

Introduction

About This Section

This section is an alphabetical reference guide to the features and commands in Enable's SQL. It provides the usage and syntax of each SQL command supported by Enable and covers Enable's extensions to ANSI standard SQL. These extensions include syntactical and functional variances, as well as enhancements made to the language in the form of expanded features and functionality.

What is SQL?

Structured Query Language (SQL) is a relational database language that allows you to manage data across multiple data tables. It is a nonprocedural language, making it an easy language to write and read. Procedural languages, such as Pascal, COBOL or Enable's Procedural Language, require you to write programs that specify what data is being retrieved from a data system, and how this data is to be found. Writing these programs can be a lengthy and complex process. Using SQL, you need only specify what data is to be retrieved, and not how this retrieval should take place. Through a limited number of commands that you enter, SQL knows where to look for data, which indexes to use, and the most effective sequence of operations to perform. You therefore have fast direct access to data.

Enable's SQL

Enable's SQL is generally consistent with standards set forth by the American National Standard for Information Systems (ANSI). While the ANSI standard for SQL is expected to expand on a continuous basis, the current standard recognizes two levels. Level 1 covers the first set of guidelines developed for SQL, but does not address the language in its entirety. Level 2, which includes and enhances Level 1, sets a comprehensive standard for the complete language of SQL.

Enable's SQL covers all of ANSI Level 1. It includes most of the operations and features as defined in Level 2, and in many cases expands on and enhances those features.

As a database language, Enable's SQL can be used in two forms: *interactive* or *embedded*. To use SQL interactively, you type and execute your SQL statements on a query screen or use the SQL interactive menus to perform simple SQL operations. Embedded SQL is used in conjunction with the DBMS module. You can insert SQL statements into procedural language reports or otherwise use them in DBMS wherever procedural language commands are permitted (Where clause, post and prefield macros, etc.).

System Requirements

To run Enable's SQL in a DOS environment your computer system must be equipped with at least two megabytes of LIM 4.0 compatible expanded memory.

Syntax Notational Conventions

The notations used throughout this section to show SQL command syntax are based on the following conventions:

Words in uppercase letters are key words and are essential to the syntax. You must type in the exact text of these words. Since SQL key words are not case sensitive, you may type them in either uppercase or lowercase letters.

Words in lowercase letters represent variable names or expressions, which you replace with your own names or expressions. The explanation given under each command discusses the type of replacements that are valid.

An item between brackets ([]) is optional. Do not include the brackets in your typed statements.

Braces ({ }) surround a list of several items, from which you must select one item. Do not type the braces.

The vertical bar (|) separates distinct items in a list of choices. Do not include the vertical bar in your statement.

Items that may be repeated one or more times are surrounded by carets (^) and followed by a comma (,) and ellipses (...). You must include the comma between repetitions of the item, but do not type the carets or the ellipses.

A tilde (~) between two optional items indicates that at least one of the items must be included. Do not type the tilde in your statement.

Naming Conventions in SQL

Except for dbsystem names, all names you assign in SQL (to tables, columns, aliases, passwords, etc.) are governed by the same rules. Names must begin with an alphabet letter, can be any combination of alphanumeric characters, the underscore (_) and the dollar sign (\$), and not exceed 30 characters in length. None of SQL's reserved words (see REF2:APP:Reserved Words) may be used as names.

Dbsystem names are subject to the same rules as all other Enable file names. (See REF1:BA:Basic Operating Features:File Names for more information on assigning file names.) If you do not specify a directory path and an extension, the dbsystem file will be created in the default directory path and assigned a .DBS extension.

Admin Menu

You can use SQL's Admin Menu to insert comments into a dbsystem, rename a dbsystem or rebuild a dbsystem that has been damaged because of system failure. You can also use this menu to obtain on-line help on granting and revoking dbsystem and table privileges, altering and locking tables, and setting environment options.

To display the Admin Menu:

1. From the SQL Top Line Menu, select **A**admin.

For related topics, see SQ:Commenting on a Dbsystem, SQ:Rebuilding a Dbsystem, SQ:Renaming Components in a Dbsystem.

Commands

The following sections contain detailed information about each of the available SQL commands in alphabetical order.

ALTER TABLE

Use: The ALTER TABLE command allows you to change the structure of your dbsystem's base tables by adding, dropping, inserting, modifying and moving columns.

Syntax: ALTER TABLE table_name
 {ADD ^column_name data_type [data_constraints_list]^,...
 |DROP ^column_name^,...
 |INSERT ^column_number column_name data_type [data_constraints_list]^,...
 |MODIFY ^column_name [data_type]~[data_constraints_list]^,...
 |MOVE column_name TO column_number }

Details: Each ALTER TABLE command is followed by the name of the table to be affected, and then by the action you wish to perform (ADD, DROP, INSERT, MODIFY, or MOVE a column). You can add, drop, insert or modify several columns at a time, as long as individual column specifications are separated by commas.

Columns are defined by name, data type and optional data constraints. Valid column data types are CHAR, NUMERIC, DECIMAL, INTEGER, SMALLINT, FLOAT, REAL, DOUBLE PRECISION, MONEY, LOGICAL, DATE, TIME, VARCHAR, LONG VARCHAR, LONG, PHONE, STATE and ZIP.

Your data constraints list may contain one or more data constraints that further define what data is acceptable for the specified column. Valid elements for this list are NULL, NOT NULL, UNIQUE, NOT UNIQUE, MIN, MAX, DEFAULT, IN and USING. New columns are assumed to be NULL and NOT UNIQUE unless otherwise specified.

Use ALTER TABLE with ADD to add a new column to an existing table. You must specify the name of the column, the data type, and any optional data constraints desired.

Use ALTER TABLE with DROP to drop an existing column from a table. You need only specify the column name, not the entire column definition.

Use ALTER TABLE with INSERT to insert a new column into an existing table in a particular column position. You must specify a column name unique for that table, the column's data type and any additional data constraints desired. The new column will be inserted into the table based on the column number you specified, and the other columns will shift accordingly.

Use ALTER TABLE with MODIFY to change the width of a column that has a character data type (CHAR, VARCHAR or LONG). Specify the column name and data type and enter the new width.

Each column in a table is numbered sequentially based on the order of creation. Use ALTER TABLE with MOVE to change the position of columns in a table by assigning a new number to a specific column. Enter the column name and the new column number desired. (The system table SYSCOLUMNS stores the internal column number of each column in every table of the dbsystem.)

Example: To add the COUNTY column (maximum length=15characters) to the PERSONNEL table:

```
ALTER TABLE PERSONNEL ADD COUNTY CHAR(15)
```

To drop the STREET and CITY columns from the PERSONNEL table:

```
ALTER TABLE PERSONNEL DROP STREET, CITY
```

To insert a new column called BIRTH_YEAR (for required numeric data equal to or greater than 1949) into the third column position of the PERSONNEL table:

```
ALTER TABLE PERSONNEL INSERT 3
BIRTH_YEAR SMALLINT NOT NULL MIN 1949
```

To change the STREET field in the PERSONNEL table to a maximum length of 30 characters:

```
ALTER TABLE PERSONNEL MODIFY STREET VARCHAR(30)
```

To change the ZIPCODE column in the PERSONNEL table from the 1st column position to the 4th column position:

```
ALTER TABLE PERSONNEL MOVE ZIPCODE TO 4
```

For related topics, see SQ:Data Constraints, SQ:Data Types, and SQ:System Tables.

CLOSE cursor

Use: The CLOSE command is used only in embedded SQL. It is used to close a defined cursor set. It is always paired with the OPEN command and must be inserted between successive uses of the OPEN command.

Syntax: CLOSE cursor_name

Details: The *cursor_name* is the name assigned to a set of records defined by a DECLARE CURSOR command.

Example: To close the cursor set called XY, enter:

```
CLOSE XY
```

COMMENT ON

Use: The COMMENT ON command is used to attach descriptive comments to a base table, view, external table reference, index, column or synonym. Each system catalog table, SYSTABLES, SYSCOLUMNS and SYSINDEX, contains a REMARKS column that stores the descriptive comments you specify. Comments on a base table, view, synonym or external table reference are stored in the SYSTABLES catalog table; comments on a column in the SYSCOLUMNS table; and comments on an index in the SYSINDEX table.

Syntax: COMMENT ON
{TABLE table_name
|VIEW view_name
|EXTERNAL external_name
|SYNONYM synonym_name
|INDEX table_name.index_name
|COLUMN table_name.column_name}[@dbsystem]
IS char_string

Details: When commenting on a column or an index, you must specify the table where the column or index is located. The table name is appended to the front of the column or index name.

In a COMMENT ON statement, the dbsystem name must be specified if you are connected to more than one dbsystem.

The *char_string* represents the text of the comment. The text may be up to 255 characters in length and must be enclosed in quotation marks.

Example: Place the comment "Personnel information" on the PERSONNEL table:

```
COMMENT ON TABLE PERSONNEL IS "PERSONNEL INFORMATION"
```

Place the comment "Work number" on the IDNUM column of the PERSONNEL table:

```
COMMENT ON COLUMN PERSONNEL.IDNUM IS "WORK NUMBER"
```

For information on using the SQL interactive menus to comment on your dbsystem, refer to SQ:Commenting on your Dbsystem. For other related topics, see SQ:Component References and Identifiers and SQ:System Tables.

COMMIT

Use: The COMMIT command is used to make permanent all changes that have been made to the current dbsystem using the INPUT, INSERT, DELETE and UPDATE

commands. You can issue as many COMMIT statements as you like during a session. However, the COMMIT command is not necessary if the autocommit function is turned on during the session.

Once you have used the COMMIT command to save changes, you cannot use the ROLLBACK command to undo them.

Syntax: COMMIT [WORK]

Details: While WORK may be included in a COMMIT statement, it is optional and does not change the nature of the statement.

For information on saving your changes using the SQL interactive menus, see SQ:Saving Your Work. For other related topics, see SQ:Commands:ROLLBACK, SQ:Undoing Your Work, and SQ:Commands:SET AUTOCOMMIT.

CONNECT

Use: The CONNECT command is used to open and access a dbsystem.

Syntax: CONNECT [user_name[/password]] @dbsystem

Details: If a dbsystem was created with a specified user name, you need proper authorization to access that system. If you are the dbsystem administrator (DBA), you may connect to the dbsystem by specifying the user name (and any password) that you used to create the dbsystem. If you are not the DBA but have been granted CONNECT, RESOURCE or DBA privileges by the DBA, you may connect to the dbsystem by specifying your user name (and password if any).

You do not need authorization to connect to a system that has PUBLIC access, i.e. the owner did not specify a user name when the dbsystem was created.

If you are connecting to a dbsystem that is not in the current directory, you must specify the full file path in your CONNECT statement.

Example: Open the public access dbsystem called COMPANY.

```
CONNECT @COMPANY
```

Open the MANAGEMENT dbsystem located in the A: drive that was created by user AMY with the password KROY.

```
CONNECT AMY/KROY @A:MANAGEMENT
```

For information on using the SQL interactive menus to connect to a dbsystem, see SQ:Connecting to a Dbsystem. For other related topics, see SQ:Disconnecting from Your Dbsystem, SQ:Commands:DISCONNECT, and SQ:Commands:GRANT (on Dbsystem).

CREATE DBSYSTEM

Use: The CREATE DBSYSTEM command is used to create an SQL dbsystem.

Syntax: CREATE DBSYSTEM [user_name[/password]] @dbsystem [SAFETY (level)]

Details: SQL dbsystems can be created with or without specified ownership. To assign ownership, specify a user name, and if you wish, a password. You become the

dbssystem administrator (DBA) of the new dbssystem. Only the DBA or users who have been assigned privileges by the DBA are authorized to use the dbssystem.

If you create a dbssystem without specifying ownership, the dbssystem is designated as PUBLIC and becomes accessible to all users.

The CREATE DBSYSTEM statement must include a name for the dbssystem being created. Since your dbssystem is actually a file, the name you assign must follow the rules for all other Enable file names. See IN:File Names. If you do not specify a directory path and an extension, the dbssystem file will be created in the default directory path and assigned a .DBS extension.

When you create a dbssystem, you have the option of setting a data safety level that will determine your ability to recover data after a system failure. Safety can be set at 4 different levels – 0, 1, 2, 3 – where 0 is minimum safety and 3 is maximum. The higher the level of safety, however, the slower your system will be.

Example: Create the LIBRARY dbssystem with the user name KE_\$3DD.

```
CREATE DBSYSTEM KE_$3DD @LIBRARY
```

Create the MANAGERS dbssystem with user name MICKEY and password LOBE.

```
CREATE DBSYSTEM MICKEY/LOBE @MANAGERS
```

Create the CASES dbssystem in the directory D:\STATS with a .REF file extension, and set data safety at 2.

```
CREATE DBSYSTEM @D:\STATS\CASES.REF SAFETY (2)
```

For information on using the SQL interactive menus to create a dbssystem, see SQ:Creating a Dbssystem. For other related topics, see SQ:Safety Levels and SQ:SET SAFETY Command.

For information on other CREATE commands, see SQ:Commands:CREATE EXTERNAL, SQ:Commands:CREATE INDEX, SQ:Commands:CREATE SCHEMA, SQ:Commands:CREATE SYNONYM, SQ:Commands:CREATE TABLE AS, SQ:Commands:CREATE TABLE, and SQ:Commands:CREATE VIEW.

CREATE EXTERNAL

Use: This command creates a table reference in an SQL dbssystem for a file created outside of SQL's environment. Thereafter, the external file can be treated as an SQL table.

Syntax: CREATE EXTERNAL *table_name* AS *file_name*

Details: Any valid SQL table name can be used for *table_name*. The *file_name* is the name of the external file for which the SQL table reference will be created. Be sure to include the full file path where necessary.

The following types of files may be externally referenced with a CREATE EXTERNAL statement:

.DBF	Enable 2.0/dBASE II or Enable 3.0/dBASE III databases
.WK?	Lotus worksheets (read only)
.SSF	Enable spreadsheets (read only)

.WPF Enable word processing files (read only)
 .ASC ASCII files(read only)
 .DEF Enable database definition files
 .DTA PC-File III databases
 .DAT Condor databases
 .SBF Superbase databases
 .INF Informix databases

To create an external table reference for an ASCII file or an Enable word processing file, you must have already completed a field definitions file (.FDF) for the data. A field definitions file defines the way in which the data will be parsed into columns and rows when it is imported into SQL. Use the Database module to create the .FDF file. For detailed information, see REF1:DB:Import.

Example: Create a table reference called NEWHIRES for an Enable 2.0 dbsystem called HIRING.DBF.

```
CREATE EXTERNAL NEWHIRES AS HIRING.DBF
```

For information on using the SQL interactive menus to create an external table reference, see SQ:Creating an External Table Reference. For additional related information, see SQ:Importing External Files.

For information on other CREATE commands, see the following topics:
 SQ:Commands:CREATE INDEX, SQ:Commands:CREATE SCHEMA,
 SQ:Commands:CREATE SYNONYM, SQ:Commands:CREATE DBSYSTEM
 SQ:Commands:CREATE TABLE AS, SQ:Commands:CREATE TABLE, and
 SQ:Commands:CREATE VIEW.

CREATE INDEX

Use: This command creates an index on a specified column or columns of a base table, allowing for faster data retrieval.

Syntax: CREATE [UNIQUE] INDEX index_name ON table_name (^column [ASC|DESC]^,...)

Details: You create an index as UNIQUE if you wish to disallow duplicate values in the column(s) being indexed.

An index may be created on one or more columns. A CREATE INDEX statement must include the name of the column(s) being indexed, and the name of the table containing the column(s). If an index is created on more than one column, you should list the columns in the order you wish the data to be sorted. The first column will be treated as the primary sort key, the second column as the secondary sort key, and so forth.

To set the order of sort for each column being indexed, use ASC for ascending or DESC for descending. If no order is specified, indexed columns will be sorted in ascending order.

Example: Create a unique index called NAMENDX on the LASTNAME column of the ADDRESS table and sort by descending order.

```
CREATE UNIQUE INDEX NAMENDX ON ADDRESS (LASTNAME DESC)
```

Create an index called TITLE_NAME on the TITLE and NAME columns of the CONTACT table, sorting TITLE in descending order and NAME in ascending order.

```
CREATE INDEX TITLE_NAME ON CONTACT (TITLE DESC, NAME ASC)
```

For related topics, see SQ:Data Constraints and SQ:Indexes.

For information on other CREATE commands, see SQ:Commands:CREATE EXTERNAL, SQ:Commands:CREATE SCHEMA, SQ:Commands:CREATE DBSYSTEM, SQ:Commands:CREATE SYNONYM, SQ:Commands:CREATE TABLE AS, SQ:Commands:CREATE TABLE, and SQ:Commands:CREATE VIEW.

CREATE SCHEMA

Use: This command is used to grant RESOURCE privilege to a user, and may include CREATE, COMMENT ON and GRANT statements on behalf of that user.

Syntax: CREATE SCHEMA AUTHORIZATION user_name [/passwordIDENTIFIED BY password] [schema_list]

Details: You can assign a password to the user by appending the password to the user name (see SQ:Component References and Identifiers) or by using an IDENTIFIED BY clause.

The *schema_list* can include statements to create tables, views and indexes; to comment on tables, views or columns; and to grant privileges on tables or views created in the same statement. All tables, views and indexes created with a CREATE SCHEMA statement will automatically belong to the user identified in the statement.

No separators are required between statements in the schema list.

Example: Grant RESOURCE privilege to John, and assign him the password RYE.

```
CREATE SCHEMA AUTHORIZATION JOHN IDENTIFIED BY RYE
```

Grant RESOURCE privilege to John and assign him the password RYE. At the same time, create for him the tables A and B, with a unique index on the FD1 column of A. Also, create a view for John called TOTALS, which counts the number of records in table A. Finally, grant SELECT privileges to Anna on the view TOTALS.

```
CREATE SCHEMA AUTHORIZATION JOHN/RYE
CREATE TABLE A (FD1 INTEGER, FD2 FLOAT)
CREATE TABLE B (C1 CHAR(5), C2 SMALLINT)
CREATE UNIQUE INDEX FD1_INDX ON A (FD1)
CREATE VIEW TOTALS AS SELECT COUNT(*) FROM A
GRANT SELECT ON TOTALS TO ANNA
```

For related topics, see SQ:Commands:GRANT (on Dbsystem), SQ:Commands:GRANT (on Table), and SQ:Indexes.

For information on the use and syntax of other CREATE commands, see SQ:Commands:CREATE EXTERNAL, SQ:Commands:CREATE INDEX, SQ:Commands:CREATE DBSYSTEM, SQ:Commands:CREATE SYNONYM, SQ:Commands:CREATE TABLE AS, SQ:Commands:CREATE TABLE, and SQ:Commands:CREATE VIEW.

CREATE SYNONYM

Use: The CREATE SYNONYM command is used to create an alternate name for an existing table or view. Using a synonym, one user (with appropriate table privileges) can access another user's table without specifying the table owner's name each time.

Syntax: CREATE SYNONYM [PUBLIC.]synonym_name FOR table_name

Details: If a synonym is defined as PUBLIC, all dbsystem users can access the table through the synonym. Otherwise, only the user who created the synonym is authorized to use it.

The *table_name* must be the name of an existing table or view in the current dbsystem.

Example: Create an alternate name GOODS for the MERCHANDISE table that belongs to Sue.

```
CREATE SYNONYM GOODS FOR SUE.MERCHANDISE
```

Create a public synonym CE012 for the ADMIN_80 table.

```
CREATE SYNONYM PUBLIC.CE012 FOR ADMIN_80
```

For information on creating a synonym using the interactive menus, refer to SQ:Creating a Synonym. For other related topics, see SQ:Component References and Identifiers.

For information on the use and syntax of other CREATE commands, see SQ:Commands:CREATE EXTERNAL, SQ:Commands:CREATE INDEX, SQ:Commands:CREATE DBSYSTEM, SQ:Commands:CREATE SCHEMA, SQ:Commands:CREATE TABLE AS, SQ:Commands:CREATE TABLE, and SQ:Commands:CREATE VIEW.

CREATE TABLE

Use: The CREATE TABLE command is used to create a base table in a dbsystem and define columns for that table.

Syntax: CREATE TABLE table_name (^column_name data_type [data_constraints_list]^,...^[UNIQUE (column_list)^,...])

Details: A CREATE TABLE statement may define numerous columns for one table with commas separating individual column definitions. Columns are defined by name, data type and optional data constraints.

Valid column data types are CHAR, NUMERIC, DECIMAL, INTEGER, SMALLINT, FLOAT, REAL, DOUBLE PRECISION, MONEY, LOGICAL,

DATE, TIME, VARCHAR, LONG VARCHAR (also called LONG), PHONE, STATE and ZIP.

Your data constraints list may contain one or more data constraints that further define what data is acceptable for the specified column. Valid elements for this list are NULL, NOT NULL, UNIQUE, NOT UNIQUE, MIN, MAX, DEFAULT, IN and USING. New columns are assumed to be NULL and NOT UNIQUE unless otherwise specified.

You can create an index on individual columns by specifying UNIQUE as one of your column's data constraints. You can also create a single or multi-key index by specifying UNIQUE after your list of column definitions, followed by the name of the column(s) on which you wish to create the index. Individual column names must be separated by commas.

Example: Create a table called APARTMENTS with separate columns for cost of rent, location and number of rooms.

```
CREATE TABLE APARTMENTS
  (RENT MONEY NOT NULL,
   ROOMS SMALLINT,
   LOCATION CHAR(15) DEFAULT "ROUND LAKE")
```

Create a table called SALES with fields for item, salesperson and quantity, with a unique, multi-key index on the ITEM and SALESPERSON columns.

```
CREATE TABLE SALES
  (ITEM SMALLINT NOT NULL,
   SALESPERSON CHAR(10) NOT NULL,
   QUANTITY SMALLINT MIN 1,
   UNIQUE (ITEM, SALESPERSON))
```

For related topics and additional information, see SQ:Data Constraints, SQ:Data Types, SQ:Indexes, and SQ:Table Types.

For information on the use and syntax of other CREATE commands, see SQ:Commands:CREATE EXTERNAL, SQ:Commands:CREATE INDEX, SQ:Commands:CREATE DBSYSTEM, SQ:Commands:CREATE SCHEMA, SQ:Commands:CREATE TABLE AS, SQ:Commands:CREATE SYNONYM, and SQ:Commands:CREATE VIEW.

CREATE TABLE AS

Use: This command allows you to create a new table and populate it with the results of a query performed on another table or tables.

Syntax: CREATE TABLE table_name [(^column_name^,...)] AS select_statement

Details: You can include an optional list of column names if you want to assign new column names to the data you import into the new table. Otherwise the columns you specify in the SELECT statement will be imported into the new table with their original file names. Column data types and data constraints will carry over into the new table. If you include the optional column list, the number of names in the list must be equal to the number of fields selected in the SELECT statement.

The *select_statement* is a query performed on an existing table, following the rules and syntax of the SELECT command.

Example: Create a table called FOREIGN to contain all the records in the STUDENTS table where the LANGUAGE column = "French".

```
CREATE TABLE FOREIGN
  AS SELECT * FROM STUDENTS
  WHERE LANGUAGE = "French"
```

Create a table called FOREIGN with fields for name, address and telephone number and fill it with data from the LNAME, STREET and PHONENO columns of the STUDENTS table for each record where the LANGUAGE column = "French".

```
CREATE TABLE FOREIGN
  (NAME, ADDRESS, TELEPHONE)
  AS SELECT LNAME, STREET, PHONENO FROM
  STUDENTS
  WHERE LANGUAGE = "French"
```

Import an Enable Database file into a dbsystem without using the CREATE EXTERNAL command.

```
CREATE TABLE tablename
  AS SELECT select clause (or * for all)
  FROM @database_name.extension
```

For related topics, see SQ:Commands:CREATE TABLE and SQ:Commands:SELECT.

For information on the use and syntax of other CREATE commands, see SQ:Commands:CREATE EXTERNAL, SQ:Commands:CREATE INDEX, SQ:Commands:CREATE DBSYSTEM, SQ:Commands:CREATE SCHEMA, SQ:Commands:CREATE TABLE, SQ:Commands:CREATE SYNONYM, and SQ:Commands:CREATE VIEW.

CREATE VIEW

Use: This command is used to create a view of data from one or more tables. A view is a virtual table. It does not store actual data but allows you to view columns from different tables at the same time, and to create temporary columns based on mathematical calculations involving other columns. Views are also useful for data security since you can use them to display selective parts of a data table only.

Syntax: CREATE VIEW view_name [(^column_name^,...)] AS select_statement [WITH CHECK OPTION]

Details: You can assign new column names to each column of the view. Columns will be named in the order they appear in the *select_statement*.

The *select_statement* is a SELECT command query that identifies the data to be contained in the view.

Including a CHECK OPTION in a CREATE VIEW statement allows you to maintain the integrity of a view. When a view has a CHECK OPTION, you cannot

use the view to effect updates or inserts to the table that do not meet the criteria of the SELECT statement used to create the view.

Example: Create a view called FEMALES to show all rows in the DEPTS table where the GENDER column = "F".

```
CREATE VIEW FEMALES
  AS SELECT * FROM DEPTS
  WHERE GENDER = "F"
```

Create a view called FEMALES with the columns FNAME, LNAME, GENDER and YEARS to show data from columns FIRST, LAST, EMP_AGE and GENDER of the DEPTS table where the GENDER column = "F" and the EMP_AGE column is less than 40. Protect the view from being updated with any records but those where GENDER = "F" and EMP_AGE is less than 40.

```
CREATE VIEW FEMALES
  (FNAME, LNAME, GENDER, YEARS)
  AS SELECT FIRST, LAST, GENDER, EMP_AGE
  FROM DEPTS WHERE GENDER = "F"
  AND EMP_AGE <40
  WITH CHECK OPTION
```

For related topics and additional information, see SQ:Table Types and SQ:Commands:SELECT.

For information on the use and syntax of other CREATE commands, see SQ:Commands:CREATE EXTERNAL, SQ:Commands:CREATE INDEX, SQ:Commands:CREATE DBSYSTEM, SQ:Commands:CREATE SCHEMA, SQ:Commands:CREATE TABLE, SQ:Commands:CREATE SYNONYM, and SQ:Commands:CREATE TABLE AS.

DECLARE CURSOR

Use: This command is used only in embedded SQL. It assigns a cursor name to a set of individual rows in a table that satisfy conditions set forth in a SELECT statement. Once a cursor set is defined, you can use other cursor commands to scan the records in the set and select specific records for use.

Syntax: DECLARE cursor_name [SCROLL] CURSOR FOR select_statement

Details: If SCROLL is included in a DECLARE CURSOR statement, the data in the defined cursor set may be scrolled. This allows you to use the FETCH cursor command (once the set has been opened) to retrieve records relative to FIRST, LAST, PREVIOUS, NEXT, ABSOLUTE, RELATIVE and CURRENT cursor position. If SCROLL is not included in a DECLARE CURSOR statement, the FETCH command can only be used with NEXT.

Example: Define a cursor set XY as the data in columns AB, BC and CD of table ACT_ONE for each record where the value of column AB is greater than 5.

```
DECLARE XY CURSOR FOR
  SELECT AB, BC, CD FROM ACT_ONE
  WHERE AB>5
```

Define a scroll cursor set XY as the data in columns AB, BC and CD of table ACT_ONE and sort the records on column BC.

```
DECLARE XY SCROLL CURSOR FOR
    SELECT AB, BC, CD FROM ACT_ONE
    ORDER BY BC
```

For related topics and additional information, see SQ:Embedding SQL, SQ:Commands:FETCH, and SQ:Commands:SELECT.

DELETE

Use: The DELETE command is used to delete selected rows of data from a table or view.

Syntax: DELETE FROM table_name [WHERE condition]

Details: The *table_name* is the name of the table from which the data will be deleted. The WHERE condition is used to specify which rows will be deleted. If no WHERE condition is present, all rows will be deleted from the table.

Example: Delete all records from the STATISTICS table of persons who have retired.

```
DELETE FROM STATISTICS WHERE RETIRED = "YES"
```

Delete all records from the STATISTICS table of persons who have retired and who are not listed as teachers in the CODES table.

```
DELETE FROM STATISTICS WHERE RETIRED = "YES"
    AND NAME NOT IN
    SELECT NAME FROM CODES
    WHERE OCCUPATION = "TEACHER"
```

DELETE FROM...WHERE CURRENT

Use: This command is a cursor command used only in embedded SQL. It is used to delete a record from a table based on the current cursor position in a previously defined cursor set. It must always be preceded by a FETCH cursor command.

Syntax: DELETE FROM table_name WHERE CURRENT OF cursor_set

Details: A cursor set is a set of records in a table that has been identified by a DECLARE CURSOR statement. In a DELETE FROM statement, you must specify the name of the cursor set, and the corresponding table name.

WHERE CURRENT selects the record in the cursor set that is at the current cursor position. The cursor position must previously have been set with a FETCH cursor command.

Example: Delete the record from the MISSION table that corresponds to the current cursor position in the cursor set called SECRET.

```
DELETE FROM MISSION WHERE CURRENT OF SECRET
```

For related topics, see SQ:Embedding SQL and SQ:Commands:FETCH.

DISCONNECT

Use: This command is used to close down and quit an SQL dbsystem.

Syntax: DISCONNECT [@dbsystem]

Details: If you do not specify a dbsystem name in a DISCONNECT statement, you will be disconnected from all dbsystems to which you are currently connected.

Example: Close the dbsystem called NUMBERS.

```
DISCONNECT @NUMBERS
Close all currently open dbsystems.
DISCONNECT
```

For information on using the interactive menus to disconnect from a dbsystem, see SQ:Disconnecting from a Dbsystem.

DROP

Use: This command is used to delete an existing dbsystem, table, view, external table reference, synonym or index.

Syntax: DROP {DBSYSTEM @dbsystem |TABLE table_name |VIEW view_name |EXTERNAL table_name |SYNONYM synonym_name |INDEX index_name ON table_name }

Details: If you are dropping an index, you must specify the name of the table that is referenced by the index.

Example: Get rid of the dbsystem called REPORTERS.

```
DROP DBSYSTEM @REPORTERS
Remove the NONAME index from the ANONYMOUS table.
DROP INDEX NONAME ON ANONYMOUS
```

FETCH

Use: This command is used to retrieve an individual data record from a specified position in a previously defined cursor set. A FETCH command can only be used won a cursor set that has been established with an OPEN cursor command.

Syntax: FETCH [{FIRST |LAST |NEXT |PRIOR |ABSOLUTE value |RELATIVE value |CURRENT }] cursor_name INTO fields_list

Details: When you open a cursor set, the cursor is automatically positioned just before the first record of the set. You can fetch a record relative to the current cursor position by specifying FIRST, LAST, NEXT, PRIOR, ABSOLUTE, RELATIVE or CURRENT. If no cursor position is specified, the FETCH position defaults to NEXT, and the cursor is positioned at the record following the current position.

ABSOLUTE and RELATIVE must be followed by a value expression that returns a numeric value, equivalent to a record position in the cursor set. If an ABSOLUTE value is specified, records will be numbered sequentially, starting

with the first record if the value is positive, and the last record if the value is negative. If the value is RELATIVE, records will be numbered relative to the current cursor position.

You must specify the name that was assigned to the cursor set in a previously executed DECLARE CURSOR statement.

For *fields_list*, you must enter the list of previously defined local fields into which the record being fetched will be placed. Data from each column of the record will be entered into the local fields in the order they appear. If the record contains more columns than the number of local fields, the excess columns will be ignored. For more information on local fields, see SQ:Embedding SQL.

Example: Retrieve the record in the second to last position of the cursor set W02 and enter the data into the local fields A1, A2, A3.

```
FETCH ABSOLUTE -2 W02 INTO :A1, :A2, :A3
```

With the cursor currently at the fourth position in the cursor set W02, retrieve the record in the third position, and enter the data into the local fields A1, A2, A3.

```
FETCH PRIOR W02 INTO :A1, :A2, :A3
```

For related topics, see SQ:Embedding SQL, SQ:Commands:OPEN CURSOR, and SQ:Commands:DECLARE CURSOR.

GRANT (on Dbsystem)

Use: This command is used to accord levels of authority on a dbsystem to specific users. It is also used to change a user's password.

Syntax: GRANT ^privilege^,... TO user_list [IDENTIFIED BY password_list] [@dbsystem]

Details: The three kinds of privileges that may be awarded with this GRANT statement are DBA, RESOURCE and CONNECT. More than one privilege may be granted in a GRANT statement.

CONNECT Privilege is the lowest level of authority that may be granted on a dbsystem. A GRANT CONNECT statement simply registers a user, with or without a password, as someone who may connect to the dbsystem. If you do not issue another GRANT statement detailing what kinds of privileges the user has on specific tables, the user will only have access to tables that were created as PUBLIC.

RESOURCE Privilege enables a user to connect to a dbsystem, create tables and views in that dbsystem, and grant privileges on those tables and views.

DBA Privilege is the highest level of privilege that a user can have on a dbsystem. With DBA authority, a user can do everything that the dbsystem administrator can do, including the creation of new users for the dbsystem.

The *user_list* consists of one or more users who are being granted authority on the dbsystem. Commas should be used to separate the different users in the list. All users must be identified in the same format, either by user names only, or by user

names and passwords. A user name with a password is entered in the format *user_name|password*.

You can use the optional IDENTIFIED BY clause to assign one password to all users in the *user_list* or to assign an individual password to each user in the list. If the *password_list* contains only one password, that password is assigned to all users in the *user_list*. If the *password_list* contains more than one password, the number of passwords must be equal to the number of users in the *user_list*. Each password is then paired with each user in consecutive order. If you use the IDENTIFIED BY clause, you cannot include passwords in the *user_list*.

The IDENTIFIED BY clause can also be used to change the password of a current user or users. In the *user_list*, enter the names of the user(s) to be assigned the new password(s), and specify the new password(s) in the IDENTIFIED BY clause.

If only one dbsystem is currently open, you need not specify the dbsystem name, since a GRANT statement automatically references the current dbsystem.

However, you must specify a dbsystem name if you are connected to more than one dbsystem.

Example: Grant CONNECT privileges on the RECORDS dbsystem to users Terri and Louis with passwords TSF and LJC respectively.

```
GRANT CONNECT TO TERRI/TSF, LOUIS/LJC @RECORDS
```

Grant RESOURCE privileges to users Joan and Denise who share the password REP.

```
GRANT RESOURCE TO JOAN, DENISE IDENTIFIED BY REP
```

Assign a new password SECRET to user Denise who already has RESOURCE privileges on the dbsystem.

```
GRANT RESOURCE TO DENISE IDENTIFIED BY SECRET
```

GRANT (on Table)

Use: This command is used by the owner of a view or table to grant specific privileges on that view or table to other authorized users of the dbsystem.

Syntax: GRANT ^privilege^,... ON {table_name|view_name} TO {user_list|PUBLIC} [WITH GRANT OPTION]

Details: The privileges that may be granted on a table are SELECT, ALTER, DELETE, INDEX, INSERT, UPDATE and ALL. More than one privilege may be awarded in a GRANT statement if commas are used to separate the items.

SELECT Privilege allows the user to select records from the table.

ALTER Privilege allows a user to add, drop, insert, move or modify columns in the table.

DELETE Privilege allows a user to delete rows from the table.

INDEX Privilege allows a user to create indexes and to drop any index he or she created.

INSERT Privilege permits a user to insert rows into the table.

UPDATE Privilege may be comprehensive or limited. Users who have comprehensive UPDATE authority on a table may change data in any column of the table. Users with limited UPDATE authority may only update specific columns in a table.

To grant limited UPDATE authority, enter the names of the columns on which you wish to grant the authority, using the syntax: *GRANT UPDATE (column_list)*. Individual names in the list must be separated by commas. If your GRANT UPDATE statement does not include this column list, UPDATE authority will be granted on all columns of the table.

ALL Privilege is the highest privilege that can be granted on a table. It gives the user SELECT, UPDATE, ALTER, INDEX, DELETE and INSERT privileges on the table.

The table or view name specified in the GRANT statement must be the name of an existing base table or view.

Table privileges may be granted to PUBLIC or to a list of users identified by their user names. Privileges granted to PUBLIC may be used by any user who is able to connect to the system. If a list of users is entered, individual user names must be separated by commas.

You can grant privileges on your table to a user and allow that user to pass on those privileges to other users. By using the WITH GRANT OPTION clause, you give authority to the users identified in the *user_list* (or to all system users if PUBLIC was specified) to grant to other users the same table privileges they have been accorded.

Example: Allow users Henry and Virginia to be able to select from and insert records into the CORRECTIONS table.

```
GRANT SELECT, INSERT ON CORRECTIONS TO HENRY, VIRGINIA
```

Let user Henry be able to update the data in the DATEREC and DATEFIN columns of the CORRECTIONS table, and give him the authority to pass on that privilege to other users.

```
GRANT UPDATE (DATEREC, DATEFIN) ON CORRECTIONS TO HENRY
WITH GRANT OPTION
```

For related topics, see SQ:Commands:SELECT, SQ:Commands:ALTER TABLE, SQ:Commands:DELETE, SQ:Indexes, SQ:Commands:INSERT, and SQ:Commands:UPDATE.

INPUT

Use: The INPUT command is used to enter multiple rows of data into an existing table.

Syntax: INPUT table_name [(column_list)] values_list [END]

Details: The *table_name* must be the name of an existing base table.

If you do not include the optional *column_list*, values will be input sequentially into the columns of the table based on the order that columns appear in the table.

The first value will be entered into the first column, the second value into the second column, and so forth, until all values have been input.

Include a list of columns if you want data to be input into selective columns and in a particular column order. Column names in the list should be separated by commas. Any column that is excluded from the list will receive either the default value of that column (see SQ:Data Constraints) or the value NULL, for each row of data input. If no value is input for a column defined as NOT NULL, SQL returns an error message.

The number of input values you list should be divisible by the number of columns in the table, or by the number of columns in the column list if a column list is used. Commas must be used to separate individual values of the same row. Values must match the data type and data constraints of the column into which they are being input. If a row being input contains a column with no data, enter the value as NULL. All values that are character strings must be enclosed in quotation marks.

You may include the optional word END to mark the end of the list of values.

Example: The STOCK table keeps track of items in stock. It contains four columns in the order ITEM, UNITCOST, QUANTITY, DATE_ASSESSED.

Input four rows of data into the STOCK table.

```
INPUT STOCK
"ORGANIZER", 11.5, 40, 10/12/89
"BOOKENDS", 6.25, 52, 10/16/89
"PENHOLDER", NULL, 18, 10/16/89
"RULER", 1.5, 51, 10/18/89
```

Input two rows of data into the DATE_ASSESSED, ITEM and QUANTITY fields of the STOCK table.

```
INPUT STOCK (DATE ASSESSED, ITEM, QUANTITY)
10/18/89, "PENCILS_BX", 39
10/18/89, "ERASER", 64
END
```

For related topics, see SQ:Commands:CREATE TABLE, SQ:Data Types, SQ:Data Constraints, and SQ:Indexes.

INSERT

Use: This command is used to insert one row of values into a table, or to insert multiple rows of data from another table.

Syntax: INSERT INTO table_name [(column_list)] {VALUES (values_list)
|select_statement}

Details: The *table_name* is the name of an existing base table.

For an explanation of the optional *column_list* or on entering values, see SQ:Commands:INPUT.

You can enter the row of values to be inserted, or use a SELECT statement to copy rows of data from another table.

Example: Insert a set of values into the ITEM and QUANTITY columns of the STOCK table.

```
INSERT INTO STOCK (ITEM, QUANTITY)
VALUES ("SHARPENER", 37)
```

Select all records from the PURCHASES table where the SALES column = NULL and insert those records into the ITEM, QUANTITY and UNIT_COST fields of the STOCK table.

```
INSERT INTO STOCK (ITEM, QUANTITY, UNIT_COST)
SELECT ITEM, NUMBER, PRICE FROM PURCHASES
WHERE SALES IS NULL
```

For related topics, see SQ:Commands:SELECT.

LOCK TABLE

Use: This command allows you to temporarily lock one or more tables to prevent other users from having access to the table(s) you are currently using. A table is automatically unlocked whenever a COMMIT or ROLLBACK command is executed on the table. You can also unlock a locked table by using the command: LOCK ^table_name^ SHARE.

Syntax: LOCK TABLE ^table_name^ EXCLUSIVE [NOWAIT]

Details: The *table_name* is the name of the table you want to lock.

If the table to be locked is currently being used by another user, the LOCK statement will be put on hold until the table is free. Include the NOWAIT option if you want the program to automatically cancel the LOCK statement if the table is in use.

If you do not include the NOWAIT option, you can cancel a pending LOCK statement at any time by pressing **Ctrl/Break**.

Example: Lock the FLIERS table at the earliest opportunity.

```
LOCK TABLE FLIERS
```

For related topics, see SQ:Commands:COMMIT and SQ:Commands:ROLLBACK.

OPEN cursor

Use: This command executes the SELECT statement of a DECLARE CURSOR statement in order to retrieve specified records from a table and designate them as a cursor set.

Syntax: OPEN cursor_name

Details: The cursor name is the name given to a defined cursor set.

Example: Retrieve the records for the cursor set called PW.

```
OPEN PW
```

For related topics, see SQ:Commands:SELECT, SQ:Commands:DECLARE CURSOR, and SQ:Commands:CLOSE CURSOR.

PAUSE

Use: This command is used in embedded SQL to pause a report for keyboard action.

Syntax: PAUSE [char_string]

Details: You can enter the text of the message you want displayed in the status line when the report is paused. The text you enter must be enclosed in quotation marks. If no text is entered, the default character string, "Press any key to continue", is displayed.

Example: Pause the current report and display the message "Please press the space bar."

```
PAUSE "PLEASE PRESS THE SPACE BAR"
```

REBUILD

Use: This command is used to compress a dbsystem, or to rebuild a dbsystem following a system crash. You cannot be connected to the dbsystem you wish to rebuild.

Syntax: REBUILD dbsystem dbsystem

Details: You must enter the name of the dbsystem you wish to rebuild, followed by a new dbsystem name. When the system is rebuilt, it will be assigned the new name.

Example: Rebuild the dbsystem IMAGES with the name IMAGES1.

```
REBUILD IMAGES IMAGES1
```

For information on using the SQL interactive menus to rebuild a dbsystem, see the topic SQ:Rebuilding a Dbsystem.

RENAME

Use: This command is used to assign a new name to a table, view, synonym, external reference, index or column.

Syntax: RENAME {COLUMN table_name.column_name TO column_name
|EXTERNAL table_name TO table_name |INDEX table_name.index_name TO
index_name |SYNONYM synonym_name TO synonym_name |TABLE
table_name TO table_name |VIEW view_name TO view_name}

Details: You must enter the current name of the column, external reference, index, synonym, table or view, followed by the new name you wish to assign. When specifying the current names of indexes and columns, you must indicate the table to which they belong.

Example: Change the name of the WORKERS table to EMPLOYEES.

```
RENAME TABLE WORKERS TO EMPLOYEES
```

Change the name of the WAGE column in the EMPLOYEES table to SALARIES.

```
RENAME COLUMN EMPLOYEES.WAGE TO SALARIES
```

REVOKE (on Dbsystem)

Use: This command is used to revoke privileges previously awarded on a dbsystem with a GRANT statement.

Syntax: REVOKE privilege FROM user_list [@dbsystem]

Details: You can revoke any one of the three privileges previously granted to a user with a GRANT (on dbsystem) statement. You can revoke equal or lesser privileges than a user has been accorded, but not greater. For example, if a user has RESOURCE privilege, you can revoke RESOURCE or CONNECT privileges from that user, but not DBA. If you revoke DBA privilege from a user, the user's privilege level drops to RESOURCE. If you revoke RESOURCE privilege, the user maintains CONNECT privilege. Whenever CONNECT privilege is revoked from a user, that user is dropped from the dbsystem.

The user list is a list of those users from whom you are revoking the privilege. Users in the list are identified by user name only.

If you are connected to more than one dbsystem, your REVOKE statement must include the name of the relevant dbsystem.

Example:

Users Ted and Bob have RESOURCE privilege on the ITEMS dbsystem; change their privilege level to CONNECT.

```
REVOKE RESOURCE FROM TED, BOB @ITEMS
```

User Lloyd has RESOURCE privilege; delete him from the dbsystem.

```
REVOKE CONNECT FROM LLOYD
```

For related topics, see SQ:Commands:GRANT (on Dbsystem).

REVOKE (on Table)

Use: This command is used to revoke privileges previously awarded on a table or view with a GRANT statement.

Syntax: REVOKE ^privilege^,... ON {table_name|view_name} FROM {user_list|PUBLIC}

Details: The privileges that may be revoked on a table are SELECT, ALTER, DELETE, INDEX, INSERT, UPDATE and ALL. You can revoke a table privilege from a user only if that user was previously granted that privilege with a GRANT statement. See SQ:Commands:GRANT (on Table) for an explanation of the different types of table privileges.

The table or view name specified in the REVOKE statement must be the name of an existing base table or view.

You can revoke the privilege(s) of more than one user at a time by specifying several user names in the *user_list*. If a list of users is entered, individual user names must be separated by commas. If table privileges were previously granted to all users of the dbsystem (i.e. to PUBLIC), you can revoke the privileges of all users by specifying PUBLIC instead of identifying users individually.

Example: Revoke INSERT privileges on the CORRECTIONS table from Henry and Virginia.

```
REVOKE INSERT ON CORRECTIONS FROM HENRY, VIRGINIA
Take away public access to the AMENDMENTS table.
REVOKE ALL ON AMENDMENTS FROM PUBLIC
```

ROLLBACK

Use: This command is used to undo all changes made to the dbsystem since you opened the dbsystem or since the last COMMIT command was issued.

Syntax: ROLLBACK [WORK]

Details: WORK is an optional word that does not change the nature of the statement. The ROLLBACK command is ineffective if AUTOCOMMIT is turned on. Whenever the ROLLBACK command is issued, all locks will be unlocked. If you do not issue a ROLLBACK command before disconnecting from a dbsystem, the DISCONNECT command will cause all work to be committed.

For information on using the SQL interactive menus to perform a Rollback operation, see SQ:Undoing Your Work. For other related topics, see SQ:Commands:COMMIT, SQ:Commands:SET AUTOCOMMIT, and SQ:Saving Your Work.

SELECT

Use: This command is used to retrieve specified records from one or more data tables. SELECT statements may be used alone, embedded into CREATE TABLE AS, CREATE VIEW, DECLARE CURSOR and INSERT statements, or used in subqueries.

Syntax: SELECT {[DETAILED [ALL]][DISTINCT]} {**|^column_expression [AS alias]^,...|^table.{**|^column_expression]^,...} [INTO column_expression_list] FROM ^table_reference_name [alias]^,... [WHERE condition] [GROUP BY ^{column_number|column_name}^,...] [HAVING value_expression] [ORDER BY ^{column_number|column_name} [ASC|DESC]],...

Details: If you specify the optional key word DETAILED, you must use a GROUP BY clause or an aggregate function. The DETAILED option allows you to see all the elements that make up a group or aggregate. DETAILED should not be used in conjunction with DISTINCT.

DISTINCT and ALL perform opposite actions in a SELECT statement, and cannot be used together. DISTINCT is used to suppress duplicate records in the select set, and display a set of unique records. ALL displays all records in the select set, allowing duplicate rows. You may specify DETAILED with ALL.

The key word SELECT is immediately followed by one of several select options. If you specify "*", all columns are selected. If you specify "**", all columns plus two system record columns are selected. One system record column (SYS:RECORD) shows the number of each row as it appears in the selection. The

other system record column (*table_name*.SYS:RECORD) shows the number of each table as it is actually positioned in the dbsystem.

You can specify a list of one or more column names or expressions to be selected. If you are selecting from more than one table, and some columns have the same name, you should avoid ambiguity by prefixing the column name with the table name (*table_name.column_name*). Column expressions are logical, arithmetic, aggregate and nonaggregate expressions involving one or more column names.

The AS option allows you to assign an alias to each column name or expression, to be displayed as the column heading when the data is selected. If you use an alias that contains spaces or special characters, you should enclose it in Quotation marks.

A SELECT statement must include a FROM clause with a table reference list. The table reference list indicates the source of the data to be selected, and can contain one or more names of tables, views, synonyms, table aliases or external table references. Elements in the list must be separated by commas. If the list contains more than one element, and you want to select all columns from one table reference but not from others, suffix the table name with “.” or “.*”.

An optional WHERE condition sets the criteria for selecting particular records. WHERE clauses can consist of simple comparisons, logical and comparison operators, and subqueries of varying complexity.

The optional GROUP BY clause should be positioned after the WHERE clause, or after the FROM clause if no WHERE clause is present. A GROUP BY clause is used to divide data into groups based on the value of a particular column. The columns on which data is grouped are specified either by column name or by the number of the column as it appears in the select list.

The HAVING option is used in conjunction with a GROUP BY clause to further qualify groups of data. A HAVING clause contains a value expression that compares an aggregate value of a group of records against a constant, expression, subquery or other aggregate value.

When the optional ORDER BY clause is used, it must be the last clause in a SELECT statement. The ORDER BY clause allows records to be sorted in ascending or descending order on one or more columns. The columns on which the data is to be sorted are identified either by column name or by the number of the column as it appears in the select list. Use the optional key words ASC or DESC after each column name or number to determine whether the column should be sorted in ascending or descending order. If the order is not specified, the column will be sorted in ascending order.

Example: Retrieve all the records in the SALES table where the PRICE field is greater than 15,000.

```
SELECT * FROM SALES WHERE PRICE >15000
```

Retrieve the title and author of all books in the LIBRARY table that were published before 1960, eliminate all duplicate rows and sort them by author.

```
SELECT DISTINCT TITLE, AUTHOR FROM LIBRARY
WHERE PUBLISH_DATE <1960
ORDER BY AUTHOR
```

For related topics, see SQ:Commands:CREATE TABLE AS, SQ:Commands:CREATE VIEW, SQ:Commands:DECLARE CURSOR, and SQ:Commands:INSERT.

SELECT...FOR UPDATE

Use: This command is used to lock specific rows in a table so they can be updated with an UPDATE statement.

Syntax: SELECT statement FOR UPDATE OF column_list [NOWAIT]

Details: The SELECT statement is any valid statement that follows the rules and syntax outlined under the SELECT command.

The column list consists of the names of the columns you plan to update after you place a lock on the table. Column names in the list should be separated by commas.

The NOWAIT option works the same as in the LOCK TABLE command. If the NOWAIT option is included, a SELECT FOR UPDATE statement will be cancelled if the table specified in the SELECT statement is being used by another user. If you don't specify NOWAIT, the program will be suspended until it is able to process the statement. To cancel the procedure and resume normal operation, press **Ctrl/Break**.

Example: Lock all records in the EMPLOYEES table where the TITLE column = "Manager" so the SALARY and PERCENTAGE columns can be updated.

```
SELECT * FROM EMPLOYEES WHERE TITLE = "MANAGER"
FOR UPDATE OF SALARY, PERCENTAGE
```

For related topics, see SQ:Commands:SELECT, SQ:Commands:LOCK TABLE, and SQ:Commands:UPDATE.

SET AUTOCOMMIT

Use: This command is used to turn the AUTOCOMMIT function on and off. When AUTOCOMMIT is on, all changes made to the data in the current data tables are immediately saved. When AUTOCOMMIT is off, changes to data are not saved unless you issue a COMMIT command or disconnect from the dbssystem. When AUTOCOMMIT is turned on, the ROLLBACK command is ineffective.

Syntax: SET AUTOCOMMIT {ON|OFF}

For related topics, see SQ:Commands:COMMIT and SQ:Commands:ROLLBACK.

SET EDITERROR

Use: This command is used only in interactive SQL. With EDITERROR turned on, if you try to execute a statement that contains an error, an error message displays in the status line and the statement remains on screen so it can be amended. When EDITERROR is off, if a statement contains an error, an error message displays in

the status line and the statement is cleared from the screen. Setting EDITERROR off is particularly useful for macro execution.

Syntax: SET EDITERROR {ON|OFF}

SET NOWAIT

Use: This command is an environment setting that determines the action to be taken if SQL cannot process a statement due to locked tables. If NOWAIT is turned on, the command is aborted and an error message is returned. If NOWAIT is turned off, the statement will be put on hold until execution can be completed. Once a statement has been put on hold, you cannot process any other statements until the pending statement has been executed. To cancel a pending procedure, press **Ctrl/Break**.

Syntax: SET NOWAIT {ON|OFF}

For related topics, see SQ:Commands:LOCK TABLE.

SET SAFETY

Use: This command allows you to change the safety level of an existing dbsystem from the safety level set when the dbsystem was created. Only the dbsystem administrator (creator) of a dbsystem is authorized to change its safety level.

Syntax: SET SAFETY (level) [@dbsystem]

Details: The level you specify must be one of SQL's four safety levels: 0, 1, 2 or 3. For details on each of the four levels, see the topic SQ:Safety Levels.

If you are connected to more than one dbsystem, your statement must include the name of the dbsystem for which you are changing the safety level. If you are connected to only one dbsystem, you need not include the dbsystem name.

Example: Change the safety level of the MACHINES dbsystem to 1.

```
SET SAFETY (1) @MACHINES
```

For related topics, see SQ:Commands:CREATE DBSYSTEM.

SET TRACE

Use: This command is used only in embedded SQL, and controls message display in the current screen's status line. When TRACE is turned on, the status line is constantly updated as commands are executed. When TRACE is turned off, SQL messages do not display in the status line.

Syntax: SET TRACE {ON|OFF}

UNDECLARE CURSOR

Use: This command is used to void the cursor established by a DECLARE CURSOR statement and allow a new cursor to be defined using the same name.

Syntax: UNDECLARE CURSOR cursor_name

Example: Eliminate the cursor set defined as XY.

```
UNDECLARE CURSOR XY
```

For related topics, see SQ:Commands:DECLARE CURSOR.

UPDATE

Use: This command allows you to change existing data in a table.

Syntax: UPDATE table_name SET ^column_name = expression^,... [WHERE condition]

Details: The *table_name* is the name of the table to be updated. The *column_name* is the name of the column in the table that you want to change, and the *expression* determines the new data that will be input into the column. You may change more than one column at a time, using commas to separate individual column settings.

Expressions may consist of column names, arithmetic operators, character and numeric constants, and the value NULL. Each expression must return the new data value you wish to be input into the column. The derived values must be compatible with the data type of the column into which they are being input.

The optional WHERE condition establishes the search criteria that determine which records in the table will be affected by the UPDATE statement. If the statement contains no WHERE condition, all records in the table will be updated.

Example: Change the price of vases and mugs to \$5.00 in the MERCHANDISE table.

```
UPDATE MERCHANDISE SET UNIT PRICE = 5  
WHERE ITEM ="VASE" OR ITEM ="MUG"
```

In the EMPLOYEES table, change Maria Costa's title to "Director" and set her salary at \$75,000.

```
UPDATE EMPLOYEES  
SET TITLE="DIRECTOR", SALARY=75000  
WHERE LNAME ="COSTA" AND FNAME="MARIA"
```

For related topics, see SQ:Data Types and SQ:Operators.

WHENEVER

Use: This command is used only in embedded SQL. It determines the next step a procedural language report should take if SQL returns an error message or a record is not found. WHENEVER statements are compiled before actual run time.

Syntax: WHENEVER {SQLERROR|NOT FOUND} {CONTINUE|GOTO label}

Details: There are two kinds of WHENEVER statements, WHENEVER SQLERROR and WHENEVER NOT FOUND. You must indicate which condition you are looking for, and then what action the program should take if the condition is found. You can select CONTINUE to keep the program running, or use a GOTO command to have the program jump to a previously defined label.

Example: Whenever an SQL error is encountered, go to label "Restart"

```
WHENEVER SQLERROR GOTO RESTART
```

Whenever a record is not found, continue the program.

WHENEVER NOT FOUND CONTINUE

For related topics, see SQ:Embedding SQL.

Commenting on a Dbsystem

You can use the SQL interactive menus to place descriptive comments on various parts of your dbsystem. These comments are inserted into the system tables of the dbsystem and allow you to easily identify the different components of your dbsystem. You can comment on a column, table, index, view, synonym or external table reference.

For related topics, see SQ:System Tables and SQ:Commands:COMMENT ON.

Commenting on a Column

Each comment you place on a column is inserted into the REMARKS column for that column in the SYSCOLUMNS system table.

To comment on a column:

1. From the SQL Top Line Menu, select **A**dmin, **C**omment, **C**olumn. The Comment on Column dialog box displays.
2. Respond to the four prompts in the box:
DBSystem: Enter the name of the dbsystem that contains the table with the column on which you are commenting.
Table: Enter the name of the table that contains the column.
Column: Enter the name of the column.
Comment: Enter the text of the comment, not exceeding 255 characters in length.
3. Select **Accept**.

Figure 1 shows a completed dialog box used to comment on the NATION column of the VISITORS table in the TOURISM dbsystem.

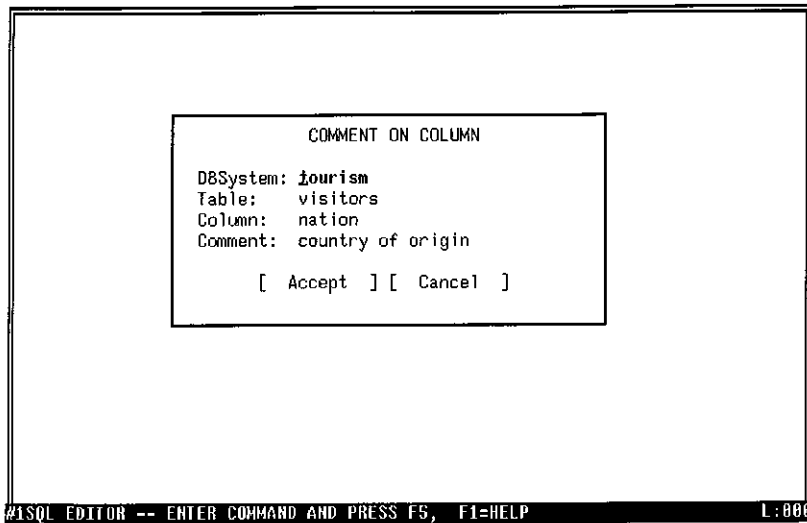


Figure 1: A completed COMMENT ON COLUMN dialog box.

Commenting on an Index

A comment placed on an index is inserted into the REMARKS column for that index in the SYSINDEX system table.

To comment on an index:

1. From the SQL Top Line Menu, select **A**dmin, **C**omment, **I**ndex. The Comment on Index dialog box displays.
2. Respond appropriately to the four prompts in the box:
DBSystem: Enter the name of the dbsystem that contains the table where the index is located.
Table: Enter the name of the table containing the index.
Index: Enter the name of the index.
Comment: Enter the actual text of your comment, not exceeding 255 characters in length.
3. Select **Accept**.

Commenting on a Table

A comment you place on a table is inserted into the REMARKS column for that table in the SYSTABLES system table.

To comment on a table:

1. From the SQL Top Line Menu, select **A**dmin, **C**omment, **T**able. The Comment on Table dialog box displays.
2. Respond appropriately to the three prompts in the box:
DBSystem: Enter the name of the dbsystem containing the table on which you are commenting.
Table: Enter the name of the table on which you are placing the comment.
Comment: Enter the text of the comment, not exceeding 255 characters in length.
3. Select **Accept**.

Commenting on a View

A comment placed on a view is placed in the REMARKS column for that view in the SYSTABLES system table.

To comment on a view:

1. From the SQL Top Line Menu, select **A**dmin, **C**omment, **V**iew. The Comment on View dialog box displays.
2. Respond appropriately to the three prompts:
DBSystem: Enter the name of the dbsystem containing the view on which you are commenting.
View: Enter the name of the view.
Comment: Enter the text of the comment, not exceeding 255 characters in length.
3. Select **Accept**.

Commenting on a Synonym

A comment placed on a synonym is placed in the REMARKS column for that synonym in the SYSTABLES system table.

To comment on a synonym:

1. From the SQL Top Line Menu, select **A**dmin, **C**omment, **S**ynonym. The Comment on Synonym dialog box is displayed.
2. Respond appropriately to the three prompts in the box:
DBSystem: Enter the name of the dbsystem that contains the synonym on which you are commenting.
Synonym: Enter the name of the synonym.
Comment: Enter the text of the comment, not exceeding 255 characters in length.
3. Select **Accept**.

Commenting on an External Table Reference

A comment placed on an external table reference is placed in the REMARKS column for that external table reference in the SYSTABLES system table.

To comment on an external table reference:

1. From the SQL Top Line Menu, select **A**dmin, **C**omment, **E**xternal. The Comment on External dialog box is displayed.
2. Respond appropriately to the three prompts in the box:
DBSystem: Enter the name of the dbsystem containing the external table reference on which you are commenting.
External: Enter the name of the external table reference.
Comment: Enter the text of the comment, not exceeding 255 characters in length.
3. Select **Accept**.

Component References and Identifiers

Enable returns an error message if an SQL statement contains an ambiguous or imprecise reference to a dbsystem component such as a table, column, index, or view.

It is possible to have two indexes with the same name, if they belong to different tables. It is also possible for you to be connected to more than one dbsystem that contains the same table name, or for two tables to have a column with the same name. In any active SQL session where a component name is not unique, you must be careful to include additional identifiers with the reference so as to pinpoint the desired component.

Your component references must also include ownership identifiers if you are querying the dbsystem of another owner, and alias identifiers if you are joining a table with itself.

Dbsystem identifiers are placed after the component name and prefixed by the @ symbol, e.g., *table_name @dbsystem*. Other identifiers are affixed to the front of the component name using a period (.), e.g., *table_name.column_name*.

Table References

If you are connected to more than one dbsystem, any reference to a table name must include the name of the dbsystem where the table is located.

For example, to select all records from the APP03 table in the RECORDS dbsystem, you would enter:

```
SELECT * FROM APP03 @RECORDS
```

When referencing a table that belongs to another user, you must prefix the table name with the table owner name. (Note that you must have privileges on the table in order to access it.)

For example, to select all records from the APP03 table that belongs to user JOHN, you would enter:

```
SELECT * FROM JOHN.APP03
```

Column References

If you are querying multiple tables and a column in one table has the same name as a column in another table, you must preface the column name with the appropriate table name. For

example, to select the CANDIDATE and DATERECD columns from the APP03 table and the DATERECD column from the APP04 table, you would enter:

```
SELECT CANDIDATE, APP03.DATERECD, APP04.DATERECD
FROM APP03, APP04
```

When joining a table with itself, you must prefix each specified column name with the corresponding table name alias. For example:

```
SELECT A1.DATERECD, A2.DATERECD FROM APP03 A1, APP03 A2
```

When you rename or comment on a column, you must prefix the column name with the table name. If the column is in a table that belongs to another user, you must also include the user name as part of the column reference. For example, to comment on the DATERECD column in the APP03 table that belongs to user JOHN, you would enter:

```
COMMENT ON JOHN.APP03.DATERECD IS "Date of receipt"
```

Index References

When renaming or commenting on an index, you must prefix the index name with the table name on which the index was created. If the table belongs to another user, you must also include the user name in the index reference. For example, to rename the REC_DATE index in the APP03 table that belongs to user JOHN, you would enter:

```
RENAME INDEX JOHN.APP03.REC_DATE TO RECEIPT
```

Connecting to a Dbsystem

The first step in using a dbsystem is to open the dbsystem. The Connect option on the SQL interactive menus is used to open a dbsystem and make it accessible to a user. A user can connect to multiple dbsystems in a session by repeatedly selecting the Connect option.

To connect to a dbsystem using the SQL interactive menus:

1. From the SQL Top Line Menu, select **F**ile, **C**onnect. The Connect dialog box displays.
2. Respond appropriately to the three prompts in the box:
 - DBSystem:** Enter the name of an existing dbsystem you wish to open. Be sure to include the appropriate file path if the dbsystem is in a directory other than the current one, or has an extension other than .DBS (SQL's default file extension).
 - User name:** Unless the dbsystem is designated as PUBLIC, you must enter a valid user name at this prompt. A user name is valid only if the user has authority to connect to the system. An authorized user of a dbsystem must be either the DBA (dbsystem administrator) or a user who has been granted CONNECT privileges by the DBA.
 - Password:** If the dbsystem requires a password, you must enter a valid password at this prompt. A valid password is either a password used by the DBA to create the dbsystem or a password belonging to a user who has been granted CONNECT privileges on the dbsystem.

3. Select **Accept**.

For related topics, see [SQ:Commands:CONNECT](#), [SQ:Disconnecting from a Dbsystem](#), and [SQ:Commands:GRANT](#) (on Dbsystem).

Create Menu

You can use SQL's Create Menu to create a dbsystem, synonym or external table reference, and to get on-line help on creating a table, index, view or schema.

You must specify a name for each component you create. For information on SQL naming conventions, see [SQ:Introduction:Naming Conventions in SQL](#).

To display the Create Menu:

1. From the SQL Top Line Menu, select **C**reate.

For related topics, see [SQ:Creating a Dbsystem](#), [SQ:Creating a Synonym](#), and [SQ:Creating an External table Reference](#).

For information on the use and syntax of SQL CREATE commands, see [SQ:Commands:CREATE EXTERNAL](#), [SQ:Commands:CREATE DBSYSTEM](#), [SQ:Commands:CREATE INDEX](#), [SQ:Commands:CREATE SCHEMA](#), [SQ:Commands:CREATE SYNONYM](#), [SQ:Commands:CREATE TABLE AS](#), [SQ:Commands:CREATE TABLE](#), and [SQ:Commands:CREATE VIEW](#).

Creating a Dbsystem

From the Query Command Screen, you can use the SQL interactive menus to create a dbsystem:

1. From the SQL Top Line Menu, select **C**reate, **D**BSystem. The Create Dbsystem dialog box displays.
2. Respond appropriately to the four prompts in the box:
 - DBSystem:** Enter the name you wish to assign to the new dbsystem. For information on SQL naming conventions, see [SQ:Introduction:Naming Conventions in SQL](#).
 - User name:** An optional user name may be entered at this prompt. By specifying a user name, you identify the owner or dbsystem administrator of the system. If a dbsystem is created with a user name, only the owner, or users given privileges by the owner, are authorized to connect to the dbsystem. If you do not enter a user name, SQL internally assigns the owner name PUBLIC, and the dbsystem becomes accessible to all users.
 - Password:** If you specified a user name at the above prompt, you may enter an optional password at this prompt. Using a password with a user name adds greater security to your dbsystem. If a dbsystem is password-protected, both the user name and the password must be specified in order to access the dbsystem.

Safety: You may set an optional data safety level at this prompt. Safety levels determine your ability to recover data in your dbsystem following a system failure. The safety level you set also affects the performance speed of your dbsystem. Enable SQL's safety levels are 0, 1, 2 and 3, where 0 is minimum safety and 3 is maximum. However, note that the higher the safety, the slower your system will operate.

If you do not specify a safety level, your dbsystem will be created with a safety of 3. On a network system, safety always defaults to 3.

WARNING: If you are working in a dbsystem set at safety level 0 or 1 and your system crashes, your dbsystem may be corrupted and data will be unrecoverable.

3. Select **Accept**.

Figure 2 shows a completed dialog box used to create a dbsystem called PRODT374.

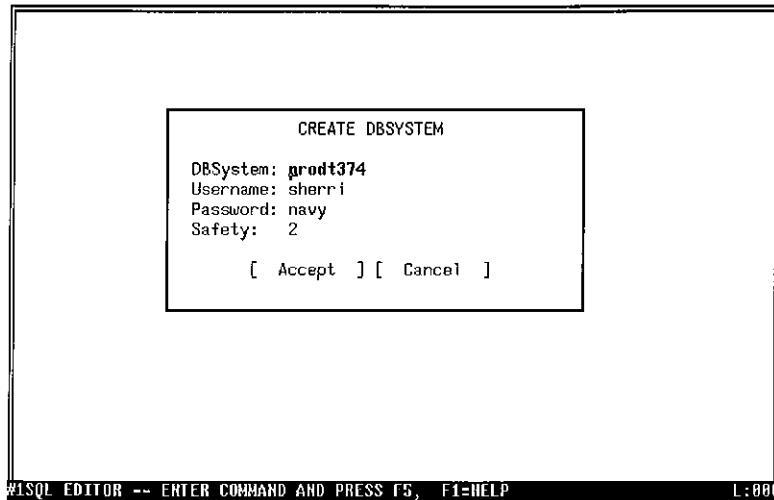


Figure 2: Create DBSystem dialog box.

For related topics, see SQ:Commands:CREATE DBSYSTEM, SQ:Safety Levels, and SQ:Commands:SET SAFETY.

Creating an External Table Reference

Using SQL's interactive menus, you can directly access data in certain non-SQL files by assigning an SQL table name to the external file. You can create external table references in SQL for the following file formats:

.DBF	Enable 2.0/dBASE II or Enable 3.0/dBASE III databases
.WK?	Lotus worksheets (read only)
.SSF	Enable spreadsheets (read only)
.WPF	Enable word processing files (read only)
.ASC	ASCII files(read only)
.DEF	Enable database definition files
.DTA	PC-File III databases
.DAT	Condor databases
.SBF	Superbase databases
.INF	Informix databases

To create a table reference for an ASCII file or an Enable word processing file, you must first create a field definitions file (.FDF) for the external data. A field definitions file defines the way in which the data will be parsed into columns and rows when it is imported into SQL. Use the DBMS module to create the .FDF file. For detailed information, refer to REF1:DB:Import.

To create an external table reference, you must be connected to the dbsystem in which the table reference will be created:

1. From the SQL Top Line Menu, select **C**reate, **E**xternal. The Create External dialog box displays.
2. Respond appropriately to the three prompts in the box:
 - DBSystem:** If you are connected to more than one dbsystem, enter the name of the dbsystem in which you are creating the external table reference. If you are connected to only one dbsystem, you may leave this prompt blank.
 - External:** Enter the name you wish to assign to the external table reference.
 - File name:** Enter the name of the non-SQL file for which you are creating the table reference. You should include the file extension, and where appropriate, the full file path, e.g. A:\COMPANY\PRODUCTS.DBF.
3. Select **A**cept.

A table reference is created for the external file specified.

For related topics, see SQ:Commands:CREATE EXTERNAL and SQ:Importing External Files.

Creating a Synonym

Synonyms are used in SQL to create alternate names for tables or views. Synonyms are particularly useful if you have privileges on another user's table. Instead of specifying the owner's name each time you reference the table, you can create another name for the table by selecting the Create Synonym option from the interactive menus. If you want the synonym to be accessible to all users who have privileges on the original table, you must append the word PUBLIC to the front of the synonym name, i.e., *PUBLIC.synonym_name*.

Before you can create a synonym for a table, you must be connected to the dbsystem where the table is located.

To create a synonym using the SQL interactive menus:

1. From the SQL Top Line Menu, select **C**reate, **S**ynonym. The Create Synonym dialog box displays.
2. Respond to the three prompts in the box:
 - DBSystem:** If you are connected to more than one dbsystem, enter the name of the dbsystem where the table for which you are creating the synonym is located. If you are connected to only one dbsystem, you may leave this prompt blank.
 - Table:** Enter the name of the table for which you are creating the synonym. If the table belongs to another user, prefix the table name with the user name. For example, if you have access to the STORAGE table belonging to user RACHEL, and you want to create a synonym for the table, you enter the table name as RACHEL.STORAGE.
 - Synonym:** Enter a name of your choice at this prompt. Prefix the name with the word PUBLIC if you are creating a public synonym, e.g. PUBLIC.ACCOUNTS.
3. Select **A**cept.

For related topics, see SQ:Commands:CREATE SYNONYM and SQ:Component References and Identifiers.

Data Constraints

In addition to the data type, columns may be defined by one or more data constraints. These data constraints further limit which values, within the defined data type, are acceptable for input into the column. Data constraints allowed in Enable's SQL are UNIQUE, NOT UNIQUE, NULL, NOT NULL, MIN, MAX, IN and USING. Data constraints are an optional part of a column definition and are placed after the data type.

Unique allows only unique values in a column and rejects duplicate data.

Not Unique allows duplication of values in a column. Unless defined as UNIQUE, columns are assumed to be NOT UNIQUE.

Not Null requires a value to be input in the column for each row of the table.

Null permits blank or null values in a column. Unless a column is defined as NOT NULL, it is assumed to be NULL.

Min value allows you to set a minimum value for the column. This value must match the data type defined for the column. Only values that equal or exceed this minimum value can be input into the column.

Max value allows you to set a maximum value for a column. This value must match the data type defined for the column. Only values that are less than or equal to the maximum value can be input into the column.

Default value allows you to set a default value for a column. This value must match the data type defined for the column. For each row of input where no value is entered for the column, the default value will be inserted internally. The DEFAULT constraint cannot be used with the UNIQUE constraint.

IN(value_list) allows you to set a list of permissible values for the column being defined. The values entered in this list must match the data type defined for the column. Only values that are included in this list can be input into the column. The IN constraint cannot be used with the MIN and MAX constraints.

Using **"format_picture"** allows you to enhance data output by setting a format picture that determines how data output from this column will be displayed. Note that the PHONE data type setting includes a format picture that controls input rather than output. (See your DBMS manual for more information on data format pictures.)

Example:

The following CREATE TABLE statement illustrates several of the data constraints described above.

```
CREATE TABLE VACATIONS
(CITY CHAR(40) NOT NULL
 IN ("VENICE", "MADRID", "MOSCOW")
 COST SMALLINT MIN 1200 MAX 3000
 AGENT CHAR(2) DEFAULT "SF"
 STARTDATE DATE UNIQUE USING "YYYY-MM-DD")
```

The CITY column is defined as a character field of no more than 40 characters. It is a NOT NULL column, which means that each row of input must contain a value for CITY. Each input value must exactly match one of the three cities given in the IN list.

The COST column is defined as a small integer field that accepts only values that are no less than 1,200 and no greater than 3,000.

The AGENT column is a character field with a maximum length of 2. For each row of input where no value is entered for this column, the character string "SF" will be automatically inserted into the column.

The STARTDATE column is defined as a date field that accepts no duplicate entries. Values input into this column can be entered in any of the four DATE formats, but all output data will be displayed in the format YYYY-MM-DD (e.g. 1990-11-29).

Data Types

When you create a table you must define one or more columns for that table. As part of the column definition, you must specify the type of data that is acceptable for that column. Once you have defined a column, only data that matches the column's data type can be input into the column. In Enable's SQL, you can choose from 17 different data types:

Char(*fixed length*) defines an alphanumeric string with a fixed number of characters. Acceptable data for this field type can consist of a combination of letters, numbers and symbols that can be less than but cannot exceed the fixed length. Each value in a CHAR column is assigned the number of spaces of the fixed length, and actual values that are less than this length are padded with blank spaces. The maximum fixed length you can define for a CHAR column is 255 characters. If no fixed length is specified, the default length is 1.

CHAR values must always be enclosed in quotation marks. Examples of values that can be input into a column defined as CHAR(19) are "Bartholomew Johnson" and "Pur10%&2\$".

Varchar(*maximum length*) defines a character string of variable length, with a maximum length setting. Values may be less than but cannot exceed the maximum length. Each value is saved with its actual size and not the size set in the length attribute (as in a CHAR column). The greatest length you can set for a VARCHAR column is 255 characters. If you do not specify a maximum length, the default length is 1. Like CHAR values, VARCHAR values must be enclosed in quotation marks.

Long or Long Varchar(*maximum length*) defines a variable length character string that exceeds 255 characters and is less than or equal to 4,000 characters. If no maximum length is set, the default length is 1. LONG VARCHAR values must be enclosed in quotation marks.

Smallint defines any whole number not greater than 32,767 and not less than -32,767. If a fractional number is entered in a SMALLINT column, SQL returns an error message. Examples of possible values in a SMALLINT column are 12 and -26000.

Integer defines any whole number not greater than 2,147,483,647 and not less than -2,147,483,647. Like SMALLINT, an INTEGER column returns an error message if fractional values are entered.

Decimal(*precision, scale*) defines fractional numbers with a fixed decimal point. You can specify a precision, which is the maximum number of digits permitted in each value of the column. The highest precision setting for a DECIMAL column is 14. If you set a precision, you can also specify an optional scale, which is the fixed number of decimal places allowed. If no precision is set, the default precision is 14. If no scale is set, the default scale is 0. Examples of values that can be entered into a column defined as DECIMAL(7,3) are 2345.014 and 577.899.

Float(*precision*) defines fractional numbers with a floating decimal point. The optional precision setting determines the maximum number of binary digits permitted in each value of the column. The highest precision you can set is 46 binary digits (14 decimal

digits). If no precision is set, the default precision is 7. Examples of values that can be input into a column defined as FLOAT(39) are *1.9890675* and *456.3210*.

Numeric(*precision, scale*) defines the same type of data as DECIMAL.

Money(*precision, scale*) defines dollar values with an optional precision and an optional scale if precision is specified. Precision can be set to a maximum of 14 digits. If no precision is specified, the default precision of a MONEY column is 14. If no scale is specified, the default scale is 2. Values in a MONEY column are displayed with a Dollar sign (\$). Examples of values you can input into a column defined as MONEY(5) are *36.1*, *4.28* and *99*.

Logical defines logical (true/false) values which must be input in one of the following forms: *True, False, Yes, No*. When selected, logical values are displayed as either *Y* (true/yes) or as *N* (false/no).

Date defines date values that can be input in four different date formats: YYYY-MM-DD, MM/DD/YYYY, DD.MM.YYYY and MON DD, YYYY. In the first three formats, the year, month and day are entered in number values. In the fourth format, 'MON' represents the first three letters of each month. Values in a DATE column are always displayed in the format MM/DD/YYYY. The minimum date you can enter in a DATE column is January 1, 1900. The maximum date is January 1, 2500. Examples of values that can be input into a DATE column are *1957-09-15* and *Jul 12, 1978* and *30.01.2043*.

Time defines time values that are entered in the following format: HH:MM[:SS] [AM|PM]. The Brackets [] are not part of the format and are used here to indicate items that are optional to the format. The Vertical bar (|) indicates a choice between two items. Military time values are acceptable for input as long as you do not include AM or PM. Examples of valid TIME values are *02:13 PM* and *10:29:34 AM* and *22:05*.

State defines a two-character value that matches one of the states in the U.S.A. A STATE column rejects any value that is not an official state abbreviation. STATE values are always cross-checked for validity with values in ZIP and PHONE columns. Examples of values that can be input into a STATE column are *OH, CA, LA* and *MA*.

Zip(*length*) defines a United States zip code that is either 5 or 9 characters in length. If you do not specify a length, the default length is 5. ZIP values are always cross-checked for validity with values in STATE and PHONE columns. Values entered in a ZIP column must be enclosed in quotation marks. Examples of values that can be input into a column defined as ZIP(9) are: *"45240"* and *"95050-3455"*.

Phone Using format picture defines telephone number values that must be entered based on a format picture you specify after USING. See your DBMS manual for more information on data format pictures. Area codes used must be valid United States area codes, since PHONE values are cross-checked for validity with values in ZIP and STATE columns. Examples of values you can input into a column defined as PHONE USING "(AAA)NNN-NNNN" are *(201)345-6100* and *(813)986-7814*.

Real defines fractional numbers with a floating decimal point and a decimal precision of 7.

Double Precision defines fractional numbers with a floating decimal point and a decimal precision of 14.

Figure 3 shows a table of the default length, value range and maximum storage size (in bytes) of each data type.

Type	Default Prec/ Length	Default Scale	Min Value	Max Value	Max Bytes
Char	1				255
Date			1/1/1900	1/1/2500	8
Decimal	14	0	-1E70	1E70	8
Double Prec	14		-1E70	1E70	8
Float	7		-1E70	1E70	8
Integer	10	0	-2147483647	2147483647	4
Logical					8
Long	1				4000
Money	14	2	-1E70	1E70	8
Numeric	14	0	-1E70	1E70	8
Phone					13
Real	7		-1E70	1E70	8
Smallint	5	0	-32767	32767	2
State	2				2
Time			01:00:00	24:59:59	8
Varchar	1				255
Zip	5				10

Figure 3: Default values and size ranges of SQL data types.

DBMS Access to SQL Data Files

Through SQL's CREATE EXTERNAL command you can read a DBMS database in SQL. Conversely, you can use DBMS interactive commands to directly access data in SQL files. Enable contains a mechanism that allows DBMS to treat an SQL table like a standard read-only database. With this feature, you can perform certain DBMS commands, such as Display or Report, on an SQL table. This function is particularly useful for printing out data from your SQL tables.

Note, however, that you cannot update an SQL table using DBMS commands. While you can find, display, sort or run a report on data in a table, you cannot perform DBMS commands that alter the data or structure of the table in any way. To update an SQL table in DBMS, you must use embedded SQL.

For related topics, see SQ:Embedding SQL and SQ:Commands:CREATE EXTERNAL.

Connecting to an SQL Dbsystem from DBMS

Before you can read an SQL table in DBMS, you must first connect to the SQL dbsystem that contains the table. From the DBMS module, you can connect to an SQL dbsystem in either of two ways:

Use embedded SQL and enter a complete SQL CONNECT statement at any DBMS Where or Fields prompt.

or

From the DBMS Top Line Menu, select SQL, Connect and respond appropriately to the prompts displayed.

A message displays in the status line indicating that the dbsystem has been connected.

Querying SQL tables in DBMS

Once you have connected to an SQL dbsystem, you can use selected DBMS commands to query the data in the dbsystem's tables.

The following DBMS commands may be performed on an SQL table: DISPLAY, QUERY, FIND, COPY, EXPORT, REPORT and SORT.

To perform a DBMS command on a table in a connected SQL dbsystem:

1. From the DBMS Top Line Menu, select the command you wish to perform.
The appropriate command screen displays.
2. At the Database prompt, enter the name of the SQL table you wish to query and add the extension .TBL. (The .TBL extension signals DBMS that an SQL table is being used as a database.)
3. Enter the appropriate responses to the other prompts on the command screen.

For example, from the DBMS module, you may want to view all data in the NUMBER, CITY and AMOUNT columns of the SQL FUNDS table where the AMOUNT column is greater than 2000. Assuming that the FUNDS table is in a currently connected SQL dbsystem, you perform the following steps:

1. From the DBMS Top Line Menu, select Display, Display.
The Display Command Screen displays.
2. At the Database prompt, type *fun ds .tbl*.
3. Leave the Index prompt blank.
4. At the Where prompt, type *amount>2000*.
5. At the Fields prompt, type *number, city, amount*.
6. Select **Accept**.
The appropriate data from the FUNDS table displays.

Limitations to Querying SQL tables in DBMS

Since SQL tables and DBMS databases are not exactly alike, please note the following limitations when reading an SQL table in DBMS.

Database Names. The maximum number of characters you can enter for a database name in DBMS is 8. If you use an SQL table name that is longer than 8 characters,

DBMS will reject it as invalid. You can overcome this limitation by renaming your SQL table or by creating a synonym or alias for the table.

Field Names. The maximum length of a DBMS field name is 10 characters. Since an SQL column is treated as a field in DBMS, any column name that is longer than 10 characters will be truncated.

Field Lengths. The maximum character width of a DBMS field is 255 characters. If your SQL table contains a column width that is greater than 255 characters (LONG VARCHAR), DBMS will truncate all data in that column to 255 characters.

Disconnecting from an SQL Dbsystem in DBMS

From the DBMS module you can disconnect from an SQL dbsystem at any time. To disconnect from a dbsystem:

1. From the DBMS Top Line Menu, select **SQL, D**isconnect.

A message appears in the status line indicating that the dbsystem has been disconnected.

Disconnecting from a Dbsystem

You can use the SQL interactive menus to disconnect from and close down a dbsystem.

To disconnect from a dbsystem:

1. From the SQL Top Line Menu, select **F**ile, **D**isconnect.
The Disconnect dialog box is displayed.
2. At the DBSystem prompt, enter the name of the dbsystem you want to close. If you are connected to only one dbsystem, you can leave this prompt blank.
3. Select **A**cept.

A message in the status line indicates that the requested dbsystem has been disconnected.

For related topics, see SQ:Commands:DISCONNECT and SQ:Commands:CONNECT.

Drop Menu

You can use SQL's Drop Menu to delete tables, views, indexes, synonyms and external table references from your dbsystem. Care should be taken when using this menu, as components that have been dropped cannot be restored.

To display the Drop Menu:

1. From the SQL Top Line Menu, select **D**rop.

For related topics, see SQ:Dropping an External Table Reference, SQ:Dropping an Index, SQ:Dropping a Synonym, SQ:Dropping a Table, SQ:Dropping a View, and SQ:Commands:DROP.

Dropping an External Table Reference

To delete an external table reference from a dbsystem, you must be connected to the dbsystem that contains the external reference.

1. From the SQL Top Line Menu, select **D**rop, **E**xternal. The Drop External dialog box displays.
2. Respond appropriately to the two prompts shown:
DBSystem: If you are connected to more than one dbsystem, enter the name of the dbsystem where the external table reference is located. Otherwise, you may leave this prompt blank.
External: Enter the name of the external table reference you wish to remove from the dbsystem.
3. Select **A**ccept.

For related topics, see SQ:Commands:DROP.

Dropping an Index

To drop an index from a table, you must first be connected to the dbsystem that contains the table.

1. From the SQL Top Line Menu, select **D**rop, **I**ndex. The Drop Index dialog box displays.
2. Respond appropriately to the three prompts:
DBSystem: If you are connected to more than one dbsystem, enter the name of the dbsystem where the index you are dropping is located. You may leave this prompt blank if you are connected to only one dbsystem.
Table: Enter the name of the table containing the index you wish to drop.
Index: Enter the name of the index.
3. Select **A**ccept.

For related topics, see SQ:Commands:DROP.

Dropping a Synonym

To remove a synonym from a dbsystem, you must be connected to the dbsystem containing the synonym.

1. From the SQL Top Line Menu, select **D**rop, **S**ynonym. The Drop Synonym dialog box displays.
2. Respond appropriately to the two prompts in the box:
DBSystem: If you are connected to more than one dbsystem, enter the name of the dbsystem containing the synonym you are dropping. If you are connected to only one dbsystem, you may leave this prompt blank.

Synonym: Enter the name of the synonym you wish to remove from the dbsystem.

3. Select **Accept**.

For related topics, see SQ:Commands:DROP.

Dropping a Table

When you drop a table from a dbsystem, any index on the table is also deleted. All virtual tables (views, synonyms and aliases) that are based on the dropped table become invalid.

To drop a table, you must be connected to the dbsystem where the table is located.

1. From the SQL Top Line Menu, select **D**rop, **T**able. The Drop Table dialog box displays.
2. Respond appropriately to the two prompts:

DBSystem: If you are connected to more than one dbsystem, enter the name of the dbsystem that contains the table you wish to drop. If you are connected to only one dbsystem, you may leave this prompt blank.

Table: Enter the name of the table you wish to drop.

3. Select **Accept**.

For related topics, see SQ:Commands:DROP and SQ:Table Types.

Dropping a View

To delete a view from a dbsystem, you must be connected to the dbsystem where the view is located.

1. From the SQL Top Line Menu, select **D**rop, **V**iew. The Drop View dialog box displays.
2. Respond to the two prompts:

DBSystem: If you are connected to more than one dbsystem, enter the name of the dbsystem where the view you wish to drop is located. Otherwise, you may leave this prompt blank.

View: Enter the name of the view to be dropped.

3. Select **Accept**.

For related topics, see SQ:Commands:DROP.

Embedding SQL

With the exception of a few commands, SQL statements can be embedded into DBMS procedural language reports or used in the DBMS module wherever procedural language commands are permitted (Where clause, post and prefield macros, etc.). You must have a thorough understanding of procedural language in order to use embedded SQL.

For related topics, see SQ:DBMS Access to SQL Data Files and SQ:SQL vs. DBMS.

Guidelines for Embedded SQL

When embedding SQL, please observe the following guidelines:

Single-line Statements. When a single-line SQL statement is embedded, begin the statement with a .SQL dot command. For example, you can open the RECORDS subsystem from a DBMS where prompt by entering the following statement:

```
.SQL CONNECT @RECORDS
```

Multi-line Statements. When a multi-line SQL statement is embedded into a procedural language report, precede the statement with the command *EXEC SQL*. This command alerts DBMS that an embedded SQL statement follows. To indicate the end of the SQL statement, place a semicolon (;) at the end of the last line, or insert the command *END-EXEC* after the last line of the statement. For example, to create an SQL table, you can embed an SQL statement in a procedural language report in either of the following ways:

```
EXEC SQL
  CREATE TABLE HOMES
    (TYPE CHAR(10) NOT NULL,
    COST MONEY NOT NULL,
    LOCATION CHAR(15) DEFAULT "SPRING LAKE")
END EXEC
```

or

```
EXEC SQL
  CREATE TABLE HOMES
    (TYPE CHAR(10) NOT NULL,
    COST MONEY NOT NULL,
    LOCATION CHAR(15) DEFAULT "SPRING LAKE");
```

Local Fields. When a local field is referenced in an SQL statement, it must be preceded by a colon (:). For example, to input the last record from the STATS cursor into the local fields COST, QUANTITY, MAKE, you would enter the following SQL statement:

```
.SQL FETCH LAST STATS INTO :COST, :QUANTITY, :MAKE
```

Null Values. When using embedded SQL to select values from an SQL table into DBMS local fields, or vice versa, be aware that DBMS does not recognize null values. In order for DBMS to process a null value, you must define an indicator field to correspond to each local field referenced in your SQL statements. This indicator, called a null indicator, receives a system-supplied value of 0 or -1 based on whether its corresponding local field contains a null value. If the local field contains a null

value, the null indicator is set to -1. If the value is not null, the null indicator is 0. Based on the value of the null indicator, you can create conditions to determine how null values will be treated. You must use null indicators if you are selecting data that may contain null values. If a null value is encountered in a local field that has no null indicator, an error message results and program execution is terminated.

Since a null indicator is a field, you must define it before you can use it. You may assign it a name of your choice, but it must have an INTEGER data type.

Null indicators are placed after their corresponding local fields in either of the following formats:

```
:local_field:null_indicator
```

or

```
:local_field INDICATOR :null_indicator
```

Replace the words *local field* and *null indicator* with the names of the local field and null indicator respectively. The word *INDICATOR* shown in the second format is a key word and is essential to the syntax.

Figure 4 is a brief report that illustrates the use of null indicators in embedded SQL. Note that the field *BAL_IND* is the null indicator for the local field *BALANCE*.

```
.define name as text 20
.define balance as decimal 7.2
.define bal_ind as integer 2
exec sql
  select name, balance
  into :name, :balance:bal_ind
  from bank where name = "Pierson"
end-exec
.if bal_ind < 0
  Unknown Balance
.else
  [balance]
.endif
```

Figure 4: Using a null indicator in embedded SQL.

Referencing Error Messages in Embedded SQL

Each one of Enable SQL's status line messages has been assigned a numeric code. Codes with negative values are used for error messages. A code of 0 means that the operation has been successfully completed. Codes with a value of 100 indicate either no record was found or end of file was encountered.

When embedding SQL statements in a procedural language report, you can use dot commands to reference the code or text of the current SQL message. Enable's DBMS treats SQL message codes and text as local database fields, called *SQLCODE* and *SQLERRTXT* respectively. Using these field names, you can reference the current SQL message code or error text in the same way you reference any other local field.

For example, if you try to retrieve a record and an error message is returned, you may want to display the text of the message:

```
.SQL FETCH PRIOR AB INTO :DATA1
.IF SQLCODE < 0
    [SQLERRTXT]
.ENDIF
```

File Menu

SQL's File Menu allows you to perform operations that affect your entire dbsystem. You can use this menu to connect to and disconnect from a dbsystem, to save and undo changes you have made in the current session, and to quit the Query Command Screen. You can also use this menu to obtain on-line help on printing out data in your dbsystem tables.

To display the File Menu:

1. From the SQL Top Line Menu, select **F**ile.

For related topics, see *SQ:Connecting to a Dbsystem*, *SQ:Disconnecting from a Dbsystem*, *SQ:Saving Your Work*, *SQ:Undoing Your Work*, and *SQ:Quitting the Query Command Screen*.

Importing External Files into SQL

You can use SQL to access certain external (non-SQL) files that have a database format. Compatible external formats are Enable's DBMS, Word Processing and Spreadsheet files, as well as ASCII, Lotus 1-2-3, PC-File III, Superbase, Informix and Condor files.

You can use SQL's CREATE EXTERNAL command to directly read the data in an external file or use the CREATE TABLE AS command to copy data from an external database into an SQL table.

If you use the CREATE EXTERNAL command to create a table reference for an ASCII or Enable word processing file, you must have already created a field definitions file (.FDF) for the data. A field definitions file defines the way in which the data will be parsed into columns and rows when it is imported into SQL. Use the DBMS module to create the .FDF file. For detailed information, see REF1:DB:Import

When importing external data into SQL, be aware of the following:

Colons. The colon (:) has restricted use in SQL and is used only to signify a local field in a procedural language report. If you import an external data file with a field name containing a colon, SQL converts the colon to an underscore (_). For example, the field name REP:TIME would be converted to REP_TIME.

Column Names. An external file imported into SQL may contain column (field) names that are SQL reserved words. Such column names will be considered invalid if referenced in an SQL statement, and SQL will not access the data in those columns. One way of viewing the data in columns with invalid names is to perform a SELECT * statement on the entire table. However, if you need to reference the specific fields, you

should create a view of the table and assign alternate names to the fields with invalid names.

For related topics, see SQ:Embedding SQL, SQ:Creating an External Table Reference, SQ:Commands:CREATE EXTERNAL, and SQ:Commands:CREATE TABLE AS.

Indexes

You can optimize performance of SQL queries by creating indexes for your tables. An index sorts and organizes data in a way that allows for faster and more efficient data retrieval.

Indexes are created on columns, and can be either single-key or multi-key. A single-key index is created on one column only and sorts rows based on that column. A multi-key index is created on two or more columns, of which the first column is the primary sort key, the second column is the secondary sort key, and so forth.

You can define an index as unique if you want to disallow duplicate data in the column(s) on which the index is created. If you attempt to input duplicate values in a column that has a unique index, SQL returns an error message. When a unique index is created on multiple columns, the combined data in those columns must be unique for each row.

The CREATE INDEX command is used to create and name an index. You can also create unique indexes when you define columns in a CREATE TABLE statement.

For related topics, see SQ:Commands:CREATE INDEX and SQ:Commands:CREATE TABLE.

Interactive SQL

As a database language, SQL can be used in two forms: *interactive* and *embedded*. To use SQL interactively, you display SQL's Query Command Screen, and either type and execute your SQL statements directly on the screen, or use the SQL interactive menus to perform simple SQL operations.

You can display the Query Command Screen in either of two ways:

From Enable's Main Menu, select Use system, Database, SQL.

or

From the DBMS Top Line Menu or the Report Command Screen, select SQL, Command Screen.

The Query Command Screen displays as a blank screen, with SQL displayed in the upper left-hand corner of the screen. Enter your SQL statements directly onto this screen or press **F10** to display the SQL Top Line Menu and use the interactive menus.

For related topics, see SQ:Embedding SQL and SQ:Query Command Screen.

Modify Menu

SQL's Modify Menu is used to display on-line help on SQL's INPUT, INSERT, DELETE and UPDATE commands.

To display the Modify Menu:

1. From the SQL Top Line Menu, select **M**odify.

For related topics, see SQ:Commands:INPUT, SQ:Commands:INSERT, SQ:Commands:DELETE, and SQ:Commands:UPDATE.

Operators

Enable SQL supports all of the following ANSI standard operators:

Arithmetic Operators: * (multiply), / (divide), - (minus), + (add).

Simple Comparison Operators: < (less than), <= (less than or equal to), > (greater than), >= (greater than or equal to), = (equal to).

Condition Operators: AND, OR, NOT, LIKE, IN, BETWEEN, EXISTS, ANY and ALL.

For more information on their usage, consult an ANSI standard SQL manual.

Printing SQL DATA

In order to print data contained in an SQL file, you must use the DBMS Report Command Screen. You can either embed SQL statements in a procedural language report or use DBMS interactive commands to run a report on an SQL data table.

For more information on accessing your SQL data files from DBMS, see SQ:DBMS Access to SQL Data Files. For more information on embedding SQL in a procedural language report, see SQ:Embedding SQL.

Query Command Screen

In interactive SQL, you use the Query Command Screen to execute SQL statements and to view the results of queries. You can type your SQL statements directly onto the screen or select the options you want from the SQL menus.

For related topics, see SQ:Interactive SQL.

Displaying the Query Command Screen

You can display the SQL Query Command Screen in either of two ways:

From Enable's Main Menu, select **U**se system, **D**atabase, **S**QL.

or

From the DBMS Top Line Menu or the Report Command Screen, select **SQL Command Screen**.

SQL's Query Command Screen displays as a blank screen, with **SQL** displayed in the upper left-hand corner of the screen. Enter your SQL statements directly onto this screen or display the SQL Top Line Menu to use the interactive menus.

Entering Statements

You can type your SQL statements directly onto the Query Command Screen. Up to 140 characters may be entered on a line, but you may find your statements easier to read if they are broken down over several lines.

In general, only one statement is entered on the screen at a time. However, you can enter more than one statement if you use a semicolon (;) to separate the statements.

Use the **Ins** key to insert characters into text that is already on the screen and the **Del** key to erase one character of text at a time. Press the **Arrow** keys to move around within text. Press **Alt/F3** to delete a line, and **F3** to insert a line.

Executing a Statement

To execute a statement after you have typed it on the command screen, press **F5**. A message in the status line at the bottom of the screen indicates that your statement is being processed.

With the exception of **SELECT** statements, after a statement is executed, the screen clears and a message in the status line indicates that processing is complete.

If SQL is unable to process your statement because of syntactical or other errors, an error message displays at the bottom of the screen, and your SQL statement remains on screen. Guided by the error message you receive, edit the statement so it is acceptable to SQL. For example, if the error message states **MISSING LEFT PAREN**, then your statement requires a left parenthesis. If you need to delete the entire statement and enter a new one, press **Alt/F3** to delete one line of text at a time.

Once you have corrected or changed the statement, press **F5** again to execute it. If your statement contains more than one type of error, you may have to revise it several times before SQL accepts it.

The Interactive Menus

As do the other Enable modules, SQL has a set of interactive menus. These menus allow you to perform some simple SQL procedures without writing SQL statements. Instead of entering the full text of an SQL statement on the Query Command Screen, you simply respond to a series of menu-generated prompts, and the program internally builds and processes the equivalent statement.

To use the interactive menus from the Query Command Screen, press **F10** to display the SQL Top Line Menu, shown in Figure 5.

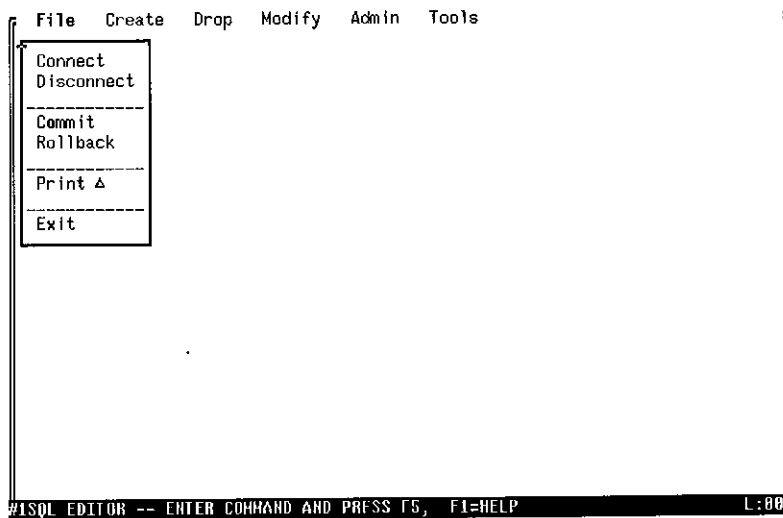


Figure 5: The SQL Top Line Menu.

Scrolling the Display of Data

When a **SELECT** statement is processed, the SQL Query Command Screen disappears and the selected records display on screen. In some cases, the data may extend beyond the screen. Press **PgUp** and **PgDn** to scroll the rows of data up and down. Press **Tab** to move the cursor to the next column on the right. Press **Shift/Tab** to move the cursor to the next column of data on the left.

To remove the display of records and return to the Query Command Screen, press **Esc** or **F5**.

Recalling a Previous Statement

You can use the Query Command Screen to review the text of statements you most recently executed in the current session. This feature is useful if you need to enter a statement similar to one you already executed. You simply redisplay the old statement and modify it. In this way, you save the time and effort that would have been spent entering the new statement, especially if the statement is long and complex.

To view the statements you entered for the current session, press **Ctrl/PgUp**. The last statement executed displays on screen. Continue pressing **Ctrl/PgUp** to view previous statements. Press **Ctrl/PgDn** to scroll through statements in the opposite direction. Once you have redisplayed a statement, you can revise it and press **F5** to execute it.

Displaying Quick Command Help

You can access on-line information on SQL's quick commands from the Query Command Screen.

To view the list of SQL function keys and keyboard commands:

1. Press **Alt/F1**.

A help box of the available quick commands displays at the top of the Query Command Screen.

The Quick Command help box remains on screen until you press **Alt/F1** again. The following is a list of the function keys and keyboard commands you can use on the Query Command Screen:

To display the above list of quick commands from the Query Command Screen, press **Alt/F1**. A function key help box displays at the top of the screen. To remove the help box, press **Alt/F1** again.

Quitting the Query Command Screen

You can quit SQL's Query Command Screen at any time. When you quit, any dbsystems that are still connected will be automatically disconnected. Exiting the interactive screen also forces an automatic COMMIT command, and all connected dbsystems will be saved to disk in their current form.

For related topics, see SQ:Commands:COMMIT and SQ:Saving Your Work.

You can quit the Query Command Screen in either of two ways:

From the SQL Top Line Menu, select **File, Exit**.

or

With the Query Command Screen blank, press **F5**.

The Query Command Screen disappears. If you entered the SQL module from the Main Menu, the Main Menu displays. If you entered SQL from the DBMS module, the DBMS Top Line Menu or the Report Command Screen displays.

Rebuilding a Dbsystem

If your computer system fails while you are working on a dbsystem set at safety level 2, the data in your dbsystem may become inaccessible. You can recreate that dbsystem by performing a Rebuild operation. The Rebuild function is also used to compress a dbsystem so it uses less disk space. Whenever you recreate a dbsystem using a Rebuild operation, you must assign a new name to the recreated system.

You cannot be connected to a dbsystem you are about to rebuild.

To rebuild a dbsystem using the SQL interactive menus:

1. From the SQL Top Line Menu, select **Admin, Rebuild**. The Rebuild dialog box displays.
2. Respond appropriately to the two prompts in the box:

DBSystem: Enter the name of the dbsystem you wish to rebuild. Include the appropriate file path where necessary.

New Name: Enter the new name you wish to assign to the recreated dbsystem.

3. Select **Accept**.

For related topics, see SQ:Commands:REBUILD and SQ:Safety Levels.

Renaming Components in a DBSystem

You can use the SQL interactive menus to change the name of any existing column, table, index, view, synonym or external table reference in your dbsystem. Once you have assigned a new name to a component, all references to that component will be updated to the new name. For example, if you rename a table, all indexes, views and synonyms that referenced the table under the old table name will continue to reference the table under its new name.

To rename any part of a dbsystem, you must first be connected to the dbsystem.

For related topics, see SQ:Commands:RENAME and SQ:Component References and Identifiers.

Renaming a Column

To rename a column:

1. From the SQL Top Line Menu, select **A**dmin, **R**ename, **C**olumn. The Rename Column dialog box displays.
2. Respond appropriately to the four prompts in the box:
 - DBSystem:** If you are connected to more than one dbsystem, enter the name of the dbsystem containing the column you wish to rename. If you are connected to only one dbsystem, you may leave this prompt blank.
 - Table:** Enter the name of the table containing the column to be renamed.
 - Column:** Enter the current name of the column.
 - New Name:** Enter the new name for the column.
3. Select **Accept**.

Renaming a Table

To rename a table:

1. From the SQL Top Line Menu, select **A**dmin, **R**ename, **T**able. The Rename Table dialog box displays.
2. Respond appropriately to the three prompts in the box:
 - DBSystem:** If you are connected to more than one dbsystem, enter the name of the dbsystem containing the table to be renamed. If you are connected to only one dbsystem, you may leave this prompt blank.
 - Table:** Enter the current name of the table.
 - New Name:** Enter the new name for the table.

3. Select **Accept**.

Renaming an Index

To rename an index:

1. From the SQL Top Line Menu, select **A**dmin, **R**ename, **I**ndex. The Rename Index dialog box displays.
2. Respond appropriately to the four prompts in the box:
DBSystem: If you are connected to more than one dbsystem, enter the name of the dbsystem containing the index you wish to rename. If you are connected to only one dbsystem, you may leave this prompt blank.
Table: Enter the name of the table containing the index to be renamed.
Index: Enter the current name of the index you are renaming.
New Name: Enter the new name for the index.

3. Select **Accept**.

Renaming a View

To rename a view:

1. From the SQL Top Line Menu, select **A**dmin, **R**ename, **V**iew. The Rename View dialog box displays.
2. Respond appropriately to the three prompts in the box:
DBSystem: If you are connected to more than one dbsystem, enter the name of the dbsystem containing the view you wish to rename. If you are connected to only one dbsystem, you may leave this prompt blank.
View: Enter the current name of the view you wish to rename.
New Name: Enter the new name you wish to assign to the view.

3. Select **Accept**.

Renaming a Synonym

To rename a synonym:

1. From the SQL Top Line Menu, select **A**dmin, **R**ename, **S**ynonym. The Rename Synonym dialog box displays.
2. Respond appropriately to the three prompts in the box:
DBSystem: If you are connected to more than one dbsystem, enter the name of the dbsystem containing the view you wish to rename. If you are connected to only one dbsystem, you may leave this prompt blank.
Synonym: Enter the current name of the synonym you wish to rename.
New Name: Enter the new name you wish to assign to the synonym.

3. Select **Accept**.

Renaming an External Table Reference

To rename an external table reference:

1. From the SQL Top Line Menu, select **A**dmin, **R**ename, **E**xternal. The Rename External dialog box is displayed.
2. Enter the appropriate responses to the three prompts in the box:
DBSystem: If you are connected to more than one dbsystem, enter the name of the dbsystem containing the view you wish to rename. If you are connected to only one dbsystem, you may leave this prompt blank.
External: Enter the current name of the external table reference you are renaming.
New Name: Enter the new name for the external table reference.
3. Select **Accept**.

Safety Levels

When you create a dbsystem, you have the option of setting a data safety level that will determine your ability to recover data after a system failure. Safety can be set at four different levels – 0, 1, 2, 3 – where 0 is minimum safety and 3 is maximum. The higher the level of safety, however, the slower your system will be.

Level 0. When safety is 0, your dbsystem is fully buffered and operates in the fastest mode. The transactions you perform are not internally logged, so the ROLLBACK command is ineffective. Since the system is fully buffered, changes you make to your dbsystem may be held in memory and not immediately committed to your disk. Your dbsystem will not be fully updated until you disconnect or exit SQL. If your computer system were to fail while you were operating at this safety level, your dbsystem might be corrupted and data would be unrecoverable.

Level 1. At this level, your dbsystem is still fully buffered, but not as fast as level 0. Your transactions are now logged internally, so ROLLBACK is possible. Like level 0, changes to your dbsystem may be held in memory, so you must disconnect or exit SQL to ensure that your dbsystem is fully updated. If your computer system failed while you were working at safety level 1, your data might be totally unrecoverable.

Level 2. Your system is only partially buffered, so it is safer than level 1 but not quite as fast. In the event of a system failure, the data in your dbsystem would still be intact, though inaccessible. You can then use SQL's REBUILD command to reconstruct your dbsystem.

Level 3. This is the slowest but safest mode of operation. At this level, your dbsystem is constantly updated, making explicit data recovery unnecessary. This is the default safety mode if no safety is specified in a CREATE DBSYSTEM statement. On a network system, safety will always be 3.

WARNING: If you are working in a dbsystem set at safety level 0 or 1 and your system crashes, your dbsystem may be corrupted and data will be unrecoverable.

You may optionally set a safety level when you create a dbssystem. If no safety level is specified, your dbssystem will be created with a safety of 3. You can set a safety level even after a dbssystem is created by using a SET SAFETY command.

For related topics, see SQ:Commands:CREATE DBSYSTEM, SQ:Commands:SET SAFETY, SQ:Commands:ROLLBACK, and SQ:Commands:REBUILD.

Saving Your Work

When you disconnect from a dbssystem, your dbssystem is saved in its existing state. You can also save your changes at any time during a session by performing a COMMIT operation. The COMMIT function saves all changes to your dbssystem that were made since you connected to the dbssystem or since the last COMMIT or ROLLBACK operation. Once you have committed your changes to disk, you cannot use a ROLLBACK command to undo them.

Note that the COMMIT operation is only valid when the AUTOCOMMIT function is turned off, since AUTOCOMMIT causes all changes to be saved automatically.

For related topics, see SQ:Commands:SET AUTOCOMMIT, SQ:Commands:COMMIT, SQ:Commands:ROLLBACK, and SQ:Undoing Your Work.

To perform a COMMIT operation using the SQL interactive menus:

1. From the SQL Top Line Menu, select **File, Commit**.
You will be prompted to verify that you want to save your changes.
2. Select **Accept**.

A message displays in the status line to indicate that your work was successfully committed.

SQL vs. DBMS – The SQL Advantage

A database is a body of organized data that can be sorted, protected, viewed and modified. Using either SQL or Enable's Database Management System (DBMS), you can create and manage such a unit. Both DBMS and SQL can be used to build and index a database. Both contain commands to insert, delete, sort, protect and retrieve selective data. In fact, there are few tasks you can do with SQL that you cannot do in DBMS. The difference between the two systems, therefore, lies not in functionality, but in application. While SQL does have some distinct advantages over DBMS, it is intended to enhance rather than to compete with DBMS. You may elect either SQL or DBMS as your primary data management tool, but you will derive more benefit through their combined use.

One of the main benefits offered by SQL is ad hoc querying. An SQL user is concerned only with the data being retrieved. In DBMS, the user must specify what data should be retrieved, as well as the process of data retrieval. One short, seemingly simple SQL statement could therefore translate into a lengthy DBMS procedural language report. This difference between the two methods is most obvious in the areas of multiple database querying and in the processing of data operations such as data grouping and aggregate sorting. In order to query multiple databases in DBMS, you must set links in place prior to retrieval. And performing

the actual queries frequently entails complex procedural language reporting. The same process can be completed in SQL with one short statement.

Another advantage SQL has over DBMS is that of advanced data security. If you are part of a network system where you share data systems with other users, you are probably concerned with the levels of data access throughout the system. SQL allows you to control user access. You can limit which users have access to a database system or to a particular table within the system. You can also determine how much privilege a user should have—to view data only, or to view and modify data.

In addition to greater data security, SQL offers a built-in safety feature for protecting your data during system failure. When you create a database system, you have a choice of four safety levels. With the highest safety level turned on, you would lose no data in the event of a power failure.

The fact that SQL has a national standard syntax proves it a useful database language to know. Expertise in SQL is easily transferred from one SQL product to another. Used alone or with DBMS, it is a powerful data management tool.

The Power of SQL: An Example

The following example demonstrates the simplicity and power of SQL.

The SPORTS database is used to store information on every athlete participating in an international sports meet. Each athlete is listed by name, nationality, age and primary sports event. The corresponding fields in the database are NAME, COUNTRY, AGE and EVENT. From this database, we need to generate a list of different countries participating in the meet, showing the average age of each country's athletes, and the number of athletes from each country. This data list must also be sorted on the column showing the average age. The resulting list should look similar to the following display:

COUNTRY	AVG. AGE	NUMBER
Norway	16	3
Japan	16	12
Australia	17	2
Canada	18	8
U.S.A.	19	15
Sweden	19	4

Using SQL to generate the desired list from the SPORTS data table, you perform the following query:

```
SELECT COUNTRY, AVG(AGE), COUNT(NAME)
FROM SPORTS
GROUP BY COUNTRY
ORDER BY 2
```

To obtain the same results using DBMS procedures, you first create an index on the COUNTRY field in the SPORTS database. Then you create the procedural language report shown in Figure 6 and run it against a dummy database.

```

.define avg_array[1000] as integer ?
.define link_array[1000] as integer ?
.define count_array[1000] as integer ?
.define num_countries as integer ?
.define temp_string as text 12
.define tot as integer ?
.define i as integer ?
.define count as integer ?
.open sports index country
.read sports first
.let num_countries = 1
.let count = 0
.let temp_string = sports.country
.let link_array[1] = sports.sys:record
.while sports.sys:record < > 0
  .if temp_string = sports.country
    .let count = count + 1
    .let tot = tot + sports.age
  .else
    .let avg_array[num_countries] = tot / count
    .let count_array[num_countries] = count
    .let num_countries = num_countries + 1
    .let temp_string = sports.country
    .let link_array[num_countries] = sports.sys:record
    .let tot = sports.age
    .let count = 1
  .endif
  .read sports next
.endif
.let avg_array[num_countries] = tot / count
.let count_array[num_countries] = count
.let count = 1
.while count > 0
  .let count = 0
  .let i = 1
  .while i < num_countries
    .if (avg_array[i]) > (avg_array[i + 1])
      .let temp_string = @string(avg_array[i + 1])
      .let avg_array[i + 1] = avg_array[i]
      .let avg_array[i] = @num(temp_string)
      .let temp_string = @string(link_array[i + 1])
      .let link_array[i + 1] = link_array[i]
      .let link_array[i] = @num(temp_string)
      .let temp_string = @string(count_array[i + 1])
      .let count_array[i + 1] = count_array[i]
      .let count_array[i] = @num(temp_string)
      .let count = count + 1
    .endif
    .let i = i + 1
  .endif
.endif
.reformat off
.clrscreen
Country Average Age Number
.let count = 1
.while count < = num_countries
  .read sports first sys:record = link_array[count]
(sports.country{12}) [avg_array[count]{3}] [count_array[count]{7}]
  .let count = count + 1
.endif

```

Figure 6: The DBMS version of a simple SQL query.

Statistical Functions

Enable's SQL supports the five ANSI standard aggregate functions: AVG, COUNT, MAX, MIN and SUM. (For more information on these functions, refer to documentation of ANSI standard SQL.) Enable's SQL also provides functions for computing standard deviation and variance.

Standard Deviation

The syntax for calculating the standard deviation of a column is: `STD(column_name)`. For general information on this function, see REF2:IN:Functions.

Variance

The syntax for calculating the variance of the values in a column is: `VAR(column_name)`. For general information on this function, see REF2:IN:Functions.

Structure of an SQL File

Dbsystem

Each Enable SQL file is an entire database system, which we refer to as a *dbsystem* (pronounced dee-bee-system). A dbsystem is a collection of data groups that generally contain related information. For example, a company might use a dbsystem to store information on every aspect of the company. For better data management, this information would be divided into data groups such as personnel information, payroll, expenses, sales, inventory and customers. Even though each group stores data on a different subject, the groups are said to be relational because they share a common element—they pertain to the same company.

Table

In a dbsystem, each data group is called a *table*. (The data group that contains data on all employee records might be the PERSONNEL table, and the data group with information on salaries and tax withholdings could aptly be called the PAYROLL table.) An SQL table is functionally equivalent to an Enable DBMS database. In addition to actual data storage, tables contain information on field definitions, indexes and data security. Each table resembles a mathematical matrix and is organized into rows and columns.

Row

A *row* is the same as a database record and contains a set of information on one case. For example, one row in the PERSONNEL table might contain the address, telephone number, social security number and name of a particular employee.

Column

While a row is a horizontal cross section of a table, a *column* designates a vertical block of a table. A column stores a set of similar data in a table, all of which must be of the same type. For example, one column in the PERSONNEL table might be the PHONE column. This

column would contain the telephone number of each employee in the company. Another column, the STREET column, would contain the street addresses for all the employees of the company.

Figure 7 illustrates the relationship between columns and rows in the PERSONNEL table.

The PHONE column

NAME	S.S. #	PHONE	STREET
Barber R.	890-23-9433	489-3300	14 Lake Drive
Flem A.	586-40-8917	326-5514	233 Monty Avenue
Jones K.	673-55-3041	280-8756	54 Driftwood Lane
Shield S.	901-68-5212	326-3148	197 Kent Street
Watnick P.	455-03-7362	324-8620	7 Willow Street

One Row

Figure 7: The PERSONNEL table.

System Tables

When you create an SQL dbsystem, the system automatically creates an accompanying data dictionary. This data dictionary is used by the system to store information about the structure and components of your dbsystem. The SQL data dictionary is actually a collection of tables, commonly referred to as the *system tables*.

The system tables in Enable's SQL are SYSTABLES, SYSCOLUMNS, SYSPASSWD, SYSTABAUTH, SYSTRANS and SYSINDX. You can query system tables as you would query other SQL tables. However, you cannot perform INSERT, ALTER, DROP, UPDATE or INPUT statements on any of these tables. System tables update automatically as you modify the structure of other tables in your dbsystem.

You cannot grant or revoke privileges on a system table. Any user who is able to connect to a dbsystem can query its system tables.

Systables

The SYSTABLES table lists detailed information on every table in the current dbsystem, including the system tables. This table consists of 12 columns.

The CREATOR column lists the name of the user who created the table. For each system table, the creator is listed as SYSTEM. For tables created for public access, the creator name is listed as PUBLIC.

The TNAME column stores the name of each table in the dbsystem.

The TABLETYPE column is an integer field that shows the table type of each table. A table type of 1 describes a user-created base table (created with a CREATE TABLE statement). Type 3 describes a system table. Other table types are 256, 512 and 1024 for synonyms, views and external table references respectively.

The **NCOLS** column shows the actual number of columns in each table.

The **HSTAMP** column contains the number of the header stamp assigned to a table whenever the table's structure is changed. The information in this column is mainly for internal use, but by comparing values in this column you can determine what tables were changed most recently. The higher the header stamp, the more recently the table's structure was changed.

The **CREATE_DATE** column stores the date the table was created.

The **CREATE_TIME** column stores the time the table was created.

The **HCOL** column stores the number of columns ever created for the table. If columns are added to the table, this number will increase by the number of columns added. Dropping columns, however, does not affect the value in the HCOL column.

The **HROW** column lists the number of rows that were ever input or inserted into the table. Adding new rows to the table causes this number to increase. Dropping rows does not affect the HROW value.

The **T_FID** column stores the internal file identification number for each system table and base table. File identification numbers are assigned to base tables, system tables and indexes.

The **REMARKS** column contains descriptive comments you make on tables using the **COMMENT ON** command.

The **VTEXT** column is relevant only to views and external table references. For each view listed, the VTEXT column lists the clause used to define the contents of the view when the view was created. For each external table reference, the external file name is given.

Figure 8 is a screen display of data retrieved from a dbssystem's SYSTABLES by performing the statement: **SELECT * FROM SYSTABLES**. Due to screen limitations, only the first four columns of the table are shown. The other columns are viewed by pressing **Tab** to cursor to the right. Note that the first six tables listed are always the system tables. The **CHECKING** and **INVENTORY** tables were created by the user **SHARON**.

CREATOR	TNAME	TABLETYPE	NCOLS
SYSTEM	SYSTABLES	3	12
SYSTEM	SYSCOLUMNS	3	18
SYSTEM	SYSPASSWD	3	4
SYSTEM	SYSTABAUTH	3	16
SYSTEM	SYSTRANS	3	14
SYSTEM	SYSINDEX	3	10
SHARON	CHECKING	1	3
SHARON	INVENTORY	1	4

#1SQL 1 of 8 L:000

Figure 8: Displaying data in the SYSTABLES table.

Syscolumns

The SYSCOLUMNS table stores information on every column of every table in the dbsystem. This table has 18 columns.

The CREATOR column stores the name of the user who created the column. The creator of each column in a system table is listed as SYSTEM. The creator of each column in a public access table is listed as PUBLIC.

The TNAME column stores the name of the table on which the column was created.

The CNAME column lists the name of the column.

The T_FID column lists the internal file identification number that the system assigned to each system table and base table.

The COLTYPE column contains an integer that corresponds to the column's data type as follows: 1 = CHAR; 3 = PHONE; 4 = STATE; 5 = ZIP; 6 = VARCHAR; 8 = LONG VARCHAR; 10 = SMALLINT; 11 = INTEGER; 12 = NUMERIC; 13 = DECIMAL; 14 = FLOAT; 15 = REAL; 16 = DOUBLE PRECISION; 17 = MONEY; 18 = DATE; 19 = TIME; 20 = LOGICAL.

The NULLS column is a logical field that shows whether a column has been defined as NULL. The value in the NULLS column is either "T" (True) or "F" (False).

The UNQ column is also a logical field, and shows whether a column has been defined as UNIQUE.

The **LENGTH** column contains the maximum character length allowed in the column.

The **PREC** column contains the maximum precision allowed in each numeric column.

The **SCALE** column stores the scale defined for each numeric column.

The **COLNO** column shows the numerical order of columns as you have set them up in the table. Changes you make to the table structure using the ALTER command with MOVE, DROP or ADD are reflected in the way columns are numbered in the COLNO column. This is the order in which columns will display when you perform a SELECT * statement on the table.

The **A_COLNO** column contains the numerical order in which each column was created. Once a number has been assigned to a column, this internal number is never reassigned, even if the column is deleted or is moved to a new column position.

The **MIN_VALUE** column shows the minimum value of a column if the column was defined as having a minimum value.

The **MAX_VALUE** column shows the maximum value of a column if the column was defined as having a maximum value.

The **DEFAULT_VALUE** column shows the default value of a column if the column was defined as having a default value.

The **USING_PIC** column shows the format picture used for a column if the column was defined as having a format picture.

The **REMARKS** column contains any comments you made on a column using the COMMENT ON command.

The **OK_VALUES** column shows the values that are acceptable for input into each column that was defined with an IN list.

Syspasswd

The SYSPASSWD table contains information on all privileges that were granted on the dbsystem with GRANT (on dbsystem) statements. This table consists of 4 columns.

The **UNAME** column lists the name of each user who has been granted dbsystem privileges.

The **PRIV** column is an integer field that shows the type of privilege granted to each user. Type 1 is CONNECT, 2 is RESOURCE and type 3 is DBA.

The **ACTIVE** column stores data for internal use only.

The **EXP_DATE** column stores data for internal use only.

Systabauth

The SYSTABAUTH table stores information on privileges that were granted to users on the different tables in the dbsystem. Each row in the table is equivalent to a GRANT (on table) statement. This table has 15 columns.

The **GRANTOR** column lists the name of the user who performed the GRANT statement.

The **GRANTEE** column stores the name of the user who was awarded the privileges in the GRANT statement.

The **CREATOR** column stores the name of the owner or creator of the table on which the privileges were granted.

The **TNAME** column contains the name of the table on which the privileges were granted.

The **CNAME** column stores the names of any particular columns in the table on which privileges were awarded.

The **T_FID** column stores the file identification number of each base table. This data is for internal use only.

The **GRANT_DATE** column lists the date on which table privileges were granted. Note that this date is always taken from your system's internal clock. If your system date is not correct, the data in the GRANT_DATE column will be wrong.

The **GRANT_TIME** column shows the time at which each GRANT statement was executed. As with the GRANT_DATE column, the data for this column is taken from your system's internal clock and will not be correct if the system time is inaccurate.

The **ALT** column indicates whether the ALTER privilege was granted. If the privilege was awarded, a Y is listed; if the privilege was awarded with a GRANT OPTION clause, a G is listed. If the column is blank, the privilege was not granted.

The **DEL** column indicates whether the DELETE privilege was granted. This column is interpreted in the same way as the ALT column.

The **INX** column indicates whether the INDEX privilege was granted. This column is interpreted in the same way as the ALT column.

The **SEL** column indicates whether the SELECT privilege was granted. This column is interpreted in the same way as the ALT column.

The **UPD** column indicates whether the UPDATE privilege was granted. This column is interpreted in the same way as the ALT column.

The **ACTIVE** column stores data for internal use only.

The **EXP_DATE** column stores data for internal use only.

Sysindx

The SYSINDEX table stores information on each index that was created in each table in the dbsystem. This table has 10 columns.

The **INDEX_NAME** column lists the name of each index in the dbsystem.

The **INDEX_CREATOR** column lists the name of the user who created the index.

The **CREATOR** column lists the name of the owner or creator of the table that contains the index.

The **TNAME** column stores the name of the table that contains the index.

The **T_FID** column contains the internal file identification number for each base table.

The **I_FID** column contains the internal file identification number for each index.

The **UNQ** column is a logical field that indicates whether the index is unique. A value of T signifies a unique index; a value of F signifies that the index is not unique.

The **COL_COUNT** column is an integer field that shows the number of columns on which the index was created.

The **COL_LIST** column lists the name(s) of the column(s) on which the index was created.

The **REMARKS** column shows any comments you made on an index using a COMMENT ON command.

Systrans

The SYSTRANS table stores data on the number of different transactions executed on the base tables during the current session. This table consists of 11 columns.

The **TID** column stores data for internal use only.

The **CONNECT_NAME** column lists the name of each connected user who has performed a transaction on a table.

The **CREATOR** column lists the name of the owner or creator of the table on which the transaction was performed.

The **TNAME** column shows the name of the table on which the transaction was performed.

The **BASE_TABLE_FID** column stores the internal file identification number of each base table.

The **SYSTAB_RECNO** column lists the system record number of each table, which is the internal order number assigned to each table when it is created. The system record number of each table in the dbsystem is stored in the SYS:RECORD column of the SYSTABLES table, and can be viewed with the statement: SELECT ** FROM SYSTABLES.

The **ROLLBACK_FID** column stores data for internal use.

The **MU_INSERT_FID** column stores data for internal use.

The **SU_HROW** column stores data for internal use.

The **CMND_ROLLBACK_FID** column stores data for internal use.

The **CMND_MU_INSERT_FID** column stores data for internal use.

The **STATUS** column stores data for internal use.

The **T_INDX** column stores data for internal use.

The **SESSION_ID** column stores data for internal use.

Table Types

SQL recognizes two kinds of table. A *base table* stores actual data, and a *virtual table* is an alternate means of accessing data in a base table. When we specify a table, we are generally referring to both types of tables.

All tables created with a CREATE TABLE statement are base tables. System tables are considered to be restricted base tables, since you cannot directly change the structure or data in these tables.

Views, synonyms, aliases and external table references are all virtual tables. They allow you to reference data in different ways. A view lets you assign a name to a select set of data identified by a SELECT statement performed on one or more tables. Aliases and synonyms simply provide alternate names for base tables, views, synonyms and external table references. An external table reference allows you to access data in a data file that was created outside of SQL.

For related topics, see SQ:Commands:CREATE TABLE and SQ:System Tables.

Undoing Your Work

You can undo changes you have made to a dbsystem by performing a ROLLBACK operation. The ROLLBACK function reverses all changes made to your dbsystem during the current session since work was last saved, i.e. since you connected to the dbsystem or since the last COMMIT operation.

Note that the ROLLBACK operation is not valid when the AUTOCOMMIT function is turned on, as all changes are saved automatically.

For related topics, see SQ:Commands:ROLLBACK, SQ:Commands:COMMIT, SQ:Commands:SET AUTOCOMMIT, and SQ:Saving Your Work.

You can use the SQL interactive menus to perform a ROLLBACK operation:

1. From the SQL Top Line Menu, select **F**ile, **R**ollback.
You will be prompted to verify that you wish to discard your changes.
2. Select **A**ccept to complete the operation.

Once the operation is complete, a message in the status line indicates that the ROLLBACK was successfully executed.

Quick Commands

In addition to using the Top Line Menu to accomplish your word processing tasks, Enable also provides an extensive range of quick commands. Reviewing these commands, and noting which of them are most applicable to the type of tasks you perform regularly, will facilitate your work.

The quick commands available in SQL are as follows:

Function	Keystroke Combination
Display SQL quick commands on screen	Alt/F1
Move cursor to last character of current text line	F2
Insert blank line into text	F3
Delete current line of text	Alt/F3
Execute SQL statement	F5
Exit from blank Query Command Screen	F5
Size, move or close the Query Command Screen	F9
Display SQL Top Line Menu	F10
Scroll backward through recently executed statements	Ctrl/PgUp
Scroll forward through recently executed statements	Ctrl/PgDn
Open another window	Alt/Home

SQ:68

Enable
