the view.
IBMREQD	CHAR(1)	Whether the row came from the basic machine-readable material (MRM) tape: N no Y yes

SYSIBM.SYSVIEWS Table

The SYSIBM.SYSVIEWS table contains one or more rows for each view.

Column Name	Data Type	Description
NAME	VARCHAR(18)	Name of the view.
CREATOR	CHAR(8)	Authorization ID of the owner of the view.
SEQNO	SMALLINT	Sequence number of this row; the first portion of the view is on row one and successive rows have increasing values of SEQNO.
CHECK	CHAR(1)	Whether the CHECK option was specified in the CREATE VIEW statement: N no Y yes **Note:** The value will be N if the view has no WHERE clause.
IBMREQD	CHAR(1)	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.
TEXT	VARCHAR(254)	The text or portion of the text of the CREATE VIEW statement.

SYSIBM.SYSVLTREE Table

The SYSIBM.SYSVLTREE table contains a row for each view whose parse tree could not be totally contained in the table SYSIBM.SYSVTREE. The row contains the remainder of the parse tree.

Column Name	Data Type	Description
IBMREQD	CHAR(1)	Whether the row came from the basic machine-readable material (MRM) tape: N no Y yes
VTREE	VARCHAR(4000)	The remaining part of the parse tree of a view.

SYSIBM.SYSVOLUMES Table

The SYSIBM.SYSVOLUMES table contains one row for each volume of each storage group.

Column Name	Data Type	Description
SGNAME	CHAR(8)	The name of the storage group.
SGCREATOR	CHAR(8)	Authorization ID of the owner of the storage group.
VOLID	CHAR(6)	The serial number of the volume.
IBMREQD	CHAR(1)	Whether the row came from the basic machine-readable material (MRM) tape: N no Y yes

SYSIBM.SYSVTREE Table

The SYSIBM.SYSVTREE table contains a row for each view. Each row contains the parse tree of the view. If the parse tree is longer than 4000 bytes, the rest of the parse tree is saved in the SYSIBM.SYSVLTREE table.

Column Name	Data Type	Description
NAME	VARCHAR(18)	Name of the view.
CREATOR	CHAR(8)	Authorization ID of the owner of the view.
TOTLEN	INTEGER	Total length of the parse tree.
IBMREQD	CHAR(1)	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.
VTREE	VARCHAR(4000)	Parse tree or portion of the parse tree of the view.

Appendix D. The Communications Database

The Communications Data Base (CDB) contains several tables that hold information about your connections with remote DB2 subsystems. DB2 administration is responsible for the CDB. This appendix describes the columns of five different tables.

SYSIBM.SYSLOCATIONS Table

Every DB2 subsystem has a location name and an associated VTAM LUNAME. The SYSIBM.LOCATIONS table shows this association and contains a row for every DB2 subsystem.

Column Name	Data Type	Description
LOCATION	CHAR(16)	Unique network location name for DB2 subsystem.
LOCTYPE	CHAR(1)	Reserved and must be blank.
LINKNAME	CHAR(8)	LUNAME for the same DB2 subsystem.
LINKATTR	VARCHAR(64)	Reserved and must be a string of length zero.

SYSIBM.SYSLUMODES Table

Each row of this table provides VTAM with conversation limits for a *specific* combination of LUNAME and MODENAME. This table is accessed only at DDF startup for negotiation of session limits with a remote DB2 for a specific node. This negotiation is called *Change-Number-of-Sessions* (CNOS) processing.

Column Name	Data Type	Description
LUNAME	CHAR(8)	LUNAME of the remote DB2 subsystem. All LUNAMES defined in this table must also be defined in SYSLUNAMES.
MODENAME	CHAR(8)	Name of a logon mode description in the VTAM logon mode table.
CONVLIMIT	SMALLINT	Maximum number of active conversations between the local and remote DB2 subsystems for this mode. Used to override the number in the DSESLIM parameter of the VTAM APPL definition statement for this mode.
AUTO	CHAR(1)	When CNOS processing and preallocation of sessions will be initiated: blank Default: deferred until the first needed reference to the LUNAME via this MODENAME. N Deferred until the first needed reference to the LUNAME via this MODENAME. Y Initiated at DDF startup.

SYSIBM.SYSLUNAMES Table

The SYSIBM.SYSLUNAMES table lists the characteristics associated with an LUNAME that denotes a remote subsystem that you can send an SQL query to. Each row in this table represents a separate subsystem.

Column Name	Data Type	Description
LUNAME	CHAR(8)	LUNAME of the remote subsystem. LUNAMES of all remote subsystems *must* appear in this table if they are to be allowed to communicate with the local subsystem.
SYSMODENAME	CHAR(8)	Mode used to establish intersystem conversations.
USERSECURITY	CHAR(1)	Security acceptance option for attach requests: C Conversation: all conversations received must contain an AUTHID and password. A Already verified: a conversation received may contain either an AUTHID and password, or an AUTHID with no password. If a password is supplied, it will be passed to RACF for password validation.
ENCRYPTPSWDS	CHAR(1)	Whether passwords are encrypted: Y Yes: For outbound requests, the encrypted password is extracted from RACF and sent to the remote subsystem. For inbound requests, received passwords are treated as encrypted. N No.
MODESELECT	CHAR(1)	Whether to use the SYSMODESELECT table: blank Uses default mode IBMDB2LM. N Uses default mode IBMDB2LM. Y Searches SYSMODESELECT for appropriate mode name.
USERNAMES	CHAR(1)	Level of "come from" checking and ID translation required: blank No translation occurs. O Outbound request: subject to ID translation. I Inbound request: subject to ID translation and "come from" checking. B Both requirements, O and I, must be met.

SYSIBM.SYSMODESELECT Table

The SYSIBM.SYSMODESELECT table associates authorization IDs and/or application plans with modes.

Column Name	Data Type	Description
AUTHID	CHAR(8)	Authorization ID of the request for data from another subsystem. Blank, the default, indicates that the specified MODENAME is to apply to all authorization IDs.
PLANNAME	CHAR(8)	Plan name associated with the request for data from another subsystem. Blank, the default, indicates that the specified MODENAME is to apply to all plan names.
LUNAME	CHAR(8)	LUNAME to which the specific MODENAME applies.
MODENAME	CHAR(8)	Name of the logon mode in the VTAM logon mode table. Used when creating a conversation supporting the request for data from another subsystem. If blank, default mode (IBMDB2LM) will be used.

SYSIBM.SYSUSERNAMES Table

The SYSIBM.SYSUSERNAMES table is used to carry out an ID translation and/or a "come from" check.

Column Name	Data Type	Description
TYPE	CHAR(1)	How the row is to be used: O Outbound. I Inbound and "come from" checking.
AUTHID	CHAR(8)	Authorization ID to be translated. Applies to any authorization ID if blank.
LUNAME	CHAR(8)	LUNAME of a remote subsystem. Identifies the sending and receiving subsystems. If blank, the new AUTHID applies for all otherwise undefined subsystems.
NEWAUTHID	CHAR(8)	Translated value. A blank specifies no translation.
PASSWORD	CHAR(8)	Password to accompany an outbound request, if passwords are not encrypted. If passwords are encrypted, the column is not used.

Appendix E. SQL Reserved Words

The following words are reserved words in SQL. They may not be used as ordinary identifiers in forming names. They may be used as delimited identifiers by enclosing them between double quotation marks.

ADD
ALL
ALTER
AND
ANY
AS
AUDIT

BETWEEN
BUFFERPOOL
BY

CLUSTER
COLUMN
COUNT
CURRENT
CURSOR

DATABASE
DELETE
DESCRIPTOR
DISTINCT
DROP

EDITPROC
END-EXEC[11]
ERASE
EXECUTE
EXISTS

FIELDPROC
FOR
FROM

GO
GOTO
GRANT
GROUP

HAVING

IMMEDIATE
IN
INDEX
INSERT
INTO
IS

KEY

LIKE
LOCKSIZE

NOT
NULL
NUMPARTS

OF
ON
OR
ORDER

PART
PLAN
PRIQTY
PRIVILEGES

SECQTY
SELECT
SET
SOME
STOGROUP
SYNONYM

TABLE
TABLESPACE
TO

UNION
UPDATE
USER
USING

VALIDPROC
VALUES
VCAT
VIEW
VOLUMES

WHERE
WITH

The Systems Application Architecture definition of SQL has additional reserved words. These additional reserved words are not enforced by DB2, but we suggest that you do not use them as ordinary identifiers in names that will have a continuing use. See *SAA CPI Database Reference*, SC26-4348, for a list of these words.

[11] COBOL only

Glossary

access path. The path used to get to data specified in SQL statements. An access path can involve either an index, a sequential search or a combination of both.

alias. (1)A locally defined name that can be used in SQL statements to refer to a table or view in the same or a remote DB2 subsystem. (2)An alternate name for a member of a partitioned data set.

application. A program or set of programs that performs a task; for example, a payroll application.

application plan. The control structure produced during the bind process and used by the database manager to process SQL statements encountered during application execution.

application-embedded SQL. SQL statements coded within an application program.

attachment facility. An interface between DB2 and TSO, IMS/VS, CICS, or batch address spaces. An attachment facility allows application programs to access DB2.

authorization ID. A string that designates a set of privileges. It may represent an individual, an organizational group, or a function, but DB2 does not determine this representation.

autocommit. A SPUFI option that commits the effects of SQL statements automatically if they are successfully executed.

bind. The process by which the output from the precompiler is converted to a usable control structure called an application plan. This process is the one during which access paths to the data are selected and some authorization checking is performed.

> **automatic bind.** Binding done automatically (without a user issuing a BIND command) when an application program is being run and the bound application plan has been invalidated.
>
> **dynamic bind.** Binding done dynamically (as the SQL statements are entered) when SQL statements are entered through dynamic SQL.
>
> **incremental bind.** Binding of an SQL statement is done during the execution of an application process because the statement could not be bound during the bind process and VALIDATE(RUN) was specified.
>
> **static bind.** The process by which the output from the precompiler is converted to a usable control structure called an application plan. This process is the one during which access paths to the data are selected and some authorization checking is performed.

buffer pool. Main storage reserved to satisfy the buffering requirements for one or more tables or indexes.

catalog. A collection of tables that contain descriptions of objects such as tables, views, and indexes.

CDB. communications database.

character string. A sequence of bytes representing bit data, single-byte characters, or a mixture of single and double-byte characters.

checkpoint. A point at which DB2 records internal status information on the DB2 log. This log is used in the recovery process should DB2 abnormally terminate.

clause. In SQL, a distinct part of a statement, such as a SELECT clause or a WHERE clause.

clustering index. An index that determines how rows are physically ordered in a table space.

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 in a table.

commit. The process that allows data, changed by one application or user, to be referenced by other applications or users. When a COMMIT occurs, locks are freed so that other applications can reference the just-committed data. When data has been committed, a new commit point is established.

communications database (CDB). A new system database that contains tables (SYSLOCATIONS, SYSLUMODES, SYSLUNAMES, SYSMODESELECT, SYSUSERNAMES) used to establish conversations with remote DB2 subsystems.

comparison operator. A symbol (such as =,>,<) used to specify a relationship between two values.

concurrency. The shared use of resources by multiple interactive users or application processes at the same time.

correlated subquery. A subquery (part of a WHERE or HAVING clause) applied to a row or group of rows of

the table or view named in the outer sub-SELECT statement.

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.

cursor. A named control structure used by an application program to point to a row of interest within some ordered set of rows. The cursor is used to retrieve rows from the set, possibly making updates or deletions.

cursor stability. The isolation level that provides maximum concurrency. With cursor stability, a unit of work holds locks only on its uncommitted changes and the current row of each of its cursors.

database. A collection of table spaces and index spaces.

database administrator (DBA). An individual responsible for the design, development, operation, safeguarding, maintenance, and use of a database.

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.

data type. An attribute of columns, literals, and host variables.

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.

date/time value. A value of the data type DATE, TIME, or TIMESTAMP.

DBA. database-administrator

DBCS. double-byte character set

DBID. database identifier

DBRM. database request module

DB2 catalog. DB2-maintained tables that contain descriptions of DB2 objects such as tables, views, and indexes.

DB2 Interactive (DB2I). The DB2 facility that provides for the execution of SQL statements, DB2 (operator) commands, programmer commands, and utility invocation.

DB2I. DATABASE 2 Interactive.

DCLGEN. declarations generator.

DDF. distributed data facility.

declarations generator (DCLGEN). A subcomponent of DB2 that generates SQL table declarations and COBOL, C, or PL/I data structure declarations that conform to the table. The declarations are generated from DB2 system catalog information. DCLGEN is also a DSN subcommand.

default value. A predetermined value, attribute, or option that is assumed when no other is explicitly specified.

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.*.

distributed data facility (DDF). A facility of DB2, new in Version 2 Release 2, that can be started and stopped by command in order to provide connections to remote subsystems.

double-byte character set (DBCS). A set of characters in which each character is represented by two bytes. Languages such as Japanese, Chinese, and Korean, which contain more symbols than can be represented by 256 code points, require double- byte character sets. Since each character requires two bytes, entering, displaying, and printing DBCS characters requires hardware and supporting software which are DBCS capable.

duration. A number that represents an interval of time. See *labeled duration, date duration,* and *time duration.*

dynamic SQL. SQL statements that are prepared and executed within a 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 might change several times during the program's execution.

EBCDIC. extended binary coded decimal interchange code.

embedded SQL. SQL statements that are embedded within an application program and are prepared during the program preparation process before the program is executed. After it is prepared, the statement itself does not change (although values of host variables specified within the statement might change).

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) may be assigned by the user.

EUR. IBM European Standards

expression. An operand or a collection of operators and operands that yields a single value.

field procedure. A user-written routine designed to receive a single value and transform (encode or decode) it in any way the user may specify.

fixed-length string. A character or graphic string whose length is specified and cannot be changed.

function. A scalar function or column function. Same as *built-in function*.

graphic string. A sequence of DBCS characters.

host language. Any programming language in which you can embed SQL statements.

host program. A 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, a variable referenced by embedded SQL statements.

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/VS. Information Management System/Virtual Storage.

index. A set of pointers that are logically ordered by the values of a key. Indexes provide quick 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 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.

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* and *repeatable read*.

JIS. Japanese Industrial Standard.

join. A relational operation that allows retrieval of data from two or more tables based on matching column values.

K. Kilobyte (1024 bytes).

labeled duration. A number that represents a duration of years, months, days, hours, minutes, seconds, or microseconds.

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.

locking. The process which ensures integrity of data. Locking prevents concurrent users from accessing inconsistent data.

long string. A string whose actual length, or a varying-length string whose maximum length, is greater than 254 bytes or 127 double-byte characters.

mixed data string. A character string that may contain both single-byte and double- byte characters.

null. A special value that indicates the absence of information.

OBID. data object identifier.

object. Anything that can be created or manipulated with SQL — that is, databases, table spaces, tables, views, or indexes.

page. A unit of storage within a table space (4K or 32K) or index space (4K). In a table space, a page contains one or more rows of a table.

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.

plan. See *application plan*.

plan name. The name of an application plan.

precompilation. A processing of application programs containing SQL statements that takes place before compilation. SQL statements are replaced with statements that will be 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.

prepared SQL statement. A named object that is the executable form of an SQL statement that has been processed by the PREPARE statement.

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 primary key.

privilege. A capability given to a user by the execution of a GRANT statement.

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.

RACF. OS/VS2 MVS Resource Access Control Facility

RBA. relative byte address.

recovery. The process of rebuilding databases after a system failure.

referential integrity. The enforcement of referential constraints on LOAD, INSERT, UPDATE, and DELETE operations.

relative byte address (RBA). The offset of a data record or control interval from the beginning of the storage space allocated to the data set or file to which it belongs.

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.

repeatable read. The isolation level that provides maximum protection from other executing application programs. When a program executes with repeatable read protection, rows referenced by the program cannot be changed by other programs until the program reaches a commit point.

resource limit facility (RLF). A portion of DB2 code that prevents dynamic SQL queries from exceeding specified time limits.

result table. The set of rows selected for use by an application program. The application uses a cursor to retrieve the rows one by one into a host structure or a set of host variables.

rollback. Initiates the abort process.

row. The horizontal component of a table. A row consists of a sequence of values, one for each column of the table.

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.

search condition. A criterion for selecting rows from a table. A search condition consists of one or more predicates.

short string. A string whose actual length, or a varying-length string whose maximum length, is 254 bytes (127 double-byte characters) or less.

SMS. Storage Management Subsystem

SMF. system management facility.

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.

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. A language that can be used within host programming languages or interactively to access data and to control access to resources.

SQL authorization ID. The authorization ID that is used as the implicit qualifier of table, view, and index names in dynamic SQL statements. The SQL authorization ID, along with the other authorization IDs of a process, is used for authorization checking of dynamic SQL statements. It also serves a special role for CREATE, GRANT, and REVOKE statements.

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 either input or output variables used in the execution of manipulative SQL statements.

SQL escape character. See *escape character*.

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.

SQL/DS. SQL/Data System.

SQLCA. SQL communication area.

SQLDA. SQL descriptor area.

SQL ID. Short for SQL authorization ID.

SSI. MVS subsystem interface.

static SQL. See *embedded SQL.*

storage group. A named set of DASD volumes on which DB2 data can be stored.

string. A character or graphic string.

subquery. A SELECT statement within the WHERE or HAVING clause of another SQL statement. A nested SQL statement.

subselect. That form of the SELECT statement that does not include ORDER BY or UNION operators.

synonym. A user's alternative name for a table or view.

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.

table. A named data object consisting of a specific number of columns and some number of unordered rows.

table space. A page set used to store the records of one or more tables.

thread. The DB2 structure that describes an application's connection, traces its progress, provides resource function processing capability, and delimits its accessibility to DB2 resources and services. Most DB2 functions execute under a thread structure.

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.

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 tool that allows the user to monitor and collect DB2 monitoring, auditing, performance, accounting, statistics, and serviceability (global) data.

TSO. Time Sharing Option. A subsystem of MVS.

UNION. An SQL operation that combines the results of two subselects. 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.

unit of recovery (UR). A recoverable sequence of operations within a process. At any time, a process is a single unit of recovery, but the life of a process may involve many units of recovery as a result of commit or rollback operations.

UT. utility-only access.

value. Smallest unit of data manipulated in SQL.

varying-length string. A character or graphic string whose length is not fixed, but variable within set limits.

view. An alternative representation of data from one or more tables. A view can include all or some of the columns contained in the table or tables on which it is defined.

VSAM. Virtual Storage Access Method.

Bibliography

IBM DATABASE 2 Version 2 Release 2 Library

- *General Information*, GC26-4373-1
- *Administration Guide*, SC26-4374-1
  - Planning and Installing DB2
  - Communicating with Other Subsystems
  - Designing a Database
  - Security and Auditing
  - Operation and Recovery
  - Performance Monitoring and Tuning
- *Application Programming and SQL Guide*, SC26-4377-1
  - Using Interactive SQL
  - Coding SQL in your Application Program
  - Executing and Testing Your Application
  - Programming for Special Purpose Interfaces
- *SQL Reference*, SC26-4380-1
  - Concepts
  - Language Elements
  - Functions
  - Queries
  - Statements
- *Command and Utility Reference*, SC26-4378-1
  - Commands
  - Utilities
- *Reference Summary*, SX26-3771-1
  - SQL Reference Summary
  - Command and Utility Reference Summary
- *Messages and Codes*, SC26-4379-1
  - SQL Return Codes
  - DB2 Messages
  - DB2 Codes
  - IRLM Messages and Codes
- *Diagnosis Guide and Reference*, LY27-9536-1
  - Functional Descriptions
  - Keyword Descriptions
  - Diagnostic Aids and Techniques
  - Data Management
  - Physical Formats and Diagrams
  - Data Areas
  - Trace Messages and Codes
- *Licensed Program Specifications*, GC26-4375-1
- *Program Directory*

Other References

- *APL2 Programming: Using Structured Query Language (SQL), SH20-9217*
- *AS/400 SQL Reference, SC21-9608*
- *CICS/MVS Application Programming Primer, SC33-0139*
- *CICS/MVS Facilities and Planning Guide, SC33-0504*
- *CICS/MVS Installation Guide, SC33-0506*
- *CICS/OS/VS Facilities and Planning Guide, SC33-0202*
- *CICS/OS/VS: Installation and Operations Guide, SC33-0071*
- *CICS/VS Application Programmer's Reference Manual (Command Level), SC33-0241*
- *Distributed Data Library: Concepts of Distributed Data, SC26-4417*
- *IBM BASIC Programming Guide, SC26-4027*
- *IBM OS/2 EE Database Manager SQL Reference, T90X-7945*
- *IMS/VS Version 2 Application Programming, SC26-4178*
- *IMS/VS Version 2 Installation Guide, SC26-4172*
- *IMS/VS Version 2 System Definition Reference Manual, SC26-4216*
- *IMS/VS Version 2 Utilities Reference, SC26-4173*
- *Introduction to Distributed Relational Data, GG24-3200*
- *ISPF Version 2 for MVS Dialog Management Services, SC34-4021*
- *ISPF/PDF Version 2 for MVS Reference, SC34-4024*
- *ISPF and ISPF/PDF Version 2 for MVS General Information, GC34-4041*
- *MVS/ESA System Programming Library: Application Development Guide, GC28-1852*
- *MVS/ESA System Programming Library: Application Development -- 31-Bit Addressing, GC28-1820*
- *MVS/ESA System Programming Library: Application Development Macro Reference, GC28-1857*
- *MVS/XA System Programming Library: 31-Bit Addressing, GC28-1158*
- *MVS/XA System Programming Library: Supervisor Services and Macro Instructions, GC28-1154*
- *OS/VS Message Library: VS2 TSO Terminal Messages, GD23-0264*
- *OS/VS Message Library: VS2 TSO Terminal Messages, GC38-1046*
- *OS/VS2 TSO Command Language Reference, GC28-0646*
- *OS/VS2 TSO Terminal User's Guide, GC28-0645*
- *Query Management Facility: Advanced User's Guide, SC26-4343*
- *Query Management Facility: Learner's Guide, SC26-4231*
- *Systems Application Architecture: An Overview*, GC26-4341
- *SAA Writing Applications: A Design Guide*, SC26-4362
- *SAA Common Programming Interface Database Reference*, SC26-4348
- *SQL/Data System SQL Reference, TH09-8067*
- *TSO Extensions CLISTs: Implementation and Reference, SC28-1304*
- *TSO Extensions Terminal Messages, GC28-1310*

Index

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

Y

Special Characters

IBM DATABASE 2 Version 2
SQL Reference

SC26-4380-1

Reader's Comment Form

This manual is part of a library that serves as a reference source for system analysts, programmers, and operators of IBM systems. You may use this form to communicate your comments about this publication, its organization, or subject matter, with the understanding that IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you. Your comments will be sent to the author's department for whatever review and action, if any, are deemed appropriate.

Note: Do not use this form to request IBM publications. If you do, your order will be delayed because publications are not stocked at the address printed on the reverse side. Instead, you should direct any requests for copies of publications, or for assistance in using your IBM system, to your IBM representative or to the IBM branch office serving your locality.

If you have applied any technical newsletters (TNLs) to this book, please list them here: _____

Comments (please include specific chapter and page references) :

Note: Staples can cause problems with automatic mail-sorting equipment.
Please use pressure-sensitive or other gummed tape to seal this form.

If you want a reply, please complete the following information:

Name _____ Date _____

Company _____ Phone No. (_____) _____

Address _____

_____

Thank you for your cooperation. No postage is necessary if mailed in the U.S.A. (Elsewhere, an IBM office or representative will be happy to forward your comments or you may mail them directly to the address in the Edition Notice on the back of the title page.)

SC26-4380-1

Reader's Comment Form

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, NY

POSTAGE WILL BE PAID BY ADDRESSEE

International Business Machines Corporation
Department J57
P.O. Box 49023
San Jose, CA 95161-9945

Fold and tape | Please do not staple | Fold and tape

IBM®