NISTIR 5998

User's Guide for the SQL Test Suite, Version 6.0

David Flater
Leonard Gallagher
Shirley Hurwitz
Joan Sullivan

U.S. DEPARTMENT OF COMMERCE
Technology Administration
National Institute of Standards
and Technology
Gaithersburg, MD 20899-0001

NIST

NISTIR 5998

User's Guide for the SQL Test Suite, Version 6.0

David Flater
Leonard Gallagher
Shirley Hurwitz
Joan Sullivan

U.S. DEPARTMENT OF COMMERCE
Technology Administration
National Institute of Standards
and Technology
Gaithersburg, MD 20899-0001

December 1996

U.S. DEPARTMENT OF COMMERCE
Michael Kantor, Secretary

TECHNOLOGY ADMINISTRATION
Mary L. Good, Under Secretary for Technology

NATIONAL INSTITUTE OF STANDARDS
AND TECHNOLOGY
Robert E. Hebner, Acting Director

User's Guide for the SQL Test Suite, Version 6.0

ABSTRACT: This manual describes the SQL Test Suite (Version 6.0) and the procedures needed to test and evaluate an SQL implementation through a standard programming language interface or through Interactive Direct SQL. This release of the SQL Test Suite was completed December 31, 1996, and includes new tests to validate the Intermediate level of conformance to 1992 SQL standards. The SQL Test Suite, Version 6.0, was developed jointly by the U.S. National Institute of Standards and Technology (NIST), National Computing Centre Limited (NCC) in the U.K, and Computer Logic R&D in Greece. The first five versions of the NIST SQL Test Suite were produced by NIST over the years 1987 through 1995. The donation of tests by NCC and Computer Logic, under the CTS5 SQL2 Project sponsored by the European Community, has been a major contribution to the current version.

The SQL Test Suite may be used to evaluate conformance to the following SQL standards specifications: ISO/IEC 9075:1992, ANSI X3.135-1992, FIPS 127-2, and X/Open XPG4 SQL. The test suite contains tests and procedures to evaluate conformance to various levels of the standards or profiles: Intermediate SQL, Transitional SQL, Entry SQL, sizing profiles, flagging of extensions, X/Open profiles. The test suite consists of schemas and test programs for Interactive SQL as well as ten different programming language test suite types: Embedded C, Embedded COBOL, Embedded Fortran, Embedded Ada, Embedded Pascal, Module Language C, Module Language COBOL, Module Language Fortran, Module Language Ada, and Module Language Pascal.

The SQL Test Suite is used to validate commercial SQL products for conformance to ISO, ANSI, and FIPS SQL standards. The results of the validation service are listed in an online Validated Products List. The software for the SQL Test Suite can be downloaded from the Web pages of the NIST Software Diagnostics and Conformance Testing Division. To download this conformance testing software, go to:

http://www.itl.nist.gov/div897/ctg/software.htm#pubsoft and select SQL

KEY WORDS: conformance testing; database standards; interoperability; SQL; testing of software; user guide; Validated Products List; validation of software.

DISCLAIMER: Because of the nature of this report, it is necessary to mention vendors and commercial products. The presence or absence of a particular trade name product does not imply criticism or endorsement by the National Institute of Standards and Technology, nor does it imply that the products identified are necessarily the best available.

TABLE OF CONTENTS

APPENDICES

1. INTRODUCTION

This manual describes the SQL Test Suite (Version 6.0) and the procedures needed to test and evaluate an SQL implementation through a standard programming language interface or through Interactive Direct SQL.

The SQL Test Suite may be used to evaluate conformance to the following SQL standards specifications:

FIPS 127-2
ANSI X3.135-1992
ISO/IEC 9075:1992
X/Open XPG4 SQL

Goals of SQL Test Suite

This test suite was originally developed by the Information Technology Laboratory of the National Institute of Standards and Technology in support of its federally mandated program of Federal Information Processing Standards (FIPS). The purpose of this test suite is to help evaluate conformance of SQL implementations to mandatory requirements of FIPS PUB 127-2. This is the FIPS Publication that adopts ANSI X3.135-1992, a voluntary industry standard for database language SQL, for use by the federal government. FIPS PUB 127-2 became effective on December 3, 1993, six months after its publication in the *Federal Register*.

FIPS PUB 127-2 supersedes previous FIPS PUBs 127 and 127-1. The original FIPS PUB 127 required SQL in relational DBMS applications acquired or developed after August 3, 1988. FIPS PUB 127-1 offered new conformance alternatives, new programming language interfaces, a new integrity enhancement option, clarification and correction of existing specifications, and additional considerations for use in procurements. FIPS 127-2 provides a substantial, upward-compatible enhancement of Database Language SQL. It includes four levels of conformance: Entry SQL, Transitional SQL, Intermediate SQL, and Full SQL. Entry SQL is a minor enhancement over the requirements of FIPS PUB 127-1. Version 6.0 of the SQL Test Suite contains tests for Entry SQL, Transitional SQL, and Intermediate SQL.

An important objective of FIPS PUB 127-2 is:

> **To reduce overall software costs by making it easier and less expensive to maintain database definitions and database application programs and to transfer these definitions and programs among different computers and database management systems, including replacement database management systems.**

The programs in this test suite can be viewed as straightforward standard application programs that a user wishes to transfer from one standard environment to another standard environment. Is this

goal achievable on the implementation being tested? Or, is substantial analysis and modification required before these programs will execute correctly?

The process of installing and running the programs in this test suite is to be documented by the testers according to instructions provided in this manual. That documentation, along with the automated summary report of pass/fail results for individual SQL features, can be used to help evaluate conformance to FIPS PUB 127-2.

Since April 1990, NIST has offered a formal testing service. This service issues Certificates of Validation for tested products passing **all** required tests. A Validation Summary Report is issued for all implementations tested. This validation report documents, to the extent tested, the implementation's conformance to FIPS PUB 127-2. Beginning July 1, 1997, NIST will turn over SQL testing services to the private sector. Federal agencies should continue to require validated products in their SQL procurements, since validation services will continue to be available and since standard-conforming SQL products are critically important in heterogeneous distributed database environments.

The test suite can also be used to evaluate the adherence to X/Open XPG4 SQL specifications (with optional Integrity Enhancement Feature) for X/Open Branding Requirements; it covers the language bindings Embedded SQL C and COBOL only.

NIST publishes an on-line register, *Validated Products List*, showing SQL implementations that hold current Certificates of Validation and registered Validation Summary Reports. This publication also lists results of testing for the programming languages (Ada, C, COBOL, and FORTRAN) as well as Graphics, POSIX, and Security standards. As NIST transitions its testing services to the private sector, NIST World Wide Web pages will point to a directory of conformance testing programs, products, and services to provide additional information.

History of SQL Test Suite

The SQL Test Suite was first made available to the public in August 1988 as the NIST SQL Test Suite, Version 1.1. This version included tests for three programming languages: COBOL, FORTRAN, and C. Version 1.1 included tests for Embedded SQL as well as Module Language SQL. In May 1989 the test suite was enlarged and released as Version 1.2. This version included tests for additional SQL features, as well as tests for Embedded SQL Pascal and a Pascal interface to Module Language SQL. The NIST SQL Test Suite was distributed, for a fee, under the terms of a software agreement. Version 2.0 of the SQL Test Suite contained additional tests as well as the support system (software utilities) to administer the validation process.

Continuing standardization work for SQL resulted in a revised SQL standard, ANSI X3.135-1989, published December 1989. This revised standard contained integrity enhancements for SQL, including referential integrity, default values for columns, and check clauses. FIPS PUB 127-1 was revised to specify these new integrity features as an optional module which federal agencies could

either require or (by default) not require in a procurement. Version 2.0 of the test suite also contained a set of tests to validate conformance to this optional module.

In the same time frame, ANSI X3.168-1989 standardized the embedding of SQL in programming languages (Ada, C, COBOL, FORTRAN, Pascal and PL/I). The first release of the SQL Test Suite contained tests for Embedded SQL, in anticipation of this standard. Since numerous implementations of Embedded SQL already existed, prior to standardization, NIST hoped that the early availability of tests for that interface would hasten the conformance of implementations to the revised FIPS PUB 127-1. Version 3.0 provided test suites for Ada bindings to SQL and also tests for the errata in the SQL Information Bulletin SQLIB-1.

ANSI X3.135-1992, the 1992 revision of the SQL standard, represents a major enhancement in SQL functionality. Conformance to FIPS PUB 127-2, Entry SQL, requires additional capabilities from an SQL implementation beyond those required for minimal conformance to FIPS PUB 127-1. The Integrity Enhancement Feature is now mandatory. Support for the following additional features is now required: SQLSTATE status codes, delimited identifiers, renaming columns, commas in parameter lists, and SQL Errata against ANSI X3.135-1989 (approved after publication of SQLIB-1). Version 4.0 of the SQL Test Suite provides tests for all the features in Entry SQL. Although MUMPS is one of the standard programming language interfaces specified in FIPS 127-2, the SQL Test Suite does not yet have programs to validate the MUMPS interfaces to SQL.

Version 2.0 was used in the formal testing service offered by NIST which opened in April 1990. Version 3.0 became the official version of the test suite in July 1992, and Version 4.0 became the official version in January 1994.

In Version 5.0, tests were included to address features of Transitional SQL features defined in FIPS 127-2, as well as features of the X/Open CAE Specification *Structured Query Language (SQL).* Version 6.0 grew substantially through the donation of new tests for Intermediate SQL written by European collaborators, National Computing Centre in the U.K. and Computer Logic R&D in Greece. These tests were developed under the Conformance Testing Service Project for SQL-92 called CTS5 SQL2, sponsored by the European Community (EC). Version 6.0 became freely accessible over the World Wide Web on December 31, 1996.

Description of SQL Test Suite

The SQL Test Suite provides ten programming language test suite types: Embedded (preprocessor) SQL Ada, Embedded SQL C, Embedded SQL COBOL, Embedded SQL FORTRAN, Embedded SQL Pascal, Module Language Ada, Module Language C, Module Language COBOL, Module Language FORTRAN, and Module Language Pascal. The test suite also provides an Interactive Direct SQL test suite type to test interactive invocation of SQL statements as defined in FIPS 127-2. In Versions 5.0 through 6.0, new tests for Transitional and Intermediate SQL were not translated into three interfaces: Module Language FORTRAN, Embedded, and Module Language Pascal. These

three interfaces have been the least popular, and generating additional tests for these interfaces is not cost effective.

The original test programs were developed in Embedded (preprocessor) SQL for the C language. The design objective for the test programs was to provide a simple test for every general rule in the standard and to cover fully all SQL syntax.

Ada, COBOL, FORTRAN, and Pascal test routines, as well as module language test routines, were generated by software (written by NIST) from the original Embedded SQL C language. For this reason, the style of the translated code may seem unnatural for a given language. The original Embedded SQL C Language tests are very simple, using only a carefully restricted subset of the C Language. Otherwise, it would be technically infeasible to translate these tests into the other programming languages.

The Interactive Direct SQL test files were created by extracting SQL statements from the Embedded SQL C programs. Test cases were reworked to avoid reference to cursors and host variables. The resulting text files were annotated with comments describing the test and the expected results required for a "pass."

Each test is designed to be short and simple, exercising as little of the host language as possible. The host language compiler should be validated separately to ensure that it conforms to the applicable standards. The use of complex host language code in SQL conformance programs would make tests difficult to understand and would make it more difficult to resolve questions of interpretation of the SQL standard.

Many of the tests involve 3 small tables containing a total of 23 rows. The data types of columns in these tables are either character string or integer, so the tests will work across all these programming languages. Other tables are used to test approximate numeric and scaled exact numeric data types. Additional tests have been written to cover the data type variables specific to each language.

Each program contains one or more tests. Although allowing only one test per program would simplify the evaluation of implementations with a high degree of nonconformity, it would impose additional overhead on implementations with a high degree of conformity. The tests within a program are intended to be independent so any one test may be removed without affecting the remaining tests.

Each test is self-evaluating; i.e., each test is written with knowledge of the data in the database and the correct response for a specific SQL statement. Each test checks for correct execution of the SQL statement and then inserts into the reporting table, TESTREPORT, a "pass" or "fail" value for that test. After all the test programs have executed, a summary of test results is produced automatically by another program which reads TESTREPORT.

As each test is executed, a description of the test is printed on standard output (the screen) along with appropriate data values and the test result. This output should be considered as a "log" of the test programs. It is intended to assist in debugging and in analyzing nonconformities. This output is not needed to produce the automated conformance analysis of the SQL test suite.

These tests are not designed to debug DBMS software; however, they may help identify problem areas. The use of small tables does not challenge the buffer-management strategy of an implementation. In addition, the frequent use of ROLLBACK (after tests which modify tables), to restore the base data to its original state (and thus simplify testing), limits testing of the COMMIT path. Since the SQL standard does not address physical database design, it is likely that schema definition and DML tests will be run in the simplest manner possible, without optimization.

The test suite includes a few tests for the "SQL Flagger" option specified in Section 10.d of FIPS PUB 127-2. These tests contain extensions to the SQL standard. In general, if an SQL implementation supports these extensions, it must be able to flag the extensions with warning messages. These tests are to be run with the flagging turned off and then, if successful, rerun with the flagging turned on. Test evaluation for the SQL Flagger is subjective, based upon examination of any warnings which are printed (or displayed on the screen) when extensions to SQL are used. The "SQL Flagger" tests are very limited. They are intended to demonstrate the existence and style of monitoring provided by a vendor. They do not systematically attempt to detect SQL extensions which are not flagged. For Entry SQL, standard features which are required only by higher levels (beyond Entry) should all be flagged along with nonstandard features. It is desirable, but not required (until Intermediate SQL), that the flagging message indicate the exact status (Transitional SQL, Intermediate SQL, Full SQL, nonstandard extension) of the flagged feature.

The test suite has a set of programs to test the specifications in FIPS PUB 127-2, Section 16.6, "Sizing for database constructs." These minimum specifications for the precision, size, or number of occurrences of database constructs are contained (by default) in procurements which do not provide alternate specifications. Reporting of the FIPS sizing tests is separate from reporting on other tests. FIPS sizing tests are not technically considered conformance tests, and passing these tests is not required for a Certificate of Validation for FIPS 127-2.

The test suite includes a set of programs to test features from the X/Open CAE Specification *Structured Query Language (SQL)*, Document Number C201, which contain some extensions to the ISO/IEC 9075:1992 standard.

Utility programs are included to make global and program-specific changes in a controlled and systematic manner and to document those changes in the automated report.

Unless stated otherwise, all references to sections, syntax rules or general rules in this documentation are to ANSI X3.135-1992 (or equivalently ISO/IEC 9075:1992).

2. TECHNICAL GOAL OF SQL TEST SUITE

The technical goal of the test suite is to help evaluate an SQL implementation's conformance to various levels of the SQL standard, as specified in ANSI X3.135-1992 (or equivalently ISO/IEC 9075:1992), through one or more standard programming language interfaces.

The test suite contains additional tests to help evaluate conformance to: (1) the minimum sizing parameters for database constructs specified in FIPS PUB 127-2, Section 16.6, (2) the flagging of extensions, specified in FIPS PUB 127-2, Section 10.d, SQL Flagger, (3) Interactive Direct SQL, as specified in FIPS PUB 127-2 Section 16.5, and (4) X/Open Extensions for features specified in the X/Open CAE Specification.

The test suite contains ten different programming language test suite types. An SQL implementation claiming conformance to FIPS PUB 127-2 for a particular SQL interface; for example, Embedded SQL COBOL, should be tested with the appropriate test suite type. The programming language compiler used for testing should conform to the FIPS standard for that language and should be listed in the *Validated Products List*, which is published on the NIST Web Server URL address ftp://speckle.ncsl.nist.gov/vpl/sqlintro.htm.

The intention of NIST is that this test suite should be used to help evaluate compliance of implementations of SQL to FIPS PUB 127-2. A correct implementation of FIPS 127-2 requires the incorporation of the SQL standard document, ANSI X3.135-1992 (or ISO/IEC 9075:1992), into the design specifications for the SQL implementation. The SQL test suite then confirms that the standard has been interpreted and implemented correctly by the SQL supplier. The test suite is intended to be used in conjunction with the SQL supplier's own independently-developed regression tests to ensure a robust and internally consistent product. A quality SQL implementation is not achievable by simply "fixing the product" until it passes the SQL Test Suite.

It is important to recognize the limitations of this test suite and of any test suite. In particular, it would be incorrect for implementations to claim conformance to FIPS PUB 127-2 simply by virtue of correct performance of these tests. It is reasonable, however, for purposes of procurement, to substantiate claims of conformance to FIPS PUB 127-2 by demonstrating correct execution of these tests.

Performance is recognized as a critical selection factor in many DBMS procurements. However, performance is not an issue for standards validation testing and is not measured by this test suite.

Currently, it is the responsibility of the implementor to prepare the driver scripts (i.e. operating system command files, shells, makefiles, runstreams, JCL) to execute the test suite.

NIST will maintain Version 6.0 of the SQL test suite as resources allow. NIST will evaluate error reports and distribute documentation of approved corrections via a World Wide Web page on SQL programs.

3. OVERVIEW OF SQL TEST SUITE

The SQL Test Suite contains schemas and programs to test an SQL implementation for various levels of the SQL standard. The test suite contains 18 users, 19 schemas, 208-463 programs and up to 849 test cases (depending upon test suite type). The Interactive Direct SQL version is smaller, with 379 programs and 660 test cases. In all there are 5887 files in the Version 6.0 distribution.

Before testing begins, the Test Editor, TEd, is installed. This editor is used (1) to install maintenance updates from NIST and (2) to facilitate installation and documentation of any changes made by the tester to the original test suite files.

Figure 1 shows a system flow diagram for basic SQL testing. Running an SQL test suite consists of 5 steps.

In step 1, the schema files are processed in some implementation-defined manner, typically using Interactive SQL.

In step 2, a few programs are run to insert values into the base tables. The contents of these base tables will remain unchanged throughout testing; i.e., these values will be restored by each program that changes them.

In step 3, the test programs are run to interact with the database tables. Each program contains logic to evaluate the database responses and determine whether a test passes or fails. This pass/fail decision is recorded by inserting a row into the table TESTREPORT. In general, programs may be run and rerun in any order.

In step 4, static values are inserted into the reference tables. These tables are required to produce the automated summary report. These tables are also a valuable resource to testers, since they can be queried interactively to create a variety of useful cross-references. In addition to the static values, rows are inserted (by the tester, via Interactive SQL) into tables FEATURE_CLAIMED and BINDING_CLAIMED to specify which profiles and interfaces are to be tested.

In step 5, the report programs are run to produce three listings for each profile tested:

PROBLEMS	- a listing of failed or missing results
TEST RESULTS	- a listing of the result (pass / fail / missing / not applicable) for each individual test in each interface tested
TOTALS	- counts for test results for each interface tested

To test Interactive Direct SQL, the tester visually evaluates the execution log (screen display) from running the SQL command files, assigns a pass/fail grade, and completes a check list.

System Flow Diagram

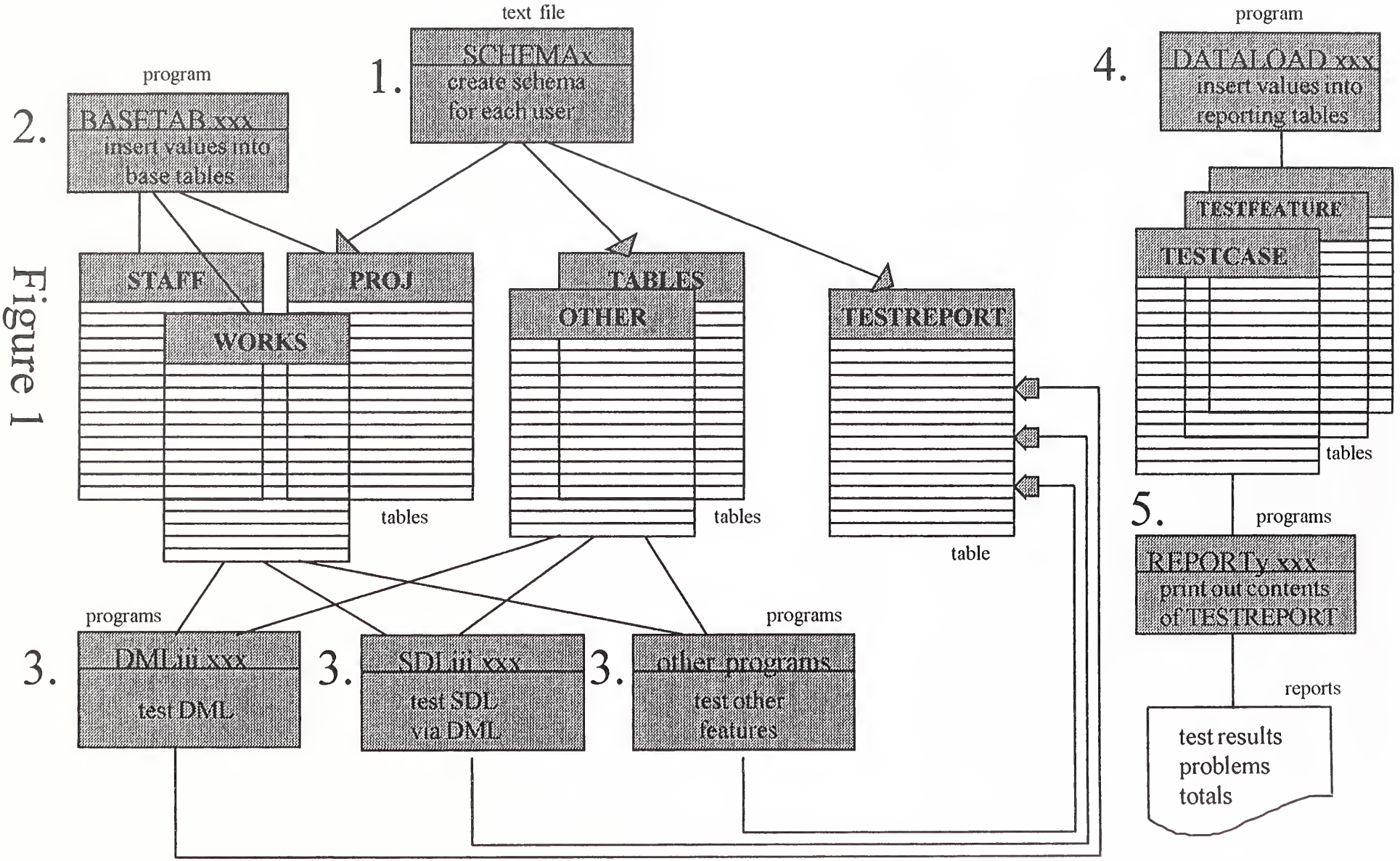


Figure 1

4. INSTALLING THE TEST SUITE

1. **Download files from the World Wide Web.** The SQL Test Suite is available as a set of compressed TAR files. You will download the default SQL TAR file and then will select from among the TAR files for different programming language interfaces.

   The full SQL Test Suite consists of 14 directories, organized as 11 TAR files. When you download the default TAR file, you will receive the basic set of directories needed by all the other directories. The default SQL TAR file contains a schema directory (SCHEMA), a reporting directory (REPORT), the Interactive SQL directory (SQL), and the utility/documentation directory (OTHER). You then download directories for the programming language interfaces you want to test. The 10 remaining directory names, consistent with our file-naming conventions (see item 5, below), are: PC, PCO, PFO, PAD, PPA, MC, MCO, MFO, MAD, and MPA. The most popular of these 10 directories is the PC directory, for testing the Precompiler C (Embedded C) interface.

2. **Uncompress the TAR files in a suitable directory.** Each TAR file will create its own directories. Lock the files so they cannot be changed. DO NOT CHANGE the test suite filenames in these directories (unless mandated by your operating system). The maintenance scheme depends on stable naming conventions.

3. **Create additional directories.** In preparation for a validation, we suggest that you create at least three additional working directories to store (1) permanent objects which you create such as driver scripts, TEd input files, makefiles, subroutine AUTHID; (2) generated intermediate objects that can be recreated at will and deleted *en masse*, such as temporary versions of the programs (output of TEd or precompiler), object modules, executables, Ted executable, etc.; and (3) audit trail/documentation objects such as logs showing schema creation and test program execution. These logs will be stored on tape or diskette after a validation, along with the permanent objects which you used to run the tests.

   The procedures for creating schemas, preprocessing, compiling, linking, and running will vary with the operating system, DBMS, programming language compilers, etc. The user of this test suite is responsible for creating the driver scripts (i.e. operating system command files, shells, makefiles, runstreams, JCL) to execute the test suite. Sample driver scripts for a few environments are included as APPENDIX A.

4. **Verify that all files have been received** for the test suite type being tested. Print the file RUN*.ALL (where * denotes a wild-card matching symbol in a file name) for the test suite type being tested. For example, the Embedded SQL C ("PC") test suite will contain a file RUNPC.ALL. This file lists the programs to be executed. Additional files are CHG*.TED and RPT*.SQL (i.e. CHGPC.TED and RPTPC.SQL for the "PC" test suite). Print these files also. All other files in the directory are programs of the type to be tested (e.g., Embedded SQL C programs - ending in "PC").

Note that each test suite type will contain a slightly different list of programs. Certain programs are tests of the Embedded SQL and are not meaningful tests for module language. For example, for the following programs you will find only precompiler versions: DML017 (tests WHENEVER) and DML063 (tests use of reserved words as host variables). Other programs are meaningful for only one host language. For each of the five embedded test suite types, there is a sample optional login program, AUTHID, which may be useful.

Note that directories PPA, MFO, and MPA contain programs which are not listed in the files RUNPPA.ALL, RUNMFO.ALL, and RUNMPA.ALL. These extraneous programs are tests for Transitional SQL features. The translation of programs into test suite types PPA, MFO, and MPA was begun, but not completed because of time constraints and limited demand. Interfaces PPA, MFO, and MPA will not be validated for conformance to Transitional SQL. Programs for validation of Entry SQL are complete in all test suite types.

5. **Identify test programs.** Each of our test programs is of the form CDRiii.xxx, DMLiii.xxx, SDLiii.xxx, FLGiii.xxx, MPAiii.xxx, MPBiii.xxx, ADAiii.xxx, CCCiii.xxx, COBiii.xxx, FORiii.xxx, ISIiii.xxx, ISTiii.xxx, PASiii.xxx, XOPiii.xxx, XTSiii.xxx, or YTSiii.xxx.

   The program prefix has the following meaning:

CDR	tests "integrity enhancement" to SQL -- **C**heck clause, **D**efault column value, **R**eferential integrity
DML	tests data manipulation language
SDL	test schema definition language via DML
ISI	information schema for Intermediate SQL
IST	information schema for Transitional SQL
FLG	SQL Flagger test
MPA	concurrency test, program A
MPB	concurrency test, program B, to be run at the same time as program A
XOP	X/Open Extension tests, for Embedded SQL C and COBOL only
XTS,YTS	Intermediate SQL programs donated by the CTS2 SQL2 project
ADA	Ada only
CCC	C Language only
COB	COBOL only
FOR	FORTRAN only
PAS	Pascal only

The xxx designates the type of program as follows:

.PAD	precompiler (embedded syntax) Ada program
.SAD	standard (calling module language) Ada program
.MAD	Module Language SQL called by a Ada program
.PC	precompiler (embedded syntax) C program
.SC	standard (calling module language) C program
.MC	Module Language SQL called by a C program
.PCO	precompiler (embedded syntax) COBOL program
.SCO	standard (calling module language) COBOL program
.MCO	Module Language SQL called by a COBOL program
.PFO	precompiler (embedded syntax) FORTRAN program
.SFO	standard (calling module language) FORTRAN program
.MFO	Module Language SQL called by a FORTRAN program
.PPA	precompiler (embedded syntax) Pascal program
.SPA	standard (calling module language) Pascal program
.MPA	Module Language SQL called by a Pascal program
.SQL	Interactive Direct SQL statements

The iii stands for an integer, assigned serially.

6. **Install the NIST Test Editor, TEd.** It is not necessary to use the NIST Test Editor for in-house evaluations. However, if you plan to have your product validated in the future, you will need to learn how to use the editor. Keep in mind that the explicit purpose of the editor is to facilitate running the test suite. Despite the natural resistance of users to learning yet another editor, and a batch editor at that, we expect that your investment of time will more than pay for itself.

   Our test editor is written in highly-portable C. The name of the source code program in the OTHER directory is TED.C. Please notify us immediately of any portability issues or errors which we have overlooked. We will assist you in debugging, if problems arise.

   Read the user documentation for TEd to understand why and how we plan to use this editor. If you change any of our test programs or schemas in the process of testing, we suggest that you use our editor to accomplish this. At the end of testing, you will have a single text file of batch editor commands (input to TEd) which documents all the changes you made. This single file will greatly simplify retesting (and validation) later. A separate text file may be used for each test suite type and for schema processing. Or, all change specifications may be stored in a single file.

   Each directory contains a sample or "starter" change file, CHG*.TED, which the tester should modify (with any editor) throughout the testing process. For example, the sample

change file for Embedded SQL C is CHGPC.TED. This file also contains instructions for installing implementation-dependent options.

Prior to scheduling for an SQL validation, the SQL testing laboratory will review the proposed file and will determine whether proposed changes are nonconformities or allowable modifications.

7. **Download the Version 6 maintenance file UPD600.TED.** The approved maintenance file, UPD600.TED, is an important input to TEd. This file should be copied into the same directory containing your driver scripts and CHG*.TED files. This file should be referenced with the "-t" option on the TEd command line to effect automatic maintenance of the SQL Test Suite. From time to time, this file should be replaced with the most recent version of the maintenance file. See the section titled "Software Maintenance" for more details.

   Since the TEd editor is used to install maintenance, it is important to process every file from the SQL Test Suite using the TEd editor with the official maintenance file UPD600.TED. This means that schema files, data files, test programs, reporting programs, etc. must have maintenance applied before they are used.

   Prior to scheduling for an SQL validation, the SQL testing laboratory will review the maintenance file to verify that it contains the most recent changes.

   Note that TEd has an option (**-o**) to write the edited program (or schema file) with a different name. If you want to change the downloaded filenames to satisfy naming conventions for your SQL processor, do so on output from TEd. For example, if your SQL preprocessor expects Embedded C programs (such as dml001.pc) to have a file extension of ec, then use the following command:

   ted -t upd600.ted -t chgpc.ted -o dml001.ec dml001.pc

   This command will read the downloaded Embedded C test program **dml001.pc**, apply the changes coded in file **upd600.ted**, apply the changes coded in file **chgpc.ted**, and then write the modified Embedded C test program **dml001.ec**. If the file **chgpc.ted** contained a command such as:

   sub dml001.pc //
   /E1/E2/

   This command would change the text **E1** to the text **E2** globally in only program **dml001.pc**. It would not change anything if program **dml001.ec** were input to TEd. So, it is important not to change filenames prior to applying the NIST file **upd600.ted**.

5. CREATING THE TEST SCHEMAS

1. **Set up accounts and passwords**, with the assistance of the DBA, for the following list of authorization id's:

HU	for file SCHEMA1
CUGINI	for file SCHEMA2
MCGINN	for file SCHEMA3
SULLIVAN1	for file SCHEMA4
FLATER	for file SCHEMA5
	for file SCHEMA6
	for file SCHEM11**
	for file SCHEM12***
CANWEPARSELENGTH18	for file SCHEMA7
SUN*	for file SCHEMA8
SULLIVAN*	for file SCHEMA9
SCHANZLE*	for file SCHEM10
CTS1***	for file CTS5SCH2***
	for file CTS5SCH3***
CTS2***	for file CTS5SCH1***
CTS3***	for file CTS5SCH5***
CTS4***	for file CTS5SCH4 ***
T7013bPC***	has no schema file
T7013PC***	has no schema file
XOPEN1++	for file XSCHEMA1
XOPEN2++	for file XSCHEMA2
XOPEN3++	has no schema file

* this authorization and schema are used to test Integrity Enhancement Feature (not used for one of the X/Open profiles, but required for FIPS 127-2).
++ this authorization and schema are used to test X/Open profiles only.
** this authorization and/or schema are used to test Transitional SQL profiles.
*** this authorization and/or schema are used to test Intermediate SQL profiles.

If the operating system does not allow a user id of length 18, then replace CANWEPARSELENGTH18 with another user id which is the maximum length allowed. If your SQL implementation uses the system user id as the SQL USER value or as the authorization id for privilege enforcement, then your computer system administrator should also create system accounts for these users. You may need to process schema and programs for each authorization id while logged in as that user.

The tester will need to choose passwords for the authorization id's. The SQL standard has no requirements concerning the passwords chosen. The tester should probably choose passwords consistent with in-house regression testing procedures, if possible.

2. **Define the schema for each of the appropriate authorization id's**, processing the schema definition text files in an implementation-defined manner; e.g., interactively. See

Step 1 of Figure 1. The file RUNSCH.ALL contains pseudo-scripts for creating the schemas. Use the SCHEMA text files as follows:

a. If your implementation is fully conforming to the standard, use SCHEMA1.STD through SCHEM10.STD. (See item (e) below for a discussion of privilege violation processing for schemas).

b. If your implementation needs semicolons to terminate SQL schema definition statements, use SCHEMA1.SMI through SCHEM10.SMI. Be sure to report this nonconformity in your final conformance analysis. Note that this will be counted as a nonconformity for a validation.

c. If your implementation needs semicolons and also requires CREATE UNIQUE INDEX instead of the UNIQUE declaration, then use SCHEMA1.NC instead of SCHEMA1.SMI. Be sure to report this nonconformity in your final conformance analysis. Note that this will be counted as a second nonconformity for a validation.

d. Otherwise, make whatever changes are needed to obtain a logically equivalent syntax for the schema definition. Keep in mind that changes are generally counted as nonconformities. It is essential that the tables HU.TESTREPORT, HU.STAFF, HU.PROJ, and HU.WORKS be installed; otherwise further evaluation of the test suite is impossible. It may be necessary to change the exact numeric data types to INTEGER instead of DECIMAL(I). (We chose to use exact numeric data types with a declared scale so that numeric precision would not be an issue.)

   Code your change specifications as inputs to TEd by editing the sample file CHGSCH.TED or CHGALL.TED, which is provided. If the schema contains a privilege violation, capture the error message as proof that a privilege violation has been detected and then edit CHGSCH.TED to specify removal of the offending text. The appropriate changes may already be coded in the CHGSCH.TED file as comments. If you are not using TEd, keep a log of your changes as documentation of either implementation-defined or as nonconformities to be reported in your final conformance analysis.

e. In an effort to simplify the process of editing schema files to remove schema privilege violation syntax (demonstrated by the SQL implementation to be a "fatal" error), alternate files are provided in the test suite. If your schema processor will entirely reject a whole schema because of a single error, then you may use the schema files ending in PV* and OK. For example, instead of using SCHEMA2.STD without and with TEd changes, you may use SCHEMA2.PV1 (with privilege violation #1) to demonstrate the fatal error, followed by SCHEMA2.OK (no privilege violation).

f. Note that there is a file of DROP TABLE and DROP VIEW statements, DROPHU.NC (clearly containing extensions to Entry SQL), to drop all views and tables created by SCHEMA1. There is a similar file, DROPSUN.NC, for SCHEMA8. These files may prove useful while you are analyzing your implementation's SDL conformance. Note that SCHEMA1 and SCHEMA8 are the only large schemas.

It is permissible to insert one or more statements at the beginning of the schema file to login or to establish the ANSI/ISO environment. You may add an implementation-defined terminator, such as a semicolon, at the end of the schema file. Other variations may be allowed after consultation with NIST or the SQL testing laboratory.

Verify that the tables have been created. Read the schema files and compare to tables and views existing in your data dictionary.

6. RUNNING/DEBUGGING WITH THE TEST PROGRAMS

1. **Code a procedure to create an executable.** Decide how to invoke all processors, such as precompilers, compilers, and TEd, to create an executable. Be sure to invoke any option on a processor which is needed to obtain the ANSI/ISO/FIPS behavior. Be sure to keep notes on which options or parameters are used (whether explicitly or implicitly), because they should be documented later in your evaluation. Read "Special Notes on Individual Programs" at the end of this section for variations in creating or running executables. Your procedure should invoke TEd as the first step, to install changes without touching the NIST original programs. On output from TEd, you should rename files to suit your SQL implementation's naming conventions. For example, you may want to rename DML001.PC to DML001.EC. We do not recommend renaming files before executing TEd, because the NIST maintenance TEd file will then need to be modified to specify the new names of the test programs.

   To save mass storage space, you will probably want to delete all intermediate files (such as outputs of TEd, precompilers, and compilers) after successful completion of the program.

2. **Code a script to invoke the procedure for each program.** For each test suite type, there is a text file, RUN*.ALL, listing the programs to be executed to support various claims of conformance. For example, file RUNPC.ALL lists the Embedded SQL C programs, organized into sections for the various possible claims. You may ignore programs not applicable for your claim. The text file may be edited to create a command file that will create executables or run the test suite. Read the notes in RUN*.ALL for additional instructions. Also, if you are going to claim support for individual features in addition to the Entry SQL Profile, you will need to use the automated reporting system to generate the correct list of programs. The files RUN*.ALL were generated by program REPORTA, and you may choose to run program REPORTA rather than using files RUN*.ALL.

The authorization id for each program is listed in these files to assist you in constructing driver scripts with the appropriate authorization id values.

If your SQL password is tied to your system login password (and this is not disallowed by the SQL standard) you will probably need to compile and/or execute each of the test programs while logged into the system as that user. Since the order of execution of the test programs is not important, you should group them by authorization id; i.e., in the interest of efficiency, you should write driver scripts which minimize login and logout.

3. **Solve the login problem.** The method of establishing a <module authorization identifier> is implementation-defined in embedded languages. We have chosen to code a CALL to a subroutine, AUTHID, with a variable, uid, containing the authorization identifier. This subroutine logs into the database with the given authorization. This solution may not work for you! You may need to move the authorization identifier into the driver scripts (external to the program) and logically delete the subroutine call. You may replace the subroutine call with an include statement or a CONNECT statement - see APPENDIX D. In Embedded SQL C, you may want to insert after the **#include <stdio.h>** statement some other statement which will effect initialization of your database. Or, you may solve the authorization identifier problem in some other way. Your report of test results should include a description of how your implementation handles login.

   Additional statements are allowed to connect to database components and to establish the ANSI/ISO environment. Basically, one or more statements to login and to establish the ANSI/ISO environment will be allowed near the beginning of each test file. For example, you may make a call to the server to invoke ANSI/ISO mode or to invoke ANSI/ISO-required features separately (e.g., SERIALIZATION mode, TRANSACTION mode, etc.). The call to AUTHID is a convenient hook for TEd to make these insertions. It is **very desirable** that all ANSI/ISO-required features be default features; however, it is allowable to invoke them as a group or individually. This invocation may take the form of precompiler/compiler parameters, configuration files, software installation parameters, calls from inside a program (prior to execution of test cases), etc.

   Since it is implementation-defined (in Entry SQL-92) whether the authorization is checked at run time or at compile time, the most straightforward approach is to precompile (prepare executables) and execute each program while logged in as the USER specified (by the variable **uid**) inside the test program. For client/server architectures where the catalog is not referenced until run time, it may be more convenient to compile all programs while logged on as a single user.

   Note that each test program verifies that the correct USER is logged in before executing the test cases. Incorrect USER value will STOP the program. You may easily use TEd to remove or replace this verification during debugging if your SQL implementation does not support the key word USER.

4. **Make global changes for implementation-defined parameters.** We have made an attempt to code certain constructs consistently so that global changes would be possible as a means of installing implementation-defined parameters. Edit the sample TEd input file, CHG*.TED, for the test suite type being run. Specify the appropriate implementation-defined values according to examples contained in the CHG*.TED file.

   Change authorization identifiers which are too long to shorter ones; e.g., CANWEPARSELENGTH18 may be changed to CANWEPAR. Exercise care, when making global changes to character string literals in the Pascal and Ada interfaces, not to change the length of the literal, since unequal-length comparisons and assignments may not compile or execute as expected.

   The precision of indicator variables is implementation-defined. We have chosen to use "short" in Embedded SQL C, "integer*2" in FORTRAN, "PIC S9(4) DISPLAY SIGN LEADING SEPARATE" in COBOL, and "integer" in Pascal. If this is not the correct precision or exact numeric type for your implementation, you **must change** the precision in the declarations of variables beginning with the name "indic"; e.g., indic1, indic2, etc. These variables occur in programs CDR003, CDR027, DML004, DML008, DML010, DML013, DML023, DML025, DML036, DML061, DML071, DML076, DML077, DML082, SDL026, etc. For Ada programs, the required package SQL_STANDARD (for SQL-92) or INTERFACES.SQL (for SQL-92 as corrected by Technical Corrigendum #2, TC2) specifies whether the indicator type is INT or SMALLINT.

   The precision of SQLCODE is implementation-defined for COBOL. We have chosen "PIC S9(9) COMP" in COBOL. If this is not the correct precision for your implementation, you **must change globally** the precision of the COBOL SQLCODE declaration.

   COBOL programs may need global changes in the Source-Computer and Object-Computer paragraphs.

   The precision of a CHAR column with DEFAULT USER in SCHEMA8 should be changed to reflect the implementation-defined length for USER.

   Implementation-defined keywords (possibly HOURS and PROGRAM), which are not allowed by your implementation as column and tables names, should be changed globally to some other word. This does not count as a nonconformity.

   Character set is implementation-defined. Contact NIST or the SQL testing laboratory if your character set (whether SQL processor or host language compiler) is incompatible with test suite materials. NIST or the SQL testing laboratory will discuss the implications with you and authorize appropriate workarounds.

5. **Using TEd, make global changes for documentation purposes.** Code a TEd substitution for "59-byte ID" to identify the SQL product and version being tested and the test platform, including hardware and operating system. If the length exceeds 59, then the replacement string may overflow COBOL and FORTRAN source code margins. (Compulsively wordy testers should investigate the **-c** option in TEd to specify multiple-line substitutions.)

   For FORTRAN, Ada, and Pascal interfaces, determine how to print date and time for your system. Code TEd substitutions for "date_time declaration" and "date_time print". These substitutions will differ among host languages and operating systems. C and COBOL programs have already been modified to print date and time using standard features.

6. **Using TEd, make global changes for nonconformities** which must be resolved before further testing is possible. These global changes should be included in your conformance analysis as nonconformities. All test suite materials must be supported correctly by the SQL implementation being tested. That is, in addition to the obvious test programs, all test suite materials such as schema files, dataload programs, reporting system files (schema files, report dataload program, programs reporta and reportb), are considered part of the demonstration of conformance by the SQL implementation under test. Any changes to any of these materials necessitated by nonconformities of the SQL implementation are to be documented as nonconformities.

   If you are testing Embedded SQL COBOL and your SQL implementation does not support the COBOL numeric data type [USAGE] DISPLAY SIGN LEADING SEPARATE, you will need to make global changes to the COBOL programs (.PCO and .SCO) and the COBOL modules (.MCO). TEd can be directed to change only those declarations within the scope of the BEGIN and END DECLARE SECTION.

   If your implementation of SQL does not support a direct declaration of SQLCODE, but instead requires an INCLUDE SQLCA statement, you should globally delete the declaration for SQLCODE and insert one for the required structure. For language C, you may want to direct TEd to delete all lines containing "long SQLCODE" and insert the required INCLUDE SQLCA statement before the text "main()". This will be counted as a nonconformity.

   If SQLCODE has some other name in your product, such as SQLCDE or sqlca.sqlcode, you will want to make a global change to the programs. If the substitute value is longer, you may also need to split lines which become too long. In C, if the substitute value is longer, you may prefer to "#define SQLCODE sqlca.sqlcode". This will be counted as a nonconformity.

   We have attempted to code certain SQL constructs in a consistent manner, so global changes would be possible. These constructs include:

EXEC SQL BEGIN DECLARE SECTION
EXEC SQL END DECLARE SECTION
EXEC SQL <key word>

We have taken care to code Embedded SQL statements so the global insertion of a line of code (such as a "print" of SQLCODE) either before or after executable SQL statements will not disrupt the logic of any "for" loops or "if" blocks containing them.

7. **Begin running the script** to invoke the programs listed in RUN*.ALL. Run program BASETAB to load the primary tables. The correct authorization identifier for BASETAB is HU. Then run programs CUGTAB, FLATTAB, SUNTAB0, SUNTAB1, SUNTAB2, SUNTAB3, and SULTAB1 with the authorization shown in RUN*.ALL. For testing Intermediate SQL, run program CTS5TAB. For testing of X/Open profiles, run XBASETAB. For testing X/Open profiles without Integrity Enhancement Feature, there is no need to execute programs for authorizations SUN and SULLIVAN. See Step 2 of Figure 1. Check the printout to verify that the data has been inserted. These programs may be rerun at any time to re-initialize the data in the primary tables.

8. **Run each of the programs** which are appropriate to the test suite type. See Step 3 of Figure 1. Typically, if a problem occurs in one of the test programs, the tester performs the following steps: (1) determines the cause of the problem, (2) changes the program by modifying input to TEd, (3) recreates the executable program, and (4) runs the modified executable program. Often, for the purpose of debugging, it is helpful to run the Interactive Direct SQL version of a program. The on-line user interface is often very informative and the SQL statement in question can be rapidly modified and retried until the problem is isolated. The test programs may be run and rerun in any order (except for programs beginning with MP, which must be run in pairs). Other exceptions are the sequence of programs YTS790 through YTS792 and the sequence of programs YTS793 through YTS795 for Intermediate SQL, as well as the sequence of programs XOP719 through XOP723 for X/Open profiles. Each of these sequences must be run in ascending order. See Special Notes on Individual Programs below.

   There are thirteen pairs of programs for concurrent testing:

   MPA001 and MPB001 through
   MPA008 and MPB008 for Entry SQL

   MPA009 and MPB009 through
   MPA013 and MPB013 for Transitional SQL

   Each pair is to be run concurrently, either from separate terminals or windows or started as separate batch processes. All MPA and MPB programs for a given test suite type may

be run at the same time. Any pair of concurrent programs may be run and rerun in any order.

Start the MPA program first, and then a few seconds later or when prompted by the MPA program, start the MPB program. It is common to see both programs issue messages to start the companion program, even after the tester has started both.

The concurrency programs contain tuning variables which may be used to lengthen the workload or planned waiting periods. This will allow programs to "interleave" better, as required by program logic to get a "pass."

9. **Use TEd to install changes.** If the SQL language for a given test (**except** for a FLG test) prevents a program containing several tests from running, use TEd to delete the entire problem test. Then rerun the program to exercise the remaining tests. Note that the missing test will be reported as a "fail" by the automated reporting system. Make only the global changes to the FLG programs. Do **not** make any other **special** changes (except FLG005) to these programs. (If an SQL Flagger test does not compile and execute, then the reasons given by the implementor for not compiling or executing may constitute the "flagging" required by FIPS PUB 127-2.)

   Another approach is to change the syntax of the problem test to syntax acceptable to your DBMS. This would allow you to further evaluate the SQL implementation; however, it would probably give a false test result of "pass." Code these changes in the SYNTAX DEFICIENCIES SECTION of the CHG*.TED file. This file is a log of changes applied to tests which fail syntactically, although they may pass functionally.

10. **Run PREDML.** When you are ready for a final analysis of a test suite type, remove all rows for that test suite type from the table TESTREPORT. This will eliminate conflicting results caused by earlier rerunning of changed tests. This is done by executing the program PREDML for authorization identifier HU. Then rerun all the test programs. Capture the screen printout of this final run as part of the documentation of your testing.

7. RUNNING THE AUTOMATED REPORTING SYSTEM

1. **Install the Reporting Structure.**

   Run the following in Interactive SQL as user HU:

   a. REPORT.SQL, in directory REPORT (creates the tables) See Appendix H.1.
   b. DATALOAD.SQL, in directory REPORT (loads the static data)
   ** If DATALOAD.SQL causes problems, refer to the long instructions on the media, file REPORT_L.DOC in directory REPORT.

Compile the following programs:

a. REPORTA (any embedded or module language)
b. REPORTB (any embedded or module language)
c. REPORTZX.C, in directory REPORT (ANSI C with no embedded SQL)

Put the executables for REPORTA, REPORTB, and REPORTZX together in a directory where user permissions will allow the creation of temporary files.

2. **Execute the Reporting System.**

a. Insert one or more of the following values into BINDING_CLAIMED according to which bindings you wish to test: 'PCO', 'PFO', 'PC ', 'PPA', 'PAD', 'MCO', 'MFO', 'MC ', 'MPA', 'MAD', 'SQL'. For example, to test Embedded C: INSERT INTO BINDING_CLAIMED VALUES ('PC ');.

b. Insert the profile identifiers for the profiles that you want to test into FEATURE_CLAIMED. For example, to test Transitional SQL:

```
INSERT INTO FEATURE_CLAIMED VALUES ('P135');.
```

The profile identifiers can be found in the diagram PROFILES.PS (in directory OTHER) or by typing the Interactive SQL command:

```
SELECT * FROM REPORTFEATURE
    WHERE FEATURE1 LIKE 'P%';
```

The typical FIPS 127-2 validation (for Entry SQL) will use only the following inserts into FEATURE_CLAIMED:

```
INSERT INTO FEATURE_CLAIMED VALUES ('P125)';
INSERT INTO FEATURE_CLAIMED VALUES ('P325');
INSERT INTO FEATURE_CLAIMED VALUES ('P415');
```

NOTE: Subprofiles are automatically selected when you select a parent profile. Refer to the diagram PROFILES.PS in directory REPORT or to Appendix H.3. For example, if you select 'P135' (FIPS 127-2 Transitional SQL), the following subprofiles are automatically selected: 'P110' (FIPS 127-2 Entry Syntax Flags), 'P120' (ISO/IEC 9075:1992 Entry SQL), and 'P125' (FIPS 127-2 Entry SQL).

c. Run REPORTA. If you are testing a combination of individual features, rather than one of the established profiles, then you will need to capture the output to get a list of programs that you must run to test the feature(s) that you selected. If only some of the tests in a particular program are required, REPORTA will also provide a TEd change to delete the extra tests. Include the TEd change specification in your TEd file. Run any additional test suite programs required for your claim.

d. Run RPT*.SQL for each binding claimed (test suite type). Before producing the final reports, it is often useful to know if there are any unexpected failures, any missing tests, or any conflicting results (both "pass" and "fail" for a single test). An efficient way to ensure that the table TESTREPORT contains no surprises is to run the Interactive Direct SQL queries in the appropriate RPT*.SQL file. For example, for Embedded SQL COBOL, run RPTPCO.SQL. Run it under authorization HU. RPT*SQL must be run after REPORTA in order to detect missing tests. If the query results of RPTPCO.SQL accurately reflect the testing, then you are ready to run the final report.

e. Run REPORTB to generate the temporary files used by REPORTZX.

f. Run REPORTZX and look at COMBINED.LST to see the results of testing. As a general rule, you should always run REPORTB immediately before running REPORTZX.

g. If REPORTZX shows a test number is missing or a test has failed incorrectly, then you will need to follow restart procedures below.

NOTE: The diagram REPORTIN.PS in directory OTHER shows the data flow of the above procedure.

NOTE: Programs REPORTA, REPORTB, REPORTZX are run only once to report on all bindings claimed. This is a different architecture from the one in Version 4.0, where the reporting programs were run for each binding claimed.

3. **Restart Procedures for the Reporting System.**

If you do find a test case with conflicting results (e.g., both a "pass" and a "fail") for the test suite type being tested, then the test will be assigned a "fail" by the automated reporting system. If a test fails for an operational reason (such as starting two copies of MPA001 at the same time) and does not represent a real failure (a nonconformity) and you wish to rerun to demonstrate conformance, then delete all results for that test number and that test suite type. You may then rerun the program containing the test. A similar procedure is allowed for a test case which fails because it was executed incorrectly; e.g., before loading the initial data into the tables. For example, **DELETE FROM HU.TESTREPORT WHERE TESTTYPE = 'PCO' AND TESTNO = '0099'**; will remove all results for the Embedded SQL COBOL test number 99. Note that duplicate test results in TESTREPORT do not cause a problem.

If test cases are missing because you simply failed to run the containing program, then you need to run the missing program(s) and rerun REPORTB and REPORTZX. It is not necessary to rerun REPORTA. Refer to Appendix H.2 to see the structure of the reporting system.

If you want to change the bindings or features claimed, you will need to adjust tables BINDING_CLAIMED and FEATURE_CLAIMED using Interactive SQL. You will then need to rerun programs REPORTA, REPORTB, and REPORTZX.

8. PREPARING FOR VALIDATION OR REGRESSION TESTING

1. **Prepare the final CHG*.TED file.** Edit the cumulative change file which has been used to specify changes to TEd. Remove any of the original NIST comments or examples which are not applicable. Analyze each change and assign it to one of the sections in the CHG*.TED file. Create a new section if necessary and document each change with a comment explaining the purpose of the change.

2. **Select a testing strategy.** Now that all test programs have been debugged on your SQL implementation, it is time to plan for regression testing and/or validation. You will want to develop procedures to run the tests from beginning to end with minimal effort. We have found, through experience conducting validations, there are two approaches generally used to execute the test suite.

   a. Process/Link/Execute: For each test program, one at a time: (1) prepare the executable program (execute TEd, precompiler, compiler, linker), (2) execute it, and (3) delete it. This saves on mass storage and generates a complete log. This approach is recommended for debugging and for validations running only one or two test suite types.

   b. Prepare executables in advance: An efficient way to run multiple test suite types is the following: (1) prepare executable versions of each program, (2) prepare a driver script to execute PREDML and all of the test programs (except the MPB programs) in the correct order, (3) turn on the "screen capture" or log and start the driver script, (4) when the MPA programs call for the MPB programs, start the appropriate MPB program from another terminal, or specify in the driver script that MPB starts a few seconds after MPA, (5) turn off the "screen capture" or log and print or review the output, and (6) rerun the reporting programs in batch if the page ejects got lost in the "screen capture."

      Steps 3 through 5 of this strategy typically take us 30 minutes, while step 1 takes 3 hours; although we have seen platforms which complete all steps in 20 minutes. Step 1 can be broken up into several driver scripts, submitted as separate processes or run from different terminals. There is no need to worry about concurrency during step 1. While we are running steps 3 through 5, we do worry about concurrency, and we do not run any test programs from other test suite types.

   Each of the concurrency program pairs uses tables used by no other programs in that test suite type. Consequently, all thirteen program pairs may be run simultaneously, and they

may overlap the stream of other test programs in execution without concern for accidentally altering data used by another test program. However, no two test suite types should be executed simultaneously.

9. SPECIAL NOTES ON INDIVIDUAL PROGRAMS

AUTHID — This is a sample subroutine which can be modified with implementation-defined statements to accomplish login and/or to establish the ANSI/ISO test environment. It is entirely optional. Instead of using a subroutine to login, the tester may globally replace the call to AUTHID in the main routine with some implementation-defined statement.

CCC004 CCC009 — Each of these C language programs contains an additional subroutine, CCC004S and CCC009S respectively, which needs compilation and additional link statements. Embedded SQL C test suites contain programs CCC004 and CCC009; Module SQL C contains only program CCC004.

DML015 — Tests in this program are not independent of each other. If one test is deleted, other tests in the program may be affected.

DML035 — In COBOL, this program is optional (and may not compile) because it contains a nonstandard variable with USAGE COMP-1. Change the data type to suit the compiler.

DML038 — Test number 0205 prints a Cartesian product of three tables. Do not panic and cancel the program when you see 360 lines scrolling across the screen. Do cancel the job, however, if you see more than 400 lines!

DML044 DML047 — In Pascal, there is no standard way to continue a character string literal onto the the next line. In order to test database columns for long character strings, we coded a procedure, "concat," to create a long character string value. "concat" is used in test 0216 to assign a 118-byte value to variable vtr119 and in test 0222 to assign a 240-byte value to variable STR240. If procedure "concat" does not work for your compiler, propose another method to assign the long character string values.

DML063 — This is an Embedded (only) SQL program to test the use of SQL key words as host identifiers. A vendor may declare up to one third of these key words to be "reserved" and not allowable as host identifiers. The CHG*.TED file must contain specifications to disallow the "reserved" key words.

DML102 — Two copies of this program may be run in place of MPA013.PC and MPB013.PC. This program is a more elegant approach to concurrency testing, but unfortunately, it

contains C language constructs which do not translate easily into the other test suite types. This option is available only in Embedded SQL C.

DML103 Two copies of this program may be run in place of MPA012.PC and MPB012.PC.

DML116 This program contains two subroutines for each Embedded SQL main routine. For module language interfaces, there are two host language subroutines and three modules to be linked together.

DML169 Check that some form of flagging relevant to the extension is present for each test that compiles. Read the comments in the program.

FLG005 This SQL Flagger program must be modified to execute two SQL extensions of the tester's choosing. If the SQL implementation does not support a character function or integer function extension, then any extension may be coded. The tester must then verify that the extensions are flagged. If the SQL implementation does not support extensions at all, this will need to be demonstrated after discussions with NIST.

FLG* All SQL Flagger programs, except FLG005, which fail to compile because the extension being tested is not supported, are judged to pass by default. Test cases for which flagging is based on "catalog lookup," rather than "syntax only," are optional (for Entry SQL); therefore flagging is not required, even if the feature is supported. These optional programs are FLG006 and FLG009.

MP* MPA* programs should always be started before the corresponding MPB* program. If any pair of concurrent programs have no screen display for 15 minutes at the point where deadlock is expected, then cancel the programs and consider them passed. The standard does not require deadlock management. It only requires transaction serializability when the implementation successfully processes SQL statements. If any pair of concurrent programs appears, from the screen display, to be restarting transactions over and over without progress, contact NIST for additional program fixes (program code) to introduce asymmetries into the transaction workload and to provide expanding time gaps between transactions.

XOP719 through XOP723

This set of programs from the X/Open profile are not independent of each other. The tests are associated with GRANT and REVOKE PRIVILEGES so you need to run them in the following order: XOP719, XOP720, XOP721, XOP722, XOP723. If any of the tests in this set need to be run again, you must run the restart SQL command file XRECRE1.NC after XOP723, and then run this set of tests again.

XTS713 This program uses AUTHORIZATION T7013bPC. Some implementations may need to create a system or database authorization for this.

XTS725-8 Check that some form of flagging relevant to the extension is present for each test that compiles, specifically check that all FULL SQL functionality is flagged. Read the comments in the program.

XTS734 This program tests for the National Character Data type in comparison predicates. It is necessary to incorporate the appropriate National Character set into the program by replacing the TEd hook "_VANGELIS" to the correct character set name.

YTS767 Three tests in this program should allow alternate syntax for the CHECK clause. If there is anything but a "pass" for tests 7544, 7545, or 7546, review the implementation's syntax and substitute, using TEd, with equivalent syntax.

(YTS790 through YTS792) and (YTS793 through YTS795)
These sets of programs from Intermediate SQL are not independent of each other. Each set must be run in ascending order as they test for the granting and revoking of privileges. To restart the first set, user CTS1 should issue the command REVOKE USAGE ON CHARACTER SET CS FROM CTS2 CASCADE;

YTS814 This program applies to module language interfaces only. It contains special characters in the LATIN1 character set which may be inadvertently replaced by utilities (e.g., editors or E-mail) manipulating these programs.

10. RUNNING THE INTERACTIVE DIRECT SQL TEST SUITE

Establish Test Environment

Interactive Direct SQL allows the implementor considerable freedom in designing print formats for the screen. Unlike our programming-language test suites, pass/fail grades are assigned to each test by a tester, rather than by program logic. Each of our test files is designed to execute some SQL statements and then have the tester examine the results on the screen (or standard output) for appropriate responses.

Most Interactive Direct SQL implementations have some command to "run" a text file of SQL commands; i.e. execute a named file. As a matter of convenience, all the Interactive Direct SQL tests, except the concurrency tests, should be "run" rather than typed. Support for "interactive access to the database," as specified by Section 16.5 of FIPS 127-2 will be demonstrated during the concurrency tests.

We anticipate a variety of screen presentations and do not have any fixed criteria for column headings, numeric formats, character string wrapping, error messages, or other status feedback.

Our test files have comments after SQL statements, detailing the response needed on the screen to pass that test. All PASS comments for a given test must be judged to pass, otherwise the test fails.

An SQL Flagger test that fails is simply not applicable, since flagging applies only to extensions that are supported.

Experiment with a few test files to decide how your SQL implementation will allow you to evaluate responses against our PASS criteria.

If you have some command to force all SQL commands and comments to print on the screen, interspersed with the SQL query results, then you can simply run all the test files and capture the screen printing for later evaluation. This command may be something like ECHO or VERIFY.

If not, then some creativity may be needed to force the comments to print on the screen. See the section "Printing the Interactive Direct SQL Comments" below.

If your SQL implementation has a comment style that is different from the format used in our test files, you may globally convert the comments in the test files to your format. In all probability, you can do this with a few TEd commands. Contact NIST for suggestions if it is not obvious how to do this.

If your SQL implementation has a line-continuation style for SQL statements that is different from the format used in our test files, you may globally convert the test files to your format. Although it may be possible to use TEd to accomplish this, you should review the C program ATERM which was designed to perform this reformatting. ATERM is in the OTHER directory.

Solve the Login Problem

There are many acceptable ways to run the Interactive Direct SQL files. You can actually log in as the correct USER. You can create a script which logs in as the correct user and then runs a list of files for that user. You can modify, via TEd, the "-- AUTHORIZATION" comment at the beginning of each module to login or attach to a schema.

Execute Test Files

Run the data load file BASETAB.SQL if data has not been loaded earlier by another test suite type. Also run data load files CUGTAB.SQL, FLATTAB.SQL, SUNTAB0.SQL through SUNTAB3.SQL, and SULTAB1.SQL. For Transitional SQL run the data load file SCHEM11.STD. For Intermediate SQL, run the data load files SCHEM12.SQL and CTS5TAB.SQL. The order of execution of data load files is important. Run all of the *.SQL test files. The order of execution of test files is not usually significant. Read the file RUNSQL.ALL for a listing of the test files to be run. Print file CHECK.LST to use as a worksheet for recording pass/fail results.

SQL files named MPA010* or MPB010* must be run in the order listed in RUNSQL.ALL; i.e., MPA010A, MPB010A, MPA010B, MPB010B,... MPA010G. Read comments in these files.

If you need to modify a test file for whatever reason, do not change the downloaded *.SQL file directly. Instead, edit the file CHGSQL.TED to specify the changes, and use our editor, TEd, to install the changes. For debugging only, it may be more convenient to work with a copy of the downloaded file which is executed and then modified interactively by a local editor.

If at any time, you suspect that a failed test has corrupted the initial data in the HU directory, run the files SEEHUE.SQL (to verify that all tables in the HU schema which should be empty are empty) and SEEHUD.SQL (to view tables which should have rows inserted by the dataload programs). Delete rows from any table listed in SEEHUE.SQL, if necessary. And rerun BASETAB.SQL at any time to re-initialize the tables listed in SEEHUD.SQL.

Execute Concurrency Test Procedure

There is only one concurrency test procedure for Interactive SQL. Print file MPQUIC.TXT and follow the instructions for two testers (or equivalent). Previous concurrency procedures were very time-consuming and duplicated concurrency testing already done by the programming language interfaces. The MPQUIC procedure verifies that serializability is turned on for the Interactive SQL interface.

Interactive SQL test files MPA001* and MPB001* through MPA008* and MPB008* have been superseded by the MPQUIC procedure, which is much more efficient. These files are still included in the Interactive SQL test suite because they may be helpful for SQL products which do not have a programming language interface. Refer to APPENDIX I for detailed instructions on how to execute these tests.

Evaluate Test Results

Tests may be evaluated on the spot or at some later time by examining the log or captured screen output. A pass/fail grade is assigned to each test run. The tester should fail any test for which one or more of the PASS comments is judged to have failed. Tabulate the pass/fail results on worksheet CHECK.LST, a file in the SQL directory.

Tests for access violations may abort for an Interactive Direct SQL implementation. This is acceptable. We consider this a demonstration of support for SQL GRANTS. The tester should simply capture the screen output and then delete that test from the test program before rerunning, so that the remaining tests in the test program can be run. Capturing the screen output is needed to get a "pass" for tests which abort due to access violations.

After an UPDATE, INSERT, or DELETE statement, there may be a PASS comment that a given number of rows were affected. If the SQL implementation being tested does not provide this information for successful completion, then simply ignore the PASS comment. In most cases, there will be a subsequent SELECT statement which double checks (verifies) the success or failure of the database modifications.

For a SELECT statement, the rows will be displayed on the screen and should be counted by the tester. If there is a PASS comment immediately following the SELECT statement, the visual count of rows selected must match the count required by the PASS comment (the default count is 1 row). In addition, if the SQL implementation being tested echoes a row count, that row count must be accurate.

References to SQLCODE and/or SQLSTATE in Interactive Direct SQL are a carry-over from the original programming language tests. The revised FIPS PUB 127-2 does not require explicit support for SQLCODE or SQLSTATE. Rather, the tester should expect some implementation-defined message or messages which map to the SQL concepts of no-data (SQLCODE = 100) and run-time error (SQLCODE < 0) or to the standard SQLSTATE conditions and values. If, however, explicit values for SQLCODE or SQLSTATE are given, then an incorrect value is a failure. (Note that it is allowable for an SQLCODE 100 to be returned after every successful SELECT statement, since the SELECT may be implicitly implemented as a cursor, fetching until end of data, where SQLCODE 100 is expected.)

SQL Flagging is now required for Interactive Direct SQL. Failure to flag the supported extensions in the programs FLG001, FLG005, and FLG008 will be a nonconformity in the Interactive Direct SQL interface. However, since flagging which requires schema access is optional, failure to flag for test files FLG006 and FLG009 is not considered a nonconformity.

Print the Interactive Direct SQL Comments

If your implementation of SQL has some command to force all SQL commands and comments to print on the screen, then you may ignore this section.

You will note that the comments in our *.SQL files are coded in a very stylized manner. The goal is to provide "hooks" for you to do global changes with our editor to convert these comments into some printable object. This may be a system call for printing on the screen or some other facility allowed by your SQL implementation. Only the comments beginning with

-- TEST: and **-- PASS:**

are of interest in test evaluation.

If all else fails, you can always use SQL. You can change the comment

```
-- PASS:0247 If count = 3?
```

into

```
SELECT '0247 If count = 3' FROM HU.ECCO;
```

Using our test editor, TEd, the commands:

```
sub *.sql /-- PASS:/
          //"/
```

```
!               replace embedded single quotes
sub *.sql /-- PASS:/
      /?/' FROM HU.ECCO;/
sub *.sql /-- PASS:/
      /-- PASS:/SELECT '/
```

will cause SQL to print the PASS comments. Similar global changes will force the "-- TEST" comments to print. The one-row table called ECCO that is needed for this gimmick to work has already been created.

11. EVALUATION INSTRUCTIONS

The tests in the SQL Test Suite are designed to conform to Intermediate SQL-92. Only programs designated as FLG should contain extensions to Intermediate SQL-92.

1. **Examine the log of changes** you made to each program or to SCHEMA files. If a change was made to SQL syntax to allow your implementation to complete the test using nonconforming syntax (such as FOR UPDATE in the DECLARE CURSOR statement), or even alternate conforming syntax, you should report this change as a nonconformity in your analysis. You should document, for informational purposes, any changes allowed for implementation-defined options.

2. **Report errors in the test suite.** If a change was made to SQL syntax to correct an error (programming bug or interpretation of the standard) in a test, then report this error to NIST or an SQL testing laboratory. You should report this discrepancy in your conformance analysis as an unresolved issue.

3. **Examine the output of REPORTZX.** Three reports are produced by program REPORTZX: PROBLEMS.LST, TOTALS.LST, and TESTNO.LST. These three files are rewritten as one file, COMBINED.LST, to simplify printing. Print COMBINED.LST.

   The PROBLEMS list should be almost empty for a conforming SQL implementation. This is a list of test cases which require attention because they are missing or failed. Typically, a missing or failed test is a nonconformity; however, there are a few exceptions. Test number 0399 (for Embedded SQL C only) is always missing, because it require visual inspection in order to pass. **See Special Notes on Individual Programs above.** Missing flagger tests (programs FLG*) are judged to pass if they will not compile because the tested extension is not supported. The compilation and execution log of any test on the PROBLEMS list must be reviewed.

   The TOTALS list should be compared for accuracy against the file REPORT.TOT in the directory REPORT. The file REPORT.TOT contains control totals for the most common profiles. Other profiles are possible. Failure to match the control totals means either the control totals are wrong (NIST will fix them), the SQL implementation has errors

processing the SQL queries in REPORTA and REPORTB (this is a nonconformity), or there is some operational error in running the automated reporting system.

The TESTNO list shows the contents of TESTREPORT, rearranged for reporting in the profiles claimed. Test numbers are sorted within the appropriate subprofile. The result (pass / fail / missing / not applicable) for all test suite types appear on a single line, facilitating comparison among the supported interfaces. A test number from TESTREPORT may appear more than once, if it is required by more than one profile; e.g., X/Open and FIPS 127-2 Entry. A test number from TESTREPORT will not appear in the TESTNO list if it is not required by any of the profiles claimed. Duplicate results (e.g., duplicate "pass" results) will appear only once. Conflicting results (e.g., "pass" and "fail" results for the same test number in the same interface) will appear as a "fail".

The TESTNO list subprofile for "Entry Syntax Flags" is special; It must be evaluated further. For each "pass" result, the SQL implementation is judged to be supporting an SQL extension and should demonstrate warning message(s) that an extension is being used (as required by Section 10.d of FIPS 127-2). For each "nogo" or "missing" a fatal compilation error or run time error must appear in the execution log for the test. This error demonstrates that the SQL implementation does not support the SQL extension in the test, and therefore does not need to flag the extension. In summary, each documented error in the "Entry Syntax Flags" subprofile satisfies its test objective, and is judged to be conforming.

The TESTNO list subprofiles for "Entry Catalog-Lookup Flagging" and "Miscellaneous Informational" are strictly informational, listing the optional tests and showing the pass/fail results for those tests. These subprofiles are routinely run for FIPS 127-2 testing.

4. **Prepare a Report Package**, including:

   a. Full description of test environment. This would include date, testing staff, hardware make/model, software description and versions for precompiler/ compiler/linker/SQL engine/communications software/libraries, special installation parameters, special processor parameters. For client/server architectures, describe both platforms and communications software/hardware.

   b. Automated Summary Report COMBINED.LST

   c. Listing of global changes to schema and program files (or listing of the CHG*.TED files). (This would include data types/precision for SQLCODE and indicator variables.)

   d. Listing of specific changes to individual schema or program files (or listing of the CHG*.TED files).

e. Log of schema creation run. Include earlier logs of failed schema runs containing privilege violations which were removed.

f. Screen capture or printout of executing programs showing pass/fail printout.

g. Printouts of FLG005 through FLG013 SQL/preprocessor compilations, with the "SQL Flagger" turned on and again with the flagger turned off. Failure to flag any supported extensions in the programs FLG005, FLG008, and FLG010 and FLG013 is a nonconformity in any of the test suite types, including Interactive Direct SQL. However, since flagging which requires schema access is optional, failure to flag in program FLG006 or FLG009 is not considered a nonconformity.

h. Printouts of SQL compilations detecting syntax errors for privilege violations or standard SQL run-time errors detectable at compilation time. (These will document the detection of the error and may be needed to change a "missing" to a "pass")

i. Listing of driver scripts (i.e. operating system command files, shells, makefiles, runstreams, or JCL) used to process the schema and program files, including any special parameters settings.

j. Description of how login was accomplished, including a listing of the modified AUTHID or included file.

k. A diskette or tape copy of all schema and program files (as modified and run), to be used for regression testing and to resolve any future disputes over changes made to the SQL test suite.

l. A diskette or tape copy of a full Interactive Direct SQL run (with the possible exception of the MP* test cases). You may then avoid the tedium of rereading the Interactive Direct SQL log at some future date by using a DIFF utility for regression testing.

If you are a vendor, you will want to use this information (1) to guide your standards-conformance effort, (2) to assist federal agencies or other users who may be evaluating the conformance of your product to FIPS PUB 127-2, or (3) to demonstrate to the SQL testing laboratory, when you request to schedule a validation, that you are prepared to conduct the testing efficiently.

If you are a user, you may use this information (1) as part of your acceptance testing of implementations claiming conformance to FIPS PUB 127-2 or other SQL profiles, (2) as input to your RFP process, or (3) as a measure of your current SQL implementation's conformance and hence a measure of the portability of your standard application programs using standard SQL language.

12. SOFTWARE MAINTENANCE

As users of the test suite work with the software, questions arise about allowable changes to the programs and schema files. Questions usually fall in one of the following areas: (1) interpretation of the ANSI or ISO SQL standards, (2) interpretation of FIPS 127-2, (3) possible errors in the test suite, (4) procedures for running the test suite, (5) procedures for validations.

SQL Updates

Answers to questions of general interest which are answered in narrative form will be distributed to test suite users as SQL TESTING UPDATE newsletters. These newsletters are to be viewed as updates to our test suite documentation and to our published procedures for validations.

TEd Maintenance File

Some questions result in changes to the test suite software. These approved maintenance changes will be distributed periodically to test suite users on paper. The changes are formatted as inputs to the Test Suite Editor, TEd. The changes should be appended to the cumulative maintenance file, UPD600.TED.

The cumulative maintenance file is available in machine-readable formats. Users may access the maintenance file on NIST World Wide Web pages.

Validation Considerations

When an SQL testing laboratory performs a validation, a special validation version of the test suite may be used. This version contains the same SQL test cases as the original distribution version. There have been only a few changes made to the software in the validation version. With these changes, test laboratory staff can tell that the validation is being run from the software they bring, rather than a distribution version downloaded earlier.

The validation version does not contain maintenance updates, so your TEd file UPD600.TED must have them. Your TEd file should have exactly the same effect on the validation programs as on the programs of your distribution version. Before the validation, test laboratory staff will review your TEd file(s) and make sure that all mandatory maintenance changes are included. They will also review any implementor-proposed changes and determine whether these changes are allowed under the procedures of the test suite or whether these changes are nonconformities.

13. SQL TEST SUITE REFERENCE MATERIALS

A variety of references are attached to assist you in your testing and evaluation.

APPENDIX A. Examples of Driver Scripts

These are just examples which work in certain environments. There is no guarantee that any of these will work for you. Note that TEd acts as a filter for NIST schema files and program files.

APPENDIX B. Base Data for Primary Test Tables

This is a schematic representation of the simple tables used for most of the tests in the HU schema. This represents the contents of the tables STAFF, PROJ and WORKS after the program BASETAB has been run. Note that most of the other tables in the HU schema are empty. In order to make tests reproducible, programs in the test suite will (1) execute a ROLLBACK if base data is changed or (2) delete all rows from a table at the beginning of a test if an auxiliary table is used. BASETAB should be run whenever there is any question that the data in the HU schema may have been corrupted.

For a listing of base data used by the various schemas, refer to the appropriate dataload programs. The Interactive Direct SQL formats are the shortest and easiest to read.

APPENDIX C. TESTCASE columns (TESTNO, PROG, DESCR)

This is a listing of all the tests in the test suite. It shows the three columns from table TESTCASE: TESTNO (test number), PROG (program containing the test), and DESCR (a 50-character description).

APPENDIX D. TEd Change Files

APPENDIX D.1 is a listing of the downloaded file CHGPCO.TED. There is a similar file for each of the eleven test suite types and for the schema. APPENDIX D.2 is a listing of this file after editing by the tester to specify changes to the NIST Test Suite programs. Note that the tester has attempted to place each change in some category. This file is sent to the SQL testing laboratory as part of the prevalidation package of the SQL implementation to be tested. The laboratory examines each of the proposed changes and determines whether it will be considered a nonconformity. APPENDIX D.3 is an example of a counter-proposed file CHGPCO.TED, showing how the SQL testing laboratory interprets the proposed changes and how it is willing to prepare a validation report.

APPENDIX E. Sample Printout from Program Execution

This is a sample of what will print on standard output upon execution of any of the sample programs.

APPENDIX F. Sample Summary Reports

This is a sample printout of the three reports produced by REPORTZX: PROBLEMS, TOTALS, and TEST RESULTS. Your results will differ.

APPENDIX G. "SQL Flaggers" Examples

Considerable latitude is given to vendors in how to meet the SQL Flagger requirement of FIPS PUB 127-2 (Section 10.d). This appendix is only a suggestion of how a vendor may monitor SQL syntax for conformance to Entry SQL-92. Desirable features include: designation of the token which begins the syntax for the extension, identification of the name of the extension being used, and location of all extensions within each statement (as opposed to locating only the first).

APPENDIX H. Automated Reporting System Diagrams

APPENDIX H.1 shows the Table Definitions (CREATE TABLE statements) for the Reporting System. APPENDIX H.2 through APPENDIX H.4 are diagrams that are helpful in understanding the architecture of the automated reporting system. These diagrams are not included in the ASCII version of the SQL User's Guide in directory OTHER. They are stored as PostScript files in directory REPORT.

APPENDIX H.2, Test Reporting Structure, depicts a dataflow for the three programs in the reporting structure (REPORTA, REPORTB, and REPORTZX). This diagram is helpful when switching reporting options and/or restarting the reporting system. APPENDIX H.2 is PostScript file REPORTIN.PS.

APPENDIX H.3, SQL Testing Profiles, is a four-page diagram stored in four PostScript files:
PROFIPS.PS shows the FIPS SQL Levels
PROISO.PS shows the ISO/IEC 9075:1992 Levels
PROSIZE.PS shows the Sizing and Optionals Levels, and
PROXOPEN.PS shows X/Open Levels.

APPENDIX H.4, Reporting System Tables, diagrams the major tables defined in APPENDIX H.1. APPENDIX H.4 is useful in identifying referential constraints among these tables. It is PostScript file REPTABLS.PS.

APPENDIX I. Informational Interactive Currency Tests

These are detailed instructions on how to test the Interactive SQL interface for serialized transactions. Eight different concurrency challenges are created by executing files and typing SQL statements according to scripts.

14. DESIGN NOTES

After six releases of the test suite, many suggestions for improvement have been incorporated into the software. There are other good suggestions which we have not incorporated, due to either lack of resources or as a conscious decision. The purpose of this section is to answer questions about the structure of the test suite and to explain some of our design decisions.

Version 5.0 of the test suite is the first version where the test cases for many features preceded all known implementations of the features. Consequently, the new tests are written with many subtests and variations on the features under test. We assume implementors will be relying on our tests as regression tests (for which they are not really intended), so we have made an effort to make the tests more thorough. Also, there is considerably more printing to assist in debugging.

The following are comments about frequently asked questions which are of general interest.

1. Most of our early tests determine if an SQL feature passes or fails after checking only some of the expected results, rather than systematically verifying that all values returned (including SQLCODE) are identical to expected values. The latter approach requires considerably more coding. With Version 5.0 of the test suite, we have begun to check status return codes and retrieved data values more thoroughly. We feel the early approach will be successful in finding nonconformities; although, it may be less successful in finding bugs.

2. Many of our early tests base their pass/fail analysis on the returned value of SQLCODE. Although some of our tests do require an SQLCODE of 100 (no data) to pass, many tests accept an SQLCODE > 0 when the standard specifies that the value should be 100. In these circumstances our tests do not adhere strictly to the SQL standard; however, they do not penalize conforming implementations. The purpose of testing for SQLCODE > 0 is to prevent the case where an implementation fails tests designed to test features (other than SQLCODE return values) when that implementation does indeed support those features. If an implementation does not support the standard values for SQLCODE, then it will fail tests 8, 13, 18, 24, etc.

3. It would obviously be useful for debugging if SQLCODE return values were printed after every SQL statement. We suggest you use our editor to install (temporarily) a printout of SQLCODE after each SQL statement, if needed. New tests (starting with Version 5.0) print both SQLCODE and SQLSTATE consistently. This makes the screen display look like a trace and is less pleasing than the earlier style.

4. Testing security (access rules violations) is difficult. We recognize access control may be enforced differently across implementations. An implementation may reject a program at syntax-evaluation time or may reject an SQL statement at run time. An implementation may treat an unauthorized table or column as non-existent or as empty. Our tests expect that unauthorized users should not be able to modify the rows in the database, nor should unauthorized users be able to retrieve results based on actual data values.

5. Some vendors produce syntax errors for test cases where it can be determined at syntax evaluation time that a general rule will be violated at execution time. For example, inserting a literal NULL into a column which has a NOT NULL constraint can be determined to be an error at syntax evaluation time. The table TESTCASE contains a value "synvio_yes" in column T_NOTE for tests in this category to indicate that a syntax

violation is likely. This message prints on the PROBLEMS list when a test is missing. A vendor may pass these tests, by demonstrating appropriate error messages.

6. Many of the tests that change the data in the tables (via INSERT, DELETE, UPDATE) issue a ROLLBACK statement (after test evaluation) to restore the data to its original state and to make the test repeatable. If the implementation does not support ROLLBACK or if it has an autocommit feature turned ON then the data is left in a changed state, and the tests may not be reproducible. If the data is corrupted, the data load programs may be executed again to restore the data.

7. Most new test programs (for Transitional SQL features) create the tables and views needed by the test. At the end of the test, the tables and views are dropped in order to facilitate retesting. This approach makes the test more modular and easier to debug. This approach was not possible for Entry SQL tests, since schema manipulation is a Transitional SQL feature.

8. Many of the new tests, especially those for dynamic SQL, need to create long character strings. Rather than creating a long character string in the host language, we have used the SQL concatenation expression. This decision was made because our translation software is not able to process concatenation of C character arrays or continuation of character literals.

9. Except for the SQL Flagger tests and the privileges violations tests, our goal is to write programs which adhere strictly to the syntax of the standard. Several beta testers suggested that we test for violations of syntax; e.g., creating a duplicate table name or a referencing an undefined column. Unlike some other standards, SQL allows a conforming implementation to provide additional facilities or options not specified by the standard. The semantics of such syntax is implementation-defined. It is beyond the scope of our SQL test suite to evaluate an extension. Our only requirement is that extensions be explicitly identified when the SQL Flagger is "on."

10. The variable names in the test suite programs are short and consequently not very descriptive. For most programs, our data names are restricted to six characters so the programs can be translated into FORTRAN. With this restriction, programs do not have data names as descriptive as Ada, C, COBOL, and Pascal programmers would expect.

11. Some users noted that our tests do not consistently close each cursor that has been opened. According to the standard, a cursor that is opened may be closed explicitly by a CLOSE statement, or implicitly by a COMMIT or ROLLBACK. Our tests use all three methods.

12. SQLSTATE tests are difficult to code for conditions involving implementation-defined precision or other features. If the code (logic) in a program is unable to cause the error condition under test, then the program logic assigns a "pass" for that SQLSTATE test. This does not mean that the SQL implementation actually produces the SQLSTATE code.

It means that the SQL implementation did meet all the requirements of the test case. It also means that we need to "build a better mouse trap."

An example of this situation is the test for data exception - indicator overflow (SQLSTATE 22022). We create a CHAR(33000) column, hoping to overflow a 16-bit indicator variable on a SELECT. If the SQL implementation supports more precision in the indicator, then the overflow condition is not raised. Or, if the SQL implementation does not support CHAR(33000), which is larger than the FIPS 127-2 requirement of at least CHAR(240), then we cannot execute the test case and a "pass" is assigned. Other difficult-to-test SQLSTATEs are: error in assignment, invalid parameter value, numeric value out of range, and serialization failure. If you have a better test case for any of these difficult-to-test conditions, please donate it to us. We are always open to suggestions for improved testing strategies.

13. Many of the Version 3.0 programs carried forward into subsequent versions still contain references to the SQL-89 standard. We apologize, even though the research to obtain the correct reference is rather straightforward. The effort to update all programs and re-test them would have delayed release for several months. In retrospect, the references probably do not belong in the programs, since the SQL standard evolves rather rapidly.

    We have developed a documentation scheme that is intended to satisfy ISO requirements for test suites to be used internationally. We have documented new test cases in the ISO format in files DOCUCTS5.TXT and DOCU_V5.TXT as well as inside the programs themselves (as print statements and comments). Test cases from Version 4.0 are documented less rigorously in file DOCU_V4.TXT. File DOCUNIST.TXT provides a convenient list of all test cases developed by NIST for both Versions 5.0 and 6.0. As always, if users of the test suite have questions about references, or if they wish to challenge our interpretation of the standard, we welcome their calls and email.

15. ANNOTATED BIBLIOGRAPHY

The following is an informal listing of documents of interest to users of the SQL Test Suite and to federal agencies acquiring SQL implementations. Frequently requested items are listed first.

Validated Products List

- o accessible on the World Wide Web at URL address ftp://speckle.ncsl.nist.gov/vpl/sqlintro.htm
- o lists tested SQL products and publicly available Validation Summary Reports (VSRs).
- o contains testing information for the following Information Technology Standards: Programming Languages COBOL, Fortran, Ada, and C; Database Language SQL; Graphics; POSIX; and Computer Security

Miscellaneous documents/forms/reports available on NIST Web pages

- The table of contents page for the SQL Projects is URL address ftp://speckle.ncsl.nist.gov/sql-testing/contents_sql.htm
- SQL TESTING UPDATEs (testing "updates" and "status reports")
- SQL Validation Questionnaire
- SQL Processor Validation Procedures

FIPS PUB 127-2, Database Language SQL

- Federal Information Processing Standard Publication, issued by NIST (30 pages), dated June 2, 1993
- URL: http://www.itl.nist.gov/div897/pubs/fip127-2.htm
- packaged with the adopted specification ANSI X3.135-1992 (580 pages) when purchased from NTIS
- available from NTIS (703) 487-4650

Database Language - SQL

- published as ISO/IEC 9075:1992, Database Language - SQL
- except for a different Foreword, Introduction, and Normative references, this is identical to ANSI X3.135-1992
- available from ANSI international sales office (212) 642-4900

SQL Technical Corrigendum 2

- referred to as ANSI/ISO/IEC 9075 TC2, Database Language SQL - Technical Corrigendum 2
- SQL information bulletin to inform the public of responses to interpretation requests and errata against the SQL standards ISO/IEC 9075:1992 and ANSI X3.135-1992. It is a list of all formally approved corrections, including typos, missing rules, corrected rules, etc. from the originally published SQL-92.
- Technical Corrigendum 2 (TC2), adopted in 1995, supersedes Technical Corrigendum 1 (TC1), published in 1993.
- included with any new purchase of SQL-92, and can be obtained from ANSI by asking for document number JTC1/SC21 N10146.

Call Level Interface (CLI)

- published as ANSI/ISO/IEC 9075-3:1995 Database Language SQL - Part 3: Call Level Interface.
- evolved from the popular ODBC specification
- standardizes over 40 functions used as an application programming interface (API)
- is being extended to support handles for the management of large objects and to extend the facilities for metadata access and exception diagnostics (expected in 1999).
- available from ANSI.

Persistent Stored Modules (PSM)

- published as ANSI/ISO/IEC 9075-4:1996, Database Language SQL _ Part 4: Persistent Stored Modules.
- evolved from the popular stored procedure capabilities in may client-server products.

- o extends SQL to be a computationally complete, block structured programming language with support for functions, procedures, program variables, flow-of-control statements, and sophisticated exception handling and exception resolution.
- o available from ANSI

Remote Database Access (RDA)

- o published as ISO/IEC 9579-1 and 9579-2, Remote Database Access
- o Part 1: Generic Model, Service and Protocol
- o Part 2: SQL Specification
- o available from ANSI international sales office (212) 642-4900
- o ISO/IEC 9579-2:1997, RDA SQL Specialization Amendment, adds facilities to support all conformance levels of Database Language SQL, rather than just Entry SQL as specified in the original. It is published as a consolidated document, thereby replacing the original

FIPS PUB 193, SQL Environments

- o FIPS PUB 193, dated February 3, 1995
- o specifies SQL profiles that can be used to support integration of legacy databases and other non-SQL data repositories into an SQL environment.
- o a non-mandatory FIPS that may be invoked on a case-by-case basis subject to various database integration objectives.
- o URL: ftp://speckle.ncsl.nist.gov/isowg3/FIPSdocs/fips193.{ps | txt}

CAE Specification, Structured Query Language (SQL)

- o Common Applications Environment (CAE) specification produced by X/Open Company Ltd., August 1992. The CAE Specification describes X/Open as an independent, worldwide, open systems organization supported by most of the world's largest information systems suppliers, user organizations and software companies.
- o available from X/Open Company Ltd., U.K., XoSpecs@xopen.co.uk

16. ONGOING SQL STANDARDIZATION - SQL3

The standard specification of SQL is under continual development with draft documents at various stages of development available from your National Body representatives in the ANSI/ISO/IEC standardization process. In the United States, ANSI/X3 technical committee X3H2 (Database) is a very active contributor and the U.S. technical advisory group (TAG) to this process. Draft documents are all available from ANSI as JTC1/SC21 working documents and include the following:

SQL Part 1 - Framework: An overview document intended to explain the new Part structure for future SQL development. It will also describe procedures for claiming conformance to the base standard and to various optional levels and components. Publication expected in 1999.

SQL Part 2 - Foundation: The basic definition of the SQL language. A substantial, upward compatible extension of the language facilities specified in SQL-92, including triggers, assertions,

recursion, new data types for handling collections and large objects, as well as user-defined abstract data types (ADTs), type hierarchies, inheritance, polymorphism, and other facilities normally associated with object data management. However, object references are specified in Part 8 rather than in Part 2. This part of SQL will be a leveled specification, with nested levels analogous to those specified in SQL-92. Implementations will be able to claim conformance at a specific level. This document reached ISO/IEC Committee Draft (CD) status in 1996 with final adoption and publication expected by 1999.

SQL Part 3 - Call Level Interface: An upward compatible extension to the existing SQL/CLI standard, which is published as ANSI/ISO/IEC 9075-3:1995. This document is expected to reach Committee Draft (CD) status sometime in 1997.

SQL Part 4 - Persistent Stored Modules: An upward compatible extension to the existing SQL/PSM standard, which is published as ANSI/ISO/IEC 9075-4:1996. This document is expected to reach Committee Draft (CD) status sometime in 1997 or 1998.

SQL Part 5 - Bindings: Specification of Dynamic SQL and Embedded SQL interfaces to standard programming languages such as Ada, C, COBOL, Fortran, Mumps, Pascal, and PL/I. The basis of this document is taken from the Dynamic SQL and Embedded SQL sections of SQL-92, but new facilities have been added to accommodate handles and other new SQL data types in various programming languages. This document reached ISO/IEC Committee Draft (CD) status in 1996 with final adoption and publication expected by 1999.

SQL Part 6 - XA Specialization: Originally approved as a project to be developed, if needed, to supplement the ISO/IEC project on Distributed Transaction Processing - The XA interface. XA is a collection of popular distributed transaction processing interfaces developed by X/Open. With successful publication of an ISO/IEC XA specification in 1996 (see document JTC1/SC21 N10133), it is not clear if this SQL specification is still necessary. The project was not very active in 1996 with very little technical content in the working draft base document.

SQL Part 7 - Temporal: The intent of this project is to add support for temporal data management, such as valid time and transaction time, to the SQL language. The TSQL2 Language Specification prevalent in the academic literature during 1994 to 1995 is the basis for much of the new development. This document is expected to reach Committee Draft (CD) status sometime in 1998.

SQL Part 8 - Extended Objects: The intent of this project is to specify how object identity is to be handled in SQL. The base document specifies named row types, references to instances of named row types, and reference and dereference operators, as well as other related facilities for object data management. This document reached ISO/IEC Committee Draft (CD) status in 1996 with final adoption and publication expected by 1999.

SQL Part 9 - Virtual Tables: The intent of this newly adopted project is to specify virtual tables as the mechanism for allowing SQL language access to legacy files and data repositories. The specification will provide tools that a database administrator can use to simplify the process of

making such legacy data available to standard conforming SQL applications. This document is expected to reach Committee Draft (CD) status by 1999.

ANSI/ISO/IEC 9075 TC3, Database Language SQL - DRAFT Technical Corrigendum 3: This draft corrigendum to all existing published parts of Database Language SQL includes corrections informally approved by the SQL development group since publication of TC2 in 1995, including corrections to SQL/CLI-95 and to SQL/PSM-96. It has not yet been submitted to ISO/IEC for formal adoption and publication, although such submission is expected sometime during 1997. The draft document is available from any active participant in the SQL standardization process.

VAX/VMS using Rdb
DCL Listing for Embedded C

```
$ !  Database has already been created by DBADMIN with commands:
$ !  create database filename NIST60
$ !  multischema is on  protection is ansi;
$ !  grant select, insert, delete, update, references, show,
$ !  createtab on database alias rdb$dbhandle to HU,CUGINI,...
$ !
$ ! The following three commands really belong in file login.com:
$ ! Designate file [DBADMIN]NIST60.RDB as the default database.
$ ASSIGN  [DBADMIN]NIST60.RDB  SQL$DATABASE
$ ! Set up convenient reference to precompiler
$ SQLPRE :== $SYS$SYSTEM:SQL$PRE
$ ! Set up convenient reference to interactive SQL processor
$ SQL :== $SQL$

$ ! --- Tester has logged in as user HU.
$ ! Create schema interactively in RDBMS.
$ ! SCHEMA1.STD is being used.
$ ! TED is used to install changes
$ !    and to rename file to extension "sql".
$ ted  -t upd600.ted   -t chgall.ted   -o schema1.sql  schema1.std
$ SQL
SQL> SET VERIFY
SQL> @schema1.sql
SQL> ;
SQL> EXIT

$ ! PROCEDURE TO CREATE AND RUN AN EXECUTABLE MODULE:
$ ! Use TED to remove call to subroutine authid, to install
$ !    other changes, and to rename file to extension "sc".
$ ! Pre-process, compile, link and execute embedded C routine.
$ ted   -t upd600.ted  -t chgall.ted   -o dml001.sc  dml001.pc
$ SQLPRE/CC/sqlo=(flag,cons=on,ansi_auth) dml001
$ LINK    dml001,SYS$LIBRARY:SQL$USER/LIB, -
          SYS$LIBRARY:VAXCRTLG.OLB/LIB, SYS$LIBRARY/VAXCRTL/LIB
$ RUN     dml001
$ ! Delete intermediate files, saving executable dml001.exe:
$ DELETE  dml001.sc;*
$ DELETE  dml001.c;*
$ DELETE  dml001.obj;*
$ DELETE  dml001.lis;*
====================== END OF APPENDIX ============================
```

VAX/VMS using Oracle
DCL Listing for Embedded FORTRAN

```
$ ! create schema interactively in Oracle:
$ ! DBA has already authorized user HU with password HU.
$ ! SCHEMA1.NC is being used with Oracle, Version 6.
$ ! TED is used to install changes
$ !    and to rename file to extension "sql".
$ ted  -t upd600.ted  -t chgsch.ted   -o schema1.sql  schema1.nc
$ sqlplus hu/hu
SQL> set echo on
SQL> @schema1.sql
SQL> exit
```

```
$ ! Use TED to install changes, then
$ ! pre-process, compile, link and execute embedded FORTRAN
routine:
$ ! Note that subroutine AUTHID has been pre-processed and compiled
$ @compile_pfo  dml001
```

COMPILE_PFO.COM:

```
$ted   -t upd600.ted  -t chgpfo.ted   'p1.pfo
$proc  iname='p1 host=fortran  include=sys$oracle:
$for   'p1
$@dra4:[oracle]loutl 'p1 'p1,authid,sys$oracle:sqllib/lib map$:s
$run   'p1
```

========== END OF APPENDIX ==========

UNIX Command Language for Unify
Embedded C

```
$ # set environment:
$ # point to release library directory
$ UNIFY=/usr/unify/lib
$ # point to the directory that contains the database
$ DBPATH=/usr/sqltest/db
$ #add unify release binary directory to the path
$ PATH=$PATH:/usr/unify/bin
$ export UNIFY DBPATH


$ # create schema:
$ # [note that database has already been created by DBA]
$ # TED is used to install any changes.
$ ted     -t upd600.ted  -t chgsch.ted schema.std
$ SQL     schema.std


$ # precompile, compile, link and go for program dml001:
$ # --- tester logs in as user whose default authid is HU ---

$ #     Use TED to install changes and to
$ #     rename program to desired extension
$ TED   -t upd600.ted  -t chgpc.ted   -o dml001.ec    dml001.pc
$ #     precompile dml001
$ EPP   dml001.ec
$ #     compile dml001
$ ucc   -c -I$UNIFY/.. dml001.c
$ #     link dml001
$ sqla.ld dml001 dml001.o
$ #     run dml001
$ dml001
```

========== END OF APPENDIX ==========

Base Data for Primary Test Tables

STAFF

EMPNUM CHAR(3)	EMPNAME CHAR(20)	GRADE DEC(4)	CITY CHAR(15)
E1	Alice	12	Deale
E2	Betty	10	Vienna
E3	Carmen	13	Vienna
E4	Don	12	Deale
E5	Ed	13	Akron

PROJ

PNUM CHAR(3)	PNAME CHAR(20)	PTYPE CHAR(6)	BUDGET DEC(9)	CITY CHAR(15)
P1	MXSS	Design	10000	Deale
P2	CALM	Code	30000	Vienna
P3	SDP	Test	30000	Tampa
P4	SDP	Design	20000	Deale
P5	IRM	Test	10000	Vienna
P6	PAYR	Design	50000	Deale

WORKS

EMPNUM CHAR(3)	PNUM CHAR(3)	HOURS DEC(5)
E1	P1	40
E1	P2	20
E1	P3	80
E1	P4	20
E1	P5	12
E1	P6	12
E2	P1	40
E2	P2	80
E3	P2	20
E4	P2	20
E4	P4	40
E4	P5	80

TESTCASE Columns: TESTNO, PROG, DESCR

Test Number	Program	Description
0001	dml001	CURSOR with ORDER BY DESC
0002	dml001	CURSOR with ORDER BY integer ASC
0003	dml001	CURSOR with ORDER BY DESC integer, named column
0004	dml001	CURSOR with UNION: ORDER by integer DESC
0005	dml001	CURSOR with UNION ALL
0006	dml002	Error for second consecutive OPEN without CLOSE
0007	dml003	Error for second consecutive CLOSE
0008	dml004	SQLCODE = 100: FETCH on empty table
0009	dml004	FETCH NULL value, get indicator = -1
0010	dml004	FETCH truncated CHAR column with indicator
0011	dml005	FIPS sizing - DECIMAL(15)
0012	dml006	SQLCODE < 0: DELETE CURRENT at end-of-data
0013	dml006	DELETE CURRENT row twice
0014	dml007	UPDATE CURRENT
0015	dml007	UPDATE CURRENT - view with check and unique
0016	dml008	SQLCODE < 0: 2 rows selected by single-row SELECT
0017	dml008	SELECT DISTINCT
0018	dml008	SQLCODE = 100: SELECT with no data
0019	dml008	SQLCODE = 0: SELECT with data
0020	dml008	SELECT NULL value, get indicator = -1
0021	dml008	SELECT CHAR(M) col. into short var., get indic = M
0022	dml009	INSERT (column list) VALUES (literals and NULL)
0023	dml009	SQLCODE < 0 if left-truncate DEC (>= col.def.)
0024	dml009	SQLCODE = 100: INSERT query spec. is empty
0025	dml009	SQLCODE = 0: INSERT query spec. is not empty
0026	dml009	INSERT into view, check option + unique violations
0027	dml010	INSERT short string into long column-space padding
0028	dml010	INSERT string that exactly fits in column
0031	dml010	INSERT (column list) VALUES (variables and NULL)
0033	dml011	UPDATE view without WHERE clause - all rows
0034	dml011	UPDATE table with SET column in WHERE clause
0035	dml011	UPDATE with correlated subquery in WHERE clause
0036	dml011	UPDATE view globally with check option violation
0037	dml012	DELETE without WHERE clause - all rows
0038	dml012	DELETE with correlated subquery in WHERE clause
0039	dml013	COUNT DISTINCT function
0040	dml013	SUM function with WHERE clause
0041	dml013	MAX function in subquery

```
0042  dml013  MIN function in subquery
0043  dml013  AVG function
0044  dml013  AVG function: empty result NULL value
0045  dml014  BETWEEN predicate
0046  dml014  NOT BETWEEN predicate
0047  dml014  IN predicate with subquery
0048  dml014  NOT IN predicate with subquery
0049  dml014  IN predicate with value list
0050  dml014  LIKE predicate with % (percent)
0051  dml014  LIKE predicate with _ (underscore)
0052  dml014  LIKE predicate with ESCAPE character
0053  dml014  NOT LIKE predicate
0054  dml014  IS NULL predicate
0055  dml014  NOT NULL predicate
0056  dml014  NOT EXISTS predicate
0057  dml014  ALL quantifier
0058  dml014  SOME quantifier
0059  dml014  ANY quantifier
0060  dml015  COMMIT WORK closes cursors
0061  dml015  COMMIT WORK keeps changes to database
0062  dml015  ROLLBACK WORK cancels changes to database
0063  dml015  ROLLBACK WORK closes cursors
0064  dml016  SELECT USER
0065  dml016  SELECT CHAR literal, term with numeric literal
0066  dml016  SELECT numeric literal
0067  dml017  WHENEVER NOT FOUND(SQLCODE=100), GOTO and CONTINUE
0068  dml017  WHENEVER SQLERROR (SQLCODE< 0), GOTO and CONTINUE
0069  dml018  HAVING COUNT with WHERE, GROUP BY
0070  dml018  HAVING COUNT with GROUP BY
0071  dml018  HAVING MIN, MAX with GROUP BY 3 columns
0072  dml018  Nested HAVING IN, with no outer reference
0073  dml018  HAVING MIN with no GROUP BY
0074  dml019  GROUP BY column: SELECT column, SUM
0075  dml019  GROUP BY clause
0076  dml019  GROUP BY 2 columns
0077  dml019  GROUP BY all columns, with SELECT *
0078  dml019  GROUP BY 3 columns, SELECT 2 of them
0079  dml019  GROUP BY NULL value
0080  dml020  Simple 2-table join
0081  dml020  Simple 2-table join with 1-table predicate
0082  dml020  Join 3 tables
0083  dml020  Join a table with itself
0084  dml021  Data type CHAR(20)
0085  dml021  Data type CHARACTER(20)
0086  dml021  Data type INTEGER
0087  dml021  Data type INT
```

```
0088  dml034  Data type REAL
0089  dml021  Data type SMALLINT
0090  dml034  Data type DOUBLE PRECISION
0091  dml034  Data type FLOAT
0092  dml034  Data type FLOAT(32)
0093  dml034  Data type NUMERIC(13,6)
0094  dml034  Data type DECIMAL(13,6)
0095  dml034  Data type DEC(13,6)
0096  dml022  Subquery with MAX in < comparison predicate
0097  dml022  Subquery with AVG - 1 in <= comparison predicate
0098  dml022  IN predicate with simple subquery
0099  dml022  Nested IN predicate - 2 levels
0100  dml022  Nested IN predicate - 6 levels
0101  dml022  Quantified predicate <= ALL with AVG and GROUP BY
0102  dml022  Nested NOT EXISTS with corr. subqueries, DISTINCT
0103  dml023  Subquery with = comparison predicate
0104  dml023  SQLCODE < 0: subquery with more than 1 value
0105  dml023  Subquery in comparison predicate is empty
0106  dml023  Comparison predicate <> (not equal)
0107  dml023  Short string logically blank-padded in = pred.
0108  dml024  Search condition true OR NOT (true)
0109  dml024  Search condition true AND NOT (true)
0110  dml024  Search condition unknown OR NOT (unknown)
0111  dml024  Search condition unknown AND NOT (unknown)
0112  dml024  Search condition unknown AND true
0113  dml024  Search condition unknown OR true
0114  dml025  Set functions without GROUP BY returns 1 row
0115  dml025  GROUP BY 0 groups returns 0 rows: SEL col.,AVG...
0116  dml025  GROUP BY 0 groups returns 0 rows: SELECT set fnc.
0117  dml025  Set functions with GROUP BY several groups
0118  dml026  Monadic arithmetic operator + (plus)
0119  dml026  Monadic arithmetic operator - (minus)
0120  dml026  Value expression with NULL primary yields NULL
0121  dml026  Dyadic arithmetic operators +, -, *, /
0122  dml026  SQLCODE < 0: divisor shall not be zero
0123  dml026  Evaluation order of expression
0124  dml027  UPDATE UNIQUE column (key=key+1) interim conflict
0125  dml027  UPDATE UNIQUE column (key+1): no interim conflict
0126  dml028  CLOSE, OPEN, FETCH returns first row
0127  dml028  OPEN 2 cursors at same time
0128  dml028  OPEN 3 cursors at same time
0129  dml029  Double quote mark (') in character string literal
0130  dml029  Approximate numeric literal <mantissa>E<exponent>
0131  dml029  Approximate numeric literal with neg. exponent
0135  dml033  Upper and lower case letters are distinct
0137  sdl001  CREATE SCHEMA
```

```
0138  sdl002  GRANT ALL PRIVILEGES TO PUBLIC (SELECT, INSERT)
0139  sdl003  GRANT ALL PRIVILEGES TO PUBLIC (SELECT, UPDATE)
0140  sdl004  Priv.violation: GRANT SELECT TO PUBLIC, no INSERT
0141  sdl005  GRANT SELECT and UPDATE to individual
0142  sdl006  GRANT SELECT and UPDATE WITH GRANT OPTION
0143  sdl007  GRANT SELECT and UPDATE on VIEW
0144  sdl008  Priv.violation: colunm not in UPDATE column list
0145  sdl009  Fully qualified column specification
0146  sdl010  GRANT SELECT, DELETE, INSERT
0147  sdl011  CREATE SCHEMA
0148  sdl012  CREATE TABLE with NOT NULL
0149  sdl013  CREATE TABLE with NOT NULL UNIQUE
0150  sdl014  CREATE TABLE with UNIQUE(...): INSERT VALUES
0151  sdl015  CREATE VIEW
0152  sdl016  CREATE VIEW with CHECK OPTION
0153  sdl017  CREATE VIEW joining 3 tables
0154  sdl018  Schema def.- same table name for different schemas
0155  sdl019  CREATE TABLE with UNIQUE(...): INSERT via SELECT
0156  sdl020  Tables are multi-sets, dup. INSERTed via subquery
0157  dml035  CURSOR with ORDER BY approximate numeric
0158  dml001  CURSOR with UNION and NOT EXISTS subquery
0159  dml001  CURSOR with 2 UNIONs, ORDER BY 2 integers
0160  dml001  CURSOR with UNION, UNION ALL and ORDER BY
0161  dml004  SQLCODE < 0: FETCH NULL value without indicator
0162  dml004  FETCH NULL value with INDICATOR syntax
0163  dml006  SQLCODE < 0: DELETE CURRENT without FETCH
0164  dml008  Default of SELECT is ALL, not DISTINCT
0165  dml008  Truncate CHAR column SELECTed into shorter var.
0166  dml036  INSERT NULL value with indicator = -1
0167  dml013  SUM ALL function
0168  dml013  SUM function
0169  dml013  COUNT (*) function
0170  dml013  SUM DISTINCT function with WHERE clause
0171  dml013  SUM (column) + literal
0172  dml016  SELECT USER into short variable
0173  dml021  Data type CHAR
0174  dml021  Data type CHARACTER
0175  dml021  Data type NUMERIC
0176  dml021  Data type NUMERIC(9): SELECT *
0177  dml021  Data type NUMERIC(9): SELECT column
0178  dml021  Data type DECIMAL
0179  dml021  Data type DECIMAL(8)
0180  dml023  NULLs sort together in ORDER BY
0181  dml023  NULLs are equal for DISTINCT
0182  dml029  Approx. num. literal with neg. mantissa and exp.
0183  ccc001  C language embedded host identifiers
```

0184	ccc002	C language NULL terminator
0185	cob001	COBOL - embedded host identifiers
0186	cob002	COBOL - CHAR(80)
0187	cob002	COBOL - CHAR(132)
0188	cob002	COBOL - CHAR(240)
0189	dml078	OPTIONAL - CHAR(256)
0190	dml078	OPTIONAL - CHAR(512)
0191	dml078	OPTIONAL - CHAR(1024)
0192	ccc003	C language continuation of SQL char. literal
0193	ccc003	C language comments within embedded SQL statement
0194	ccc004	C language EXTERN storage class
0195	ccc005	C language STATIC storage class
0196	cob004	COBOL PIC S9(12) precision test
0197	cob004	COBOL PIC S9(18) precision test
0198	ccc006	C language AUTO storage class
0199	sdl021	Priv.violation: GRANT SELECT to PUBLIC, no DELETE
0200	sdl022	Priv.violation: GRANT INSERT to indiv., no SELECT
0201	sdl023	Priv.violation: GRANT without GRANT OPTION
0202	dml037	Host variable names same as column names
0203	sdl024	CREATE VIEW on VIEW
0204	sdl025	Updatable VIEW with AND, OR in CHECK clause
0205	dml038	Cartesian product is produced without WHERE
0206	cob005	COBOL - continuation of SQL char. literal
0207	cob006	COBOL - comments within embedded SQL statement
0208	dml039	LIKE predicate with underscore is case sensitive
0209	dml040	Join 2 tables from different schemas
0210	cob007	COBOL - PIC S9(1) syntax
0211	cob007	COBOL - PIC S9(7) syntax
0212	dml041	Enforcement of CHECK clauses in nested views
0213	dml042	FIPS sizing - 100 columns in a row
0214	dml043	FIPS sizing - 2000 byte-row
0215	dml044	FIPS sizing - 6 columns in a UNIQUE specification
0216	dml044	FIPS sizing - 120 bytes in a UNIQUE specification
0217	for001	FORTRAN - continuation of SQL character literal
0218	dml045	FIPS sizing - 6 columns in GROUP BY
0219	dml045	FIPS sizing - 120 bytes in GROUP BY
0220	dml046	FIPS sizing - 6 columns in ORDER BY
0221	dml046	FIPS sizing - 120 bytes in ORDER BY
0222	dml047	FIPS sizing - CHARACTER(240)
0223	for001	FORTRAN - comments within embedded SQL statements
0224	dml048	FIPS sizing - 10 cursors open at once
0225	dml049	FIPS sizing - 10 tables in FROM clause
0226	dml050	FIPS sizing - 10 tables in nested SQL statements
0227	dml051	BETWEEN predicate with character string values
0228	dml051	NOT BETWEEN predicate with character string value
0229	dml052	LIKE predicate is case sensitive

```
0230  mpa001  Transactions serializable: assign sequential key
0231  mpa002  Transactions serializable: SELECT/UPDATE(replace)
0232  mpa003  Transactions serializable: UPDATE with arithmetic
0233  dml053  Tables are multi-sets: duplicate via INSERT VALUE
0234  dml037  SQL comments (double hyphen) in SQL statements
0235  cob008  COBOL - exact numeric types S9(i)V9(k)
0236  cob008  COBOL: SQLCODE < 0: exception losing signif. digit
0237  sdl026  Identifier length 18
0238  pas002  Pascal - comments within embedded SQL statements
0239  pas001  Pascal - embedded host identifiers
0240  dml054  Updatable CURSOR with ALL, IN, BETWEEN
0241  dml054  Updatable CURSOR with LIKE, NULL, >, =, <
0242  dml054  Updatable CURSOR with view, correlation name, NOT
0243  dml055  FIPS sizing - precision of SMALLINT >= 4
0244  dml055  FIPS sizing - precision of INTEGER >= 9
0245  dml055  FIPS sizing - precision of DECIMAL >= 15
0246  dml056  FIPS sizing - 100 values in INSERT
0247  dml056  FIPS sizing - 20 values in update SET clause
0248  dml057  FIPS sizing - binary precision of FLOAT >= 20
0249  dml057  FIPS sizing - binary precision of REAL >= 20
0250  dml057  FIPS sizing - binary precision of DOUBLE >= 30
0251  dml058  COMMIT keeps changes of current transaction
0252  dml058  ROLLBACK cancels changes of current transaction
0253  dml058  TEST0124 workaround (key = key+1)
0254  dml058  Column name in SET clause
0255  dml058  Key word USER for INSERT, UPDATE
0256  dml058  Key word USER in WHERE clause
0257  dml059  SELECT MAX, MIN (COL1 + or - COL2)
0258  dml059  SELECT SUM (:var * COL1 * COL2)
0259  dml059  SOME, ANY in GROUP BY, HAVING clause
0260  dml059  EXISTS in GROUP BY, HAVING
0261  dml060  WHERE (:var * (COL1 - COL2)) BETWEEN
0262  dml060  WHERE clause with computation, ANY/ALL subqueries
0263  dml060  Computed column in ORDER BY
0264  dml059  WHERE, HAVING without GROUP BY
0265  dml060  UPDATE : positioned - view with check option
0266  dml060  UPDATE : positioned - UNIQUE violation under view
0267  dml060  UPDATE compound key, interim uniqueness conflicts
0268  mpa004  Transactions serializable: deadlock management
0269  dml061  BETWEEN value expressions in wrong order
0270  dml061  BETWEEN approximate and exact numeric values
0271  dml061  COUNT(*) with Cartesian product subset
0272  dml061  Statement rollback for integrity violation
0273  dml061  SUM, MAX, MIN = NULL (not 0) for empty arguments
0274  dml062  COMMIT and ROLLBACK across schemas
0275  dml062  COMMIT and ROLLBACK of multiple cursors
```

0276	dml062	View across schemas
0277	dml061	Computation with NULL value specification
0278	dml061	IN value list with USER, literal,variable spec.
0279	dml062	IN is a 3-valued predicate, EXISTS is 2-valued
0280	dml063	SQL key words used as embedded host identifiers
0281	dml064	Updatable VIEW with ALL, IN, BETWEEN
0282	dml064	Updatable VIEW with LIKE, NULL, >, =, <
0283	dml064	Updatable VIEW with view, correlation name, NOT
0284	dml065	INSERT,SELECT character strings with blanks
0285	dml065	INSERT,SELECT integers with various formats
0286	dml066	Compatibility of structures and host variables
0287	dml066	Compatibility of arrays and host structures
0288	dml067	Embedded - multiple identifiers in 1 declaration
0289	dml067	Embedded - multiple declare sections
0290	ccc007	C language common placement of SQL statements
0291	ccc008	C language SQL statements in functions
0292	cob009	COBOL - VALUE IS initialization
0293	cob009	COBOL - placement of SQL statements
0294	pas003	Pascal - placement of SQL statements
0295	for002	FORTRAN - placement of SQL statements
0296	flg005	FIPS Flagger - vendor provided character function
0297	flg005	FIPS Flagger - vendor provided integer function
0298	pas004	Pascal - SQL statements in functions
0299	flg006	FIPS Flagger - identifier length > 18
0300	cdr001	DEFAULT value literal + number in a table
0301	cdr001	DEFAULT value USER in a table
0302	cdr002	CHECK <comp. predicate> in <tab. cons.>, insert
0303	cdr002	CHECK <comp. predicate> in <col. cons.>, insert
0304	cdr002	CHECK <between predicate> in <tab. cons.>, insert
0305	cdr002	CHECK <null predicate> in <tab. cons.>, insert
0306	cdr003	CHECK X IS NOT NULL, NOT X IS NULL are equivalent
0307	cdr003	CHECK <like predicate> in <tab. cons>, insert
0308	cdr003	CHECK <in predicate> in <tab. cons.>, insert
0309	cdr004	CHECK combination predicates in <tab. cons.>
0310	cdr004	CHECK X NOT IN, NOT X IN equivalent, insert
0311	cdr004	CHECK NOT NULL in col.cons., insert, null explicit
0312	cdr004	CHECK NOT NULL in col.cons., insert, null implicit
0313	cdr005	CHECK <comp. predicate> in <tab. cons.>, update
0314	cdr005	CHECK <comp. predicate> in <col. cons.>, update
0315	cdr005	CHECK <between predicate> in <tab. cons.>, update
0316	cdr006	CHECK <null predicate> in <tab. cons.>, update
0317	cdr006	CHECK X IS NOT NULL, NOT X IS NULL same, by update
0318	cdr006	CHECK <like predicate> in <tab. cons.>, update
0319	cdr007	CHECK <in predicate> in <tab. cons.>, update
0320	cdr007	CHECK combination pred. in <tab. cons.>, update
0321	cdr007	CHECK if X NOT LIKE/IN, NOT X LIKE/IN same, update

```
0322  cdr007  CHECK <null predicate> in <col. cons>, update
0323  cdr008  (2 pr.,1 son),both P.K e, F.K e,insert another F.K
0324  cdr008  (2 pr.,1 son),1 P.K exist,another not. insert F.K
0325  cdr008  (2 pr.,1 son),both P.K e, F.K e, delete 1 P.K
0326  cdr008  (2 pr.,1 son),P.K e, no F.K, modify P.K
0327  cdr009  (2 pr.,1 son),check PRIMARY KEY unique via insert
0328  cdr009  (2 pr.,1 son),F.K exist,modify P.K
0329  cdr009  (2 pr.,1 son),check PRIMARY KEY unique via modify
0330  cdr009  (2 pr.,1 son),modify F.K to no P.K corr.
0331  cdr009  (2 pr.,1 son),modify F.K to P.K corr. value
0332  cdr010  (self ref.), P.K exist, insert a F.K
0333  cdr010  (self ref.), delete P.K but F.K exist.
0334  cdr010  (self ref.), update P.K, no corr. F.K
0335  cdr011  (self ref.), insert a F.K but no corr. P.K
0336  cdr011  (self ref.), update P.K, but corr. F.K e.
0337  cdr011  (self ref.), update P.K, check P.K unique via var.
0338  cdr011  (self ref.), update F.K and no corr. P.K
0339  cdr011  (self ref.), update F.K and corr. P.K exist
0340  cdr012  (ref. each other), insert F.K and corr. P.K e
0341  cdr012  (ref. each other), delete P.K but corr. F.K e
0342  cdr012  (ref. each other), update P.K and no corr. F.K
0343  cdr013  (ref. each other), update P.K and corr. F.K e
0344  cdr013  (ref. each other), update F.K to no corr. P.K
0345  cdr013  (ref. each other), update F.K to corr. P.K e
0346  cdr013  (ref. each other), insert F.K and no corr. P.K
0347  cdr014  FIPS sz. (comb.keys=6), P.K unique,insert
0348  cdr014  FIPS sz. (comb.keys=6), insert F.K + no corr. P.K
0349  cdr015  FIPS sz. (comb.keys=6), delete P.K + corr. F.K e
0350  cdr015  FIPS sz. (comb.keys=6), update P.K + no corr. F.K
0351  cdr015  FIPS sz. (comb.keys=6), update P.K + corr. P.K e
0352  cdr016  FIPS sz. (comb.keys=6), P.K unique, update
0353  cdr016  FIPS sz. (comb.keys=6), update F.K to no corr. P.K
0354  cdr016  FIPS sz. (comb.keys=6), update F.K to corr. P.K e
0355  cdr017  FIPS sz. (1 pr.,6 son), insert F.K + no corr. P.K
0356  cdr017  FIPS sz. (1 pr.,6 son), delete P.K + corr. F.K e
0357  cdr017  FIPS sz. (1 pr.,6 son), update P.K but corr. F.K e
0358  cdr017  FIPS sz. (1 pr.,6 son), check key unique, update
0359  cdr017  FIPS sz. (1 pr.,6 son), update F.K to no corr. P.K
0360  cdr018  FIPS sz. (6 pr.,1 son), insert F.K, without P.K
0361  cdr018  FIPS sz. (6 pr.,1 son), delete P.K, but corr.F.K e
0362  cdr018  FIPS sz. (6 pr.,1 son), update P.K, but corr.F.K e
0363  cdr018  FIPS sz. (6 pr.,1 son), check key unique ,update
0364  cdr018  FIPS sz. (6 pr.,1 son), update F.K to no corr. P.K
0365  cdr019  (3-level schema), check insert F.K
0366  cdr019  (3-level schema), check delete P.K
0367  cdr019  (3-level schema), update mid. tab. check P.K + F.K
```

```
0368  cdr019  (3-level schema), check update P.K
0369  cdr020  update P. K, set F1=F1+1, interim violation
0370  cdr020  update F. K, set F1=F1+1, interim violation
0371  cdr020  update self-ref table, interim violation
0372  cdr020  delete self-ref table, interim violation
0373  cdr003  insert with embeded var.+ indic. var. CHECK clause
0374  cdr003  computation in update, CHECK clause
0375  cdr017  ref. integrity with computation
0376  cdr017  ref. integrity with join
0377  cdr001  DEFAULT value with explicit NULL
0378  cdr021  (ref. acr. sch.) delete P.K and corr. F.K e.
0379  cdr021  (ref. acr. sch.) update P.K and corr. F.K e.
0380  cdr022  (ref. acr. sch.) insert F.K and no corr. P.K
0381  cdr022  (ref. acr. sch.) update F.K to no P.K corr.
0382  cdr023  (ref. acr. sch.) with GRANT OPTION, insert
0383  cdr023  Priv.violation: GRANT without GRANT OPTION
0384  cdr023  Priv.violation: SELECT, but not REFERENCES
0385  cdr024  character default column values
0386  cdr024  exact numeric default column values
0387  cdr024  approximate numeric default column values
0388  cdr024  FIPS sz. default column values
0389  dml068  95-char ASCII collating sequence (PL vs. SQL)
0390  dml072  Short char column value blank-padded in larger var
0391  sdl027  Correlation names used in self-join of view
0392  for003  FORTRAN - placement of additional SQL statements
0393  dml073  SUM, MAX on Cartesian product
0394  dml073  AVG, MIN on joined table with WHERE
0395  dml073  SUM, MIN on joined table with GROUP BY
0396  dml073  SUM, MIN on joined table with WHERE,GROUP,HAVING
0397  sdl028  Grouped view
0398  ccc011  Embedded C initial value
0399  ccc009  C language storage class and class modifier comb.
0401  sdl027  View with computed columns (degrees F to C)
0402  cdr025  Computed GROUP BY view over referencing tables
0403  cdr025  View on computed GROUP BY view with joins
0404  dml069  2 FETCHes (different target types) on same cursor
0405  dml069  2 cursors open from different schemas (coded join)
0406  dml069  Subquery from different schema
0407  dml069  SELECT INTO :XX ... WHERE :XX =
0408  dml069  UPDATE references column value BEFORE update
0409  dml070  Effective outer join--join with two cursors
0410  dml076  NULL value in OPEN CURSOR
0411  dml070  Effective set difference
0412  dml070  Effective set intersection
0413  cdr025  Computed SELECT on computed VIEW
0414  dml071  WHENEVER NOT FOUND, multiple settings
```

0415	dml071	WHENEVER SQLERROR, multiple settings
0416	dml071	WHENEVER NOTFOUND overlaps WHENEVER SQLERROR
0417	dml073	Cartesian product GROUP BY 2 columns with NULLs
0418	dml073	AVG,SUM,COUNT on Cartesian product with NULLs
0419	dml073	SUM, MAX, MIN on joined table view
0420	sdl028	View with multiple SELECT of same column
0421	dml074	Module language constants and expressions
0422	dml074	Module language order of SQLCODE (not first)
0423	dml074	Module language multiple SQLCODE parameters
0424	ada001	Ada embedded host identifiers
0425	ada002	Ada comments
0426	ada002	Ada initial value
0427	ada003	Ada common placement of SQL statements
0428	ada004	Ada placement of SQL statements
0430	ada005	Ada unqualified type spec - without SQL_STANDARD
0431	dml075	Redundant rows in IN subquery
0432	dml075	Unknowns in subquery of ALL, SOME, ANY
0433	dml075	Empty subquery of ALL, SOME, ANY
0434	dml075	GROUP BY with HAVING EXISTS-correlated set fnc
0435	dml076	Host variables in UPDATE WHERE CURRENT
0436	dml076	NULL values for various SQL data types
0437	dml076	NULL values for various host variable types
0438	cdr026	(partial-NULL F.K) F.K INSERT supported
0439	cdr026	(partial-NULL F.K) F.K UPDATE supported
0440	cdr026	(partial-NULL F.K) no restrict P.K update/delete
0441	dml076	NULL value for various predicates
0442	dml075	DISTINCT with GROUP BY, HAVING
0443	dml077	VIEW with check option rejects unknown (NULL)
0444	dml077	Updatable cursor, modify value selected on
0445	dml077	Values not assigned to targets for SQLCODE=100
0446	cdr027	Table CHECK constraint allows unknown (NULL)
0447	cdr027	NULLs with check constraint and check option
0448	cdr027	PRIMARY KEY implies UNIQUE
0449	cdr027	Constraint definition is case sensitive
0450	cdr027	Referential integrity is case sensitive
0451	dml079	UNIQUEness is case sensitive
0452	dml079	Order of precedence, left-to-right in UNION [ALL]
0453	dml079	NULL with empty subquery of ALL, SOME, ANY
0454	flg008	SELECT nonGROUP column in GROUP BY
0455	flg009	Relaxed union compatability rules for columns
0456	ada006	Module language Ada subtype enforcement,name assoc
0457	mpa005	Transactions serializable: phantom read
0458	sdl029	Priv.violation: GRANT only SELECT to individual
0459	sdl029	Priv.violation: GRANT only INSERT to individual
0460	sdl029	Priv.violation: GRANT only UPDATE to individual
0461	sdl029	Priv.violation: GRANT only DELETE to individual

```
0462  dml080  SQLCODE 100: DELETE with no data
0463  dml080  SQLCODE 100: UPDATE with no data
0464  sdl030  Priv.violation: GRANT only SELECT to PUBLIC
0465  sdl030  Priv.violation: GRANT only INSERT to PUBLIC
0466  sdl030  Priv.violation: GRANT only UPDATE to PUBLIC
0467  sdl030  Priv.violation: GRANT only DELETE to PUBLIC
0468  sdl031  Priv.violation: individual without any privileges
0469  sdl031  GRANT ALL PRIVILEGES to individual
0470  sdl031  GRANT ALL PRIVILEGES to public
0471  sdl031  Priv.violation: GRANT privilege not grantable
0472  sdl032  Priv.violation: individual SELECT, column UPDATE
0473  sdl033  Priv.violation: GRANT all on view but not table
0474  sdl034  Priv.violation: need SELECT for searched UPDATE
0475  sdl034  Priv.violation: GRANT ALL w/o GRANT OPTION
0476  sdl034  Priv.violation: GRANT OPTION view but not table
0477  sdl035  Priv.violation: GRANT only SELECT on view
0478  sdl035  Priv.violation: GRANT only INSERT on view
0479  sdl035  Priv.violation: GRANT only UPDATE on view
0480  sdl035  Priv.violation: GRANT only DELETE on view
0481  sdl036  Priv.violation: no privileges on view
0482  sdl036  GRANT ALL PRIVILEGES on view
0483  sdl036  Priv.violation: GRANT UPDATE not grantable on view
0484  sdl032  Priv.violation: SELECT and column UPDATE on view
0485  sdl032  Priv.violation: SELECT and column UPDATE cursor
0486  cdr028  Priv.violation: illegal REFERENCES
0487  dml081  SQLSTATE 00000: successful completion
0488  dml081  SQLSTATE 21000: cardinality violation
0489  dml081  SQLSTATE 02000: no data
0490  dml081  SQLSTATE 22012: data exception/division by zero
0491  dml082  SQLSTATE 22022: data exception/indicator overflow
0492  dml082  SQLSTATE 22019: data exception/invalid escape char
0493  dml082  SQLSTATE 22025: data exception/invalid escape seq.
0494  dml082  SQLSTATE 22003: data exception/numeric val.range 1
0495  sdl037  Priv.violation: illegal GRANT to self
0496  dml083  SQLSTATE 22002: data exception/null but no indic
0497  dml091  SQLSTATE 22003: data exception/numeric val.range 2
0498  dml083  SQLSTATE 22001: data exception/string right trunc.
0499  ccc010  SQLSTATE 22024: data exception/unterminat.C string
0500  dml083  SQLSTATE 01003: warning/null elim. in set function
0501  dml083  SQLSTATE 01004: warning/string right truncation
0502  dml081  SQLSTATE 24000: invalid cursor state
0503  dml084  SQLSTATE 42000: syntax error or access rule vio.1
0504  dml084  SQLSTATE 42000: syntax error or access rule vio.2
0505  dml082  SQLSTATE 44000: with check option violation
0506  mpa006  SQLSTATE 40001: trans.rollback/serialization fail.
0507  mpa007  Transactions serializable:  dueling cursors
```

0508	dml085	Delimited identifiers
0509	dml085	Renaming columns with AS for ORDER BY
0510	dml085	<parameter name> = <column name> (OK with SQL-92)
0511	dml086	CHECK clauses in nested views(clarified in SQL-92)
0512	dml090	<value expression> for IN predicate
0513	dml090	NUMERIC(4) implies CHECK BETWEEN -9999 AND 9999
0514	dml141	SQLSTATE 23000: integrity constraint violation
0515	dml132	FIPS sizing: NUMERIC (15) decimal precision
0516	cdr030	SQLSTATE 23000: integrity constraint violation
0517	mpa008	Transactions serializable: Twins Problem
0518	dml087	CREATE VIEW with DISTINCT
0519	dml087	CREATE VIEW with subqueries
0520	dml087	Underscores are legal and significant
0521	dml088	New format in MODULE-<parameter declaration list>
0522	cdr029	No implied natural join on FROM T1, T2
0523	dml090	<value expression> for BETWEEN predicate
0524	dml132	FIPS sizing: 100 Items in a SELECT list
0525	dml132	FIPS sizing: 15 Table references in SQL statement
0526	cdr031	FIPS sizing: Length FOREIGN KEY column list = 120
0527	dml142	Priv. violation: HU
0528	dml142	Tables are multi-sets: cursor operations
0529	dml143	Priv. violation: SELECT in <insert statement>
0530	mpquic	Interactive SQL serializability: dirty read
0531	mpquic	Interactive serializability: non-repeatable read
0532	ada007	package SQLSTATE_CODES
0533	ada008	Misc. in package SQL_STANDARD
0534	ada009	ADA Tasks
0535	cob010	COBOL - BINARY PICTURE for INTEGER, SMALLINT type
0536	cob010	FIPS sizing - COBOL BINARY decimal precision >= 9
0537	cdr029	Table check constraint: column vs. column
0538	cdr029	With check option: column vs. column
0539	mpquic	Interactive SQL serializability: phantom read
0554	dml085	More column renaming - single row select with join
0556	dml123	Static insert, dynamic fetch, static commit
0557	dml123	Static insert, dynamic commit, static rollback
0558	dml123	Dynamic insert, static delete, dynamic count
0559	dml123	Static insert, dynamic rollback, static fetch
0560	mpa010	Table privileges vs. column privileges
0561	dml149	Double SET TRANSACTION
0562	mpquic	Interactive serializability: ISOLATION MODE
0564	dml090	Outer ref. directly contained in HAVING clause
0565	dml092	VARCHAR for Transitional SQL
0566	dml093	VARCHAR for TSQL: dynamic version
0567	dml094	CHAR type in Dynamic SQL
0568	isi001	INFORMATION_SCHEMA.TABLES definition
0569	isi002	INFORMATION_SCHEMA.VIEWS definition

```
0570  isi003  INFORMATION_SCHEMA.COLUMNS definition
0571  isi004  INFORMATION_SCHEMA.SCHEMATA definition
0572  isi005  INFORMATION_SCHEMA.TABLE_PRIVILEGES definition
0573  isi006  INFORMATION_SCHEMA.COLUMN_PRIVILEGES definition
0574  isi007  Orphaned IS data structures, Intermediate SQL
0575  isi008  VARCHAR in INFORMATION_SCHEMA
0576  dml095  NUMERIC type in Dynamic SQL
0577  dml096  DECIMAL type in Dynamic SQL
0578  dml097  INTEGER and SMALLINT types in Dynamic SQL
0579  dml098  FIPS sizing, Dynamic SQL exact numerics
0580  dml098  FIPS sizing, Dynamic SQL approximate numerics
0581  dml099  Implicit numeric casting (feature 9) dynamic
0582  dml099  Implicit numeric casting (feature 9) static
0583  dml099  FIPS sizing, Dynamic SQL character strings
0584  dml092  FIPS sizing, VARCHAR (254) strings (static)
0585  dml093  FIPS sizing, VARCHAR (254) strings (dynamic)
0586  dml098  Sizing of FLOAT in a descriptor (dynamic)
0587  dml100  SET TR READ ONLY / READ WRITE (static)
0588  dml101  SET TR READ ONLY / READ WRITE (dynamic)
0589  mpa013  SET TR ISOLATION LEVEL (static)
0590  mpa012  SET TR ISOLATION LEVEL (dynamic)
0591  dml104  NATURAL JOIN (feature 4) (static)
0592  dml104  INNER JOIN (feature 4) (static)
0593  dml104  LEFT OUTER JOIN (feature 4) (static)
0594  dml104  RIGHT OUTER JOIN (feature 4) (static)
0595  dml105  NATURAL JOIN (feature 4) (dynamic)
0596  dml105  INNER JOIN (feature 4) (dynamic)
0597  dml105  LEFT OUTER JOIN (feature 4) (dynamic)
0598  dml105  RIGHT OUTER JOIN (feature 4) (dynamic)
0599  dml106  UNION in views (feature 8) (static)
0600  dml107  UNION in views (feature 8) (dynamic)
0601  dml106  DATETIME data types (feature 5) (static)
0602  dml107  DATETIME data types (feature 5) (dynamic)
0603  ist001  INFO_SCHEM.TABLES definition
0604  ist002  INFO_SCHEM.VIEWS definition
0605  ist003  INFO_SCHEM.COLUMNS definition
0606  ist004  INFO_SCHEM.SCHEMATA definition
0607  ist005  INFO_SCHEM.TABLE_PRIVILEGES definition
0608  ist006  INFO_SCHEM.COLUMN_PRIVILEGES definition
0609  ist007  Orphaned IS data structures, Transitional SQL
0610  ist008  VARCHAR in INFO_SCHEM
0611  dml106  FIPS sizing, DATETIME data types (static)
0612  dml107  FIPS sizing, DATETIME data types (dynamic)
0613  dml106  <datetime value function> (static)
0614  dml107  <datetime value function> (dynamic)
0615  dml106  DATETIME-related SQLSTATE codes (static)
```

```
0616  dml107  DATETIME-related SQLSTATE codes (dynamic)
0617  dml108  DATETIME with predicates, set fns (static)
0618  dml109  DATETIME with predicates, set fns (dynamic)
0619  dml110  DATETIME cursor operations (static)
0620  dml111  DATETIME cursor operations (dynamic)
0621  dml112  DATETIME NULLs (static)
0622  dml113  DATETIME NULLs (dynamic)
0623  dml112  OUTER JOINs with NULLs and empty tables (static)
0624  dml113  OUTER JOINs with NULLs and empty tables (dynamic)
0625  dml112  ADD COLUMN and DROP COLUMN (static)
0626  dml113  ADD COLUMN and DROP COLUMN (dynamic)
0627  mpa010  <grant statement> (static)
0628  mpa010  <revoke statement> (static)
0629  mpa011  <grant statement> (dynamic)
0630  mpa011  <revoke statement> (dynamic)
0631  dml112  Datetimes in a <default clause> (static)
0632  dml113  Datetimes in a <default clause> (dynamic)
0633  dml112  TRIM function (static)
0634  dml113  TRIM function (dynamic)
0635  dml114  Feature 13, grouped operations (static)
0636  dml115  Feature 13, grouped operations (dynamic)
0637  dml114  Feature 14, Qualified * in select list (static)
0638  dml115  Feature 14, Qualified * in select list (dynamic)
0639  dml114  Feature 15, Lowercase Identifiers (static)
0640  dml115  Feature 15, Lowercase Identifiers (dynamic)
0641  dml114  Feature 16, PRIMARY KEY enhancement (static)
0642  dml115  Feature 16, PRIMARY KEY enhancement (dynamic)
0643  dml133  Feature 17, Multiple schemas per user
0644  dml116  Feature 18, Multiple module support
0645  dml117  Feature 19, Referential delete actions (static)
0646  dml118  Feature 19, Referential delete actions (dynamic)
0647  dml119  Feature 20, CAST functions (static)
0648  dml120  Feature 20, CAST functions (dynamic)
0649  dml121  Feature 22, Explicit defaults (static)
0650  dml122  Feature 22, Explicit defaults (dynamic)
0651  dml121  Feature 24, Keyword relaxations (static)
0652  dml122  Feature 24, Keyword relaxations (dynamic)
0653  dml124  Descriptors:  DESCRIBE OUTPUT
0654  dml124  Descriptors:  INTO SQL DESCRIPTOR
0655  dml124  Descriptors:  USING SQL DESCRIPTOR
0656  dml124  Descriptors:  datetimes
0657  dml125  Descriptors:  VARCHAR
0658  dml125  Descriptors:  SQLSTATE codes
0659  dml125  Descriptors:  TSQL orphaned features
0660  dml126  Dynamic SQL SQLSTATEs
0661  dml121  Errata:  datetime casting (static)
```

```
0662  dml122  Errata:  datetime casting (dynamic)
0663  dml121  Errata:  datetime SQLSTATEs (static)
0664  dml122  Errata:  datetime SQLSTATEs (dynamic)
0665  dml127  Diagnostics:  statement information
0666  dml127  Diagnostics:  condition information
0667  dml152  Diagnostics:  access violations
0668  dml152  Diagnostics:  COMMAND_FUNCTION (static)
0669  dml152  Diagnostics:  COMMAND_FUNCTION F# 3, 11
0670  dml126  Diagnostics:  COMMAND_FUNCTION (dynamic)
0671  dml126  Diagnostics:  DYNAMIC_FUNCTION
0672  dml152  Diagnostics:  Multiple conditions
0673  dml152  Diagnostics SQLSTATE:  inv. cond. number
0674  dml128  Diagnostics:  TSQL orphaned features
0675  dml128  Diagnostics:  MORE
0676  dml129  Diagnostics:  VARCHAR
0677  dml129  VARCHAR with <like predicate>
0678  dml130  Data type semantics with NULL / NOT NULL
0679  dml130  INFO_SCHEM:  Table data types
0680  dml130  INFO_SCHEM:  View data types
0681  dml129  INFO_SCHEM:  Varchar data types
0682  dml130  INFO_SCHEM:  Datetime data types
0683  dml131  INFO_SCHEM:  Changes are visible
0684  dml131  INFO_SCHEM:  Visibility to other users
0685  dml131  INFO_SCHEM:  Privileges and privilege views
0686  dml131  INFO_SCHEM:  Primary key enh. is not null
0687  dml131  INFO_SCHEM:  Multiple schemas per user
0688  dml134  INFO_SCHEM:  Dynamic changes are visible
0689  dml134  Many Trans SQL features #1:  inventory system
0690  dml134  Many Trans SQL features #2:  talk show schedule
0691  dml134  INFO_SCHEM:  SQLSTATEs for length overruns
0692  dml135  Many TSQL features #3:  enhanced proj/works
0693  dml135  Many TSQL features #4:  enhanced INFO_SCHEM
0694  dml135  Interval Arithmetic and Casts
0695  dml135  <updatability clause> in <declare cursor>
0696  dml136  Many TSQL features #5:  Video Game Scores
0697  dml137  Erratum:  drop behavior, constraints (static)
0698  dml138  Erratum:  drop behavior, constraints (dynamic)
0699  dml139  <drop behavior> on a REVOKE (static)
0700  xop700  X/O,DEFAULTS and LIMITS for DATA TYPES
0701  xop701  X/O,WHENEVER SQLWARNING and scoping of C labels
0702  xop702  X/O,ALTER TABLE ADD
0703  xop703  X/O,CREATE INDEX on existent/non-existent tables
0706  xop706  X/O,CREATE INDEX on at least 6 columns
0707  xop707  X/O,Limit on depth of nested sub-queries
0708  xop708  X/O,Limit on the total length of an Index Key
0709  xop709  X/O,SQL Escape Clause Processing
```

0710	xop710	X/O,Acceptance of correctly placed SQLCA
0712	xop712	X/O,MAPPING OF DATATYPES ONTO SQL DECIMAL
0719	xop719	X/O,GRANT ALL with optional PRIVILEGES omitted
0720	xop720	X/O,GRANT ALL with optional PRIVILEGES omitted
0721	xop721	X/O,REVOKE ALL with optional PRIVILEGES omitted
0722	xop722	X/O,REVOKE ALL with optional PRIVILEGES omitted
0723	xop723	X/O,DROP TABLE with outstanding grants and views
0724	xop712	X/O,MAPPING ONTO SQL SMALLINT, DECIMAL AND INTEGER
0725	xop725	X/O,INCLUDE SQLCA IN LINKAGE SECTION
0829	dml140	<drop behavior> on a REVOKE (dynamic)
0830	flg010	FIPS Flagger - WHENEVER SQLWARNING
0831	flg011	FIPS Flagger - ADD (column, ...)
0832	flg012	FIPS Flagger - CREATE INDEX
0833	flg013	FIPS Flagger - INCLUDE SQLCA
0834	dml144	<length expression> (static)
0835	dml144	<character substring function> (static)
0836	dml145	<length expression> (dynamic)
0837	dml145	<character substring function> (dynamic)
0838	dml146	<character substring function> varchar
0839	dml144	Composed <length expression> and SUBSTRING
0840	dml147	Roll back schema manipulation
0841	dml147	Multiple-join and default order of joins
0842	dml147	Multi-column joins
0843	dml148	Ordering of column names in joins
0844	dml148	Outer join predicates
0845	dml150	Parameters and indicators in dynamic SQL statement
0846	dml149	Feature 20, CAST functions (static) nits
0847	mpa009	Dynamic SQL: serializability
0848	dml153	Query spec with subquery is now updatable
0849	dml154	Descriptors: datetime length in positions
0850	dml155	Comparing fixed vs. variable length char strings
0851	ada010	Errata: SQL_STANDARD changed to Interfaces.SQL
0852	dml154	Transitive grant in COLUMN_PRIV, TABLE_PRIV
0853	dml151	Exceptions not affecting position of cursor
0854	dml154	Informational: mixing SDL and DML
0855	dml156	Dynamic SQL syntax errors
0856	dml157	Transitional Schema Definition
0857	dml158	<join condition> set function, outer reference
0858	dml159	? (dyn parm spec) in <having clause>
0859	dml160	<joined table> contained in <select list>
0860	dml160	Domains over various data types
0861	dml161	CURRENT_USER, SESSION_USER, SYSTEM_USER
0862	dml161	CURRENT_USER etc. with set session authid
0863	dml162	<joined table> directly contained in cursor, view
0864	dml163	Intermediate Dynamic SQL syntax errors
0865	dml163	Result data types for case expressions

```
0866  dml163  Case expressions in other than SELECT
0867  dml164  LIKE enhancements:  keyword search
0868  dml164  More <unique predicate>
0869  dml164  More table operations
0870  dml165  Non-identical descriptors in UNION
0871  ada011  Errata:  Interfaces.SQL.Numerics--TC3 clause 23.3
0872  ada012  Errata:  Interfaces.SQL.Varying--TC3 clause 23.3
0873  dml166  Dynamic schema creation
0874  dml167  INFORMATION_SCHEMA catalog columns
0875  dml167  INFORMATION_SCHEMA column coverage
0876  dml168  SQL_IDENTIFIER and CHARACTER_DATA domains
0877  dml169  Intermediate DB, Flag at Entry level
0878  dml168  Keyword COLUMN in ALTER TABLE is optional
0879  dml168  <drop table constraint definition>
0880  dml170  Long constraint names, cursor names
0881  dml170  Long character set names, domain names
0882  dml171  More full outer join
0883  ada013  Errata:  Interfaces.SQL.Varying.NCHAR--TC3, 23.3
0884  dml172  ASCII_FULL and SQL_TEXT in column definition
0885  dml173  FIPS sizing, CHAR (1000)
0886  dml174  FIPS sizing, VARCHAR (1000)
0887  dml175  FIPS sizing, NCHAR (500)
0888  dml176  FIPS sizing, NCHAR VARYING (500)
0889  dml177  FIPS sizing, INTEGER binary prec >= 31
0890  dml177  FIPS sizing, SMALLINT binary prec >= 15
0891  dml178  FIPS sizing, 250 columns, 4000 char data statement
0892  dml179  FIPS sizing, rowlen >= 8000, statement var >= 4000
0893  dml180  FIPS sizing, descriptor occurrences >= 100
0894  dml181  FIPS sizing, length of column lists >= 750
0895  dml182  FIPS sizing, columns in list >= 15
0896  dml183  FIPS sizing, 50 WHEN clauses in a CASE expression
0897  dml184  Constraint usage redux
0898  dml185  COLUMN_DEFAULT interpretation
0899  dml186  FIPS sizing, INTERVAL decimal leading field prec
7001  xts700  NULLIF producing NULL
7002  xts798  NULLIF producing non-NULL
7003  xts799  COALESCE with three <value expression>s
7004  xts701  Compound char. literal in <comparison predicate>
7005  xts701  Compound character literal as inserted value
7006  xts701  Compound character literal in a <select list>
7007  xts702  LIKE with unrestricted <match value>
7008  xts702  LIKE with general char. value for pattern + escape
7009  xts702  LIKE with zero-length escape
7010  xts703  UNIQUE predicate, single table, all values distinc
7011  xts703  UNIQUE PREDICATE, table subquery with non-null dup
7012  xts703  UNIQUE predicate, duplicates containing null
```

7013 xts713 Schema definition in an SQL statement-single table
7014 xts714 Schema definition named schema, implicit auth-id.
7015 xts715 Schema definition - explicit name and auth-id.
7016 xts716 SET SESSION AUTHORIZATION to current auth-id.
7017 xts717 SET SESSION AUTH. to current auth-id in. transactn
7018 xts718 SET SESSION AUTHORIZATION to different value
7019 xts719 Access to KEY_COLUMN_USAGE view
7020 xts720 Access to VIEW_TABLE_USAGE view
7021 xts721 Access to VIEW_COLUMN_USAGE view
7022 xts722 Access to CONSTRAINT_TABLE_USAGE view
7023 xts723 Access to CONSTRAINT_COLUMN_USAGE view
7024 xts724 Access to COLUMN_DOMAIN_USAGE view
7025 xts725 Flagging - Full SQL INSENSITIVE cursor
7026 xts726 Flagging Full SQL - cursor FOR UPDATE and ORDER BY
7027 xts727 Flagging - Full SQL - <explicit table> in <qry exp
7028 xts728 Flagging,Full SQL,<null predicate> with two-col ro
7029 xts729 Column name with 19 and 128 characters - regular.
7030 xts730 Table name with 19 characters - delimited.
7031 xts731 View name with 128 characters - delimited.
7032 xts732 NATURAL FULL OUTER JOIN <table ref> -- static.
7033 xts733 FULL OUTER JOIN <table ref> ON <search condition>
7034 xts734 National Character data type in comparison predica
7035 xts735 INSERT National character literal in NCHAR column
7036 xts736 Update NCHAR VARYING column with value from NCHAR
7037 xts737 Scrolled cursor with ORDER BY DESC, FETCH NEXT
7038 xts738 Scrolled cursor with ORDER BY DESC, FETCH PRIOR
7039 xts739 Scrolled cursor with ORDER BY int, name ASC, FETCH
7040 xts740 COUNT(ALL <column name>) with Nulls in column
7041 xts741 COUNT(ALL NULLIF...) with generated Nulls
7042 xts742 COUNT ALL <literal>
7044 xts744 Presence of SQL_CHARACTER in CHARACTER_SETS view
7045 xts745 Presence of ASCII_GRAPHIC in CHARACTER_SETS view
7046 xts746 Presence of LATIN1 in CHARACTER_SETS view
7047 xts747 Presence of ASCII_FULL in CHARACTER_SETS view
7048 xts748 Named constraint in column definition in schema de
7049 xts749 Named table constraint in table definition.
7050 xts750 Named domain constraint.
7051 xts751 Name of violated column constraint returned in dia
7052 xts752 ALTER TABLE ADD TABLE CONSTRAINT
7053 xts753 ALTER TABLE ADD COLUMN WITH <data type>
7054 xts754 ALTER TABLE ADD COLUMN WITH domain and constraint
7055 xts755 ALTER TABLE DROP COLUMN RESTRICT
7056 xts756 ALTER TABLE DROP COLUMN CASCADE
7058 xts758 Scrolled cursor FETCH ABSOLUTE non-literal, after
7059 xts759 Scrolled cursor on grouped view,FETCH RELATIVE,FIR
7060 xts760 MAX of column derived from <set function specifica

7061	xts761	Defined character set in <comparison predicate>
7062	xts762	Defined character set in <like predicate>
7063	xts763	Access to CHARACTER_SETS view
7064	xts764	REVOKE USAGE on character set RESTRICT
7065	xts765	REVOKE USAGE on character set CASCADE
7066	xts766	Drop character set no granted privileges
7067	xts767	DROP character set, outstanding granted privileges
7068	xts768	Presence of SQL_TEXT in CHARACTER_SETS view
7069	xts769	<Character set specification> of LATIN1 in <litera
7070	xts770	<Character set specification> of SQL_CHARACTER in
7071	xts771	CHARACTER SET ASCII_GRAPHIC in <data type>
7500	yts750	CREATE DOMAIN -SQL Procedure statement,no options
7501	yts751	CREATE DOMAIN as SQL proc statement with default
7502	yts752	CREATE DOMAIN-SQL proc statement with constraint
7503	yts753	DROP DOMAIN RESTRICT
7504	yts754	DROP DOMAIN CASCADE - domain definition in use
7505	yts755	DROP DOMAIN CASCADE-domain w. default + constraint
7506	yts756	Domain Constraint Containing VALUE
7507	yts757	INSERT value in column defined on domain
7508	yts757	Put value in col defined on dom breaking constrain
7509	yts759	GRANT USAGE on a domain
7510	yts776	DROP SCHEMA - empty schema with restrict
7511	yts777	DROP SCHEMA - non-empty schema
7512	yts783	Scr.cursor,no ORDER,FETCH all,FIRST,LAST,NEXT
7513	yts784	Scr.cursor with joined table, FETCH ABS literal
7517	yts762	<query expression> with EXCEPT
7518	yts763	<query expression> with INTERSECT CORRESPONDING
7519	yts764	<query expression> with UNION ALL CORRESPONDING BY
7520	yts778	ALTER TABLE SET COLUMN DEFAULT
7521	yts779	ALTER TABLE DROP COLUMN DEFAULT
7522	yts788	CREATE CHARACTER SET, implicit default collation
7523	yts789	CREATE CHAR SET in schema def,COLLATION FROM DEFLT
7524	yts790	GRANT USAGE on character set,WITH GRANT OPTION
7525	yts791	GRANT USAGE on character set,WITH GRANT OPTION
7526	yts792	GRANT USAGE on character set,WITH GRANT OPTION
7527	yts793	GRANT USAGE on character set,no WGO
7528	yts794	GRANT USAGE on character set, no WGO
7529	yts795	GRANT USAGE on character set, no WGO
7530	yts796	<scalar subquery> as first operand in <comp pred>
7531	yts799	<subqry> as<row val constructor>in<null predicate>
7532	yts800	<nul pred><interval value exp> as <row value cons>
7534	yts760	CASE expression with one simple WHEN
7535	yts761	CASE expression with searched WHEN
7536	yts781	Set local time zone - valid value
7537	yts765	Explicit table constrnts in TABLE_CONSTRAINTS view
7538	yts765	Column constraints in TABLE_CONSTRAINTS view

```
 7539  yts765  Unique identification in TABLE_CONSTRAINTS view
 7540  yts766  Explicit table constrnts - REFERENTIAL_CONSTRAINTS
 7541  yts766  Column constraints in REFERENTIAL_CONSTRAINTS view
 7542  yts766  Unique id in REFERENTIAL_CONSTRAINTS view
 7543  yts766  Values in columns of REFERENTIAL_CONSTRAINTS view
 7544  yts767  Explicit table constr. in CHECK_CONSTRAINTS view
 7545  yts767  Column constraints in CHECK_CONSTRAINTS view
 7546  yts767  Domain constraints in CHECK_CONSTRAINTS view
 7547  yts767  Unique identification in CHECK_CONSTRAINTS view
 7548  yts802  Support of SQL_FEATURES tab. in documentatn schema
 7549  yts803  Support SQL_SIZING table in documentation schema
 7550  yts768  Access to SCHEMATA view
 7551  yts769  Access to DOMAINS view
 7552  yts770  Access to DOMAIN_CONSTRAINTS view
 7553  yts771  Access to CHARACTER_SETS view
 7554  yts772  Access to ASSERTIONS view
 7555  yts773  Access to SQL_LANGUAGES view
 7556  yts774  Access to INFORMATION_SCHEMA_CATALOG_NAME base tab
 7557  yts775  SQL host prog. with duplicate local variable names
 7558  yts797  <scalar subquery> in SET of searched update
 7559  yts798  <scalar subqry> in <sel.list> of single-row sel.
 7560  yts780  <time zone interval> in literal
 7561  yts782  Set local time zone - invalid value, exception
 7562  yts805  Schema with crossed referential const. bet. tables
 7563  yts806  NATURAL FULL OUTER JOIN <table ref> - dynamic
 7564  yts807  TIMEZONE_HOUR + TIMEZONE_MINUTE in <extract expr.>
 7565  yts808  LOCAL time zone in <datetime value expression>
 7566  yts809  TIME ZONE in <datetime value expression>
 7567  yts810  FULL OUTER JOIN<table ref>ON<search condition> dyn
 7568  yts811  WHERE <search cond.> referencing column
 7569  yts812  <null predicate>,concat. in<row value constructor>
 7570  yts813  <null predct>, <numrc val expr> in <row val cons>
 7571  yts814  <module character set specification>
------------------------------------------------------------------
899 rows selected

NOTE  This listing was created by the following query:
select testno,prog,descr from testcase order by testno;
========================= END OF APPENDIX ========================
```

TEd Change File
Sample Downloaded Embedded SQL COBOL File

```
! ------------------------------------------------------------
!                     IDENTIFICATION SECTION
!                        Embedded COBOL
! ------------------------------------------------------------
!
sub *.* /59-byte ID/
/59-byte ID/____________________________________________________/
!
! Document version of TEd used, for audit purposes.
!
sub *.* /TEd Version #/ -m
"TEd Version #"TEd Version $ver"
!
!
! ------------------------------------------------------------
!                     MAINTENANCE SECTION
! ------------------------------------------------------------
!
! Use the  -t  option on TEd to apply the official maintenance file.
! Then use a second  -t  option on TEd to apply this file.
!      OR
! Insert the entire maintenance file in this position.
!
! ------------------------------------------------------------
!                     LOGIN SECTION
! ------------------------------------------------------------
!
! The login problem is solved here:
! If passwords are system-login passwords, then no changes are necessary.
! For other implementations, changes may be needed.
!
! The call to AUTHID may be replaced with some implementor-defined syntax.
! For example, you may choose to code:
! rep *.pco /CALL "AUTHID"/
!            EXEC SQL CONNECT :uid IDENTIFIED BY :uid END-EXEC
!
! Other changes to establish the ANSI-standard environment would be coded here.
!
!
! ------------------------------------------------------------
!                     IMPLEMENTOR-DEFINED SECTION
! ------------------------------------------------------------
!
! Install implementor-defined precision of SQLCODE
! For example, to change precision from 9 to 4, code:
!            sub *.pco /SQLCODE/
```

```
!            /S9(9)/S9(4)/
!
! Install implementor-defined data type for indicator variable
! For example, to change data type to COMP, code:
!            sub *.pco /indic/
!            /DISPLAY SIGN LEADING SEPARATE/COMP/
!
! Modify dml063 to declare vendor-reserved key words
! For example, to declare UPDATE a vendor-reserved key word, code:
!            del *dml063.pco /01  UPDATE /
!            del *dml063.pco /:UPDATE/MOVE UPDATE TO/
!
!
! Install vendor-supplied extensions to demonstrate FIPS Flagging:
! (Replace the underscores with vendor-supplied extensions.)
!
sub *{flg005,xop709}.pco //
/SUBSTR(EMPNAME,1,3) = 'Ali'/_____________________/
/ABS(GRADE) = 12/_________________/
!
! Change 18-character AUTHORIZATION ID to max supported in this DBMS/OS:
!            sub *.* /CANWEPARSELENGTH18/
!            /CANWEPARSELENGTH18/WECANPARSE12/
!
! Change COBOL programs to specify correct source and object computers
sub *.[ps]co /COMPUTER/
/xyz/_______________/
!
! ----------------------------------------------------------------
!                      APPROVED CHANGES SECTION
! ----------------------------------------------------------------
!
! Include here any changes approved by NIST for this validation:
!
! For example, to lengthen concurrency test #0230 (according to documentation),
!   to allow interleaving for a DBMS using "burst" mode CPU allocation:
!            sub *mp?001.[ps]* /keymax/ -i
!            /50/300/
!
!
! ----------------------------------------------------------------
!                      SYNTAX DEFICIENCIES SECTION
! ----------------------------------------------------------------
!
!
! If a program cannot be executed because the precompiler issues
!   a fatal syntax error message for a minor deficiency which is not
!   the purpose of the test case, propose a change to NIST.
! A syntax change should be made to allow an SQL implementation
!   to demonstrate functionality, i.e. semantic conformance.
```

```
! A global change will be counted as a single deficiency.
!
! ----------------------------------------------------------------
!                      DELETED TESTS SECTION
! ----------------------------------------------------------------
!
! If a program cannot be executed because the precompiler issues
!   a fatal syntax error message for one of the tests, either propose a
!   change to NIST or delete the entire test so that the rest of the tests
!   in the program may execute.  If this test is required, and not a FIPS
!   Flagger test, it will appear as a deficiency on REPORT 4.
! For COBOL, a performed paragraph is found at the end of the program.
!   So a separate delete command is needed for a performed paragraph.
!
! For example, to remove test number 278, code
!            del *dml061.pco /BEGIN TEST0278/END TEST0278/
!
!
======================== END OF APPENDIX ========================
```

SAMPLE TEd Change File
Vendor-modified File Proposed for Validation
Embedded COBOL File CHGPCO.TED

```
! ----------------------------------------------------------------
!                     IDENTIFICATION SECTION
!                        Embedded COBOL
! ----------------------------------------------------------------
!
sub *.* /59-byte ID/
/59-byte ID/BugFreeDB v.783, UNIX SYS X 2.2, SuperPC 440/
!
! Document version of TEd used, for audit purposes.
!
sub *.* /TEd Version #/ -m
"TEd Version #"TEd Version $ver"
!
! ----------------------------------------------------------------
!                     MAINTENANCE SECTION
! ----------------------------------------------------------------
!
! Note: maintenance is included as first TEd file, via command:
!    ted  -t upd400.ted -t chgpco.ted -o dmlxxx.eco dmlxxx.pco
! where dmlxxx.pco is a NIST-supplied test file
! and chgpco.ted is this file.
! Note that our product uses file extension eco, instead of pco.
!
! ----------------------------------------------------------------
!                     LOGIN SECTION
! ----------------------------------------------------------------
!
! The call to AUTHID is replaced with LINK statement.
rep *.pco /CALL "AUTHID"/
           EXEC SQL LINK SCHEMA :uid WITH PASSWORD :uid END-EXEC
!
! Other changes to establish the ANSI-standard environment.
ins> mp*.pco /EXEC SQL LINK SCHEMA/
           EXEC SQL SET TRANSACTION
                     ISOLATION LEVEL SERIALIZABLE END-EXEC
!
!
! ----------------------------------------------------------------
!                     IMPLEMENTOR-DEFINED SECTION
! ----------------------------------------------------------------
!
! We declare SQLCA which includes SQLCODE
! So replace SQLCODE with SQLCA, but outside of the DECLARE SECTION.
del *.pco /*01*SQLCODE*/ -p
ins> *.pco /WORKING-STORAGE/
```

```
          EXEC SQL INCLUDE sqlca END-EXEC
!
! Our implementor-defined data type for indicator variable is COMP.
sub *.pco /indic/ -i
/DISPLAY SIGN LEADING SEPARATE/COMP/
!
! Modify dml063 to declare vendor-reserved key words: WHERE and ROLLBACK.
!
del *dml063.pco /01  WHERE /
del *dml063.pco /:WHERE/MOVE WHERE TO/
del *dml063.pco /01  ROLLBACK /
del *dml063.pco /:ROLLBACK/MOVE ROLLBACK TO/
!
!
! Vendor-supplied extensions to demonstrate FIPS Flagging:
!
!     We don't support SUBSTR, but we can demonstrate LOWER:
sub *{flg005,xop709}.pco //
/SUBSTR(EMPNAME,1,3) = 'Ali'/LOWER (EMPNAME) = 'alice'/
!
!     No substitution for ABS(GRADE) = 12, it works for us.
!
! AUTHORIZATION ID max length is 8:
sub *.* /CANWEPARSELENGTH18/
/CANWEPARSELENGTH18/EIGHT_OK/
sub *.* /SULLIVAN1/
/SULLIVAN1/SULLIVN1/
!
! Change COBOL programs to specify correct source and object computers
sub *.[ps]co /COMPUTER/
/xyz/SuperPC-440/
!
! ----------------------------------------------------------------
!                      APPROVED CHANGES SECTION
! ----------------------------------------------------------------
!
! To lengthen concurrency test #0230 (according to documentation),
! Our DBMS uses "burst" mode CPU allocation:
sub *mp?001.[ps]* /keymax/ -i
/50/300/
!
! ----------------------------------------------------------------
!                      SYNTAX DEFICIENCIES SECTION
! ----------------------------------------------------------------
!
! The following syntax changes are made to allow for further testing.
!
! our precompiler doesn't understand END-EXEC; it wants semicolon.
sub *.pco /END-EXEC/
/END-EXEC/;/
```

```
!
! our numeric literals must not end with a decimal point.
sub *dml005.pco /LONG_INT/
/00./00.0/
!
! ---------------------------------------------------------------
!                     DELETED TESTS SECTION
! ---------------------------------------------------------------
!
! UNION and UNION ALL not supported in same SQL statement
! Remove test number 160 and the paragraphs it performs
del *dml001.pco /BEGIN TEST0160/END TEST0160/
del *dml001.pco /P43./     ./
!
======================== END OF APPENDIX ==========================
```

SAMPLE TEd Change File
Accepted Counter-proposed File
Embedded COBOL File CHGPCO.TED

```
! ----------------------------------------------------------------
!                     IDENTIFICATION SECTION
!                       Embedded COBOL
! ----------------------------------------------------------------
!
sub *.* /59-byte ID/
/59-byte ID/BugFreeDB v.783, UNIX SYS X 2.2, SuperPC 440/
!
! Document version of TEd used, for audit purposes.
!
sub *.* /TEd Version #/ -m
"TEd Version #"TEd Version $ver"
!
! ----------------------------------------------------------------
!                     MAINTENANCE SECTION
! ----------------------------------------------------------------
!
! Note: maintenance is included as first TEd file, via command:
!    ted  -t upd400.ted -t chgpco.ted -o dmlxxx.eco dmlxxx.pco
! where dmlxxx.pco is a NIST-supplied test file
! and chgpco.ted is this file.
! Note that our product uses file extension eco, instead of pco.
!
! ----------------------------------------------------------------
!                     LOGIN SECTION
! ----------------------------------------------------------------
!
! The call to AUTHID is replaced with LINK statement.
rep *.pco /CALL "AUTHID"/
          EXEC SQL LINK SCHEMA :uid WITH PASSWORD :uid END-EXEC
!
! Other changes to establish the ANSI-standard environment.
ins> mp*.pco /EXEC SQL LINK SCHEMA/
          EXEC SQL SET TRANSACTION
                     ISOLATION LEVEL SERIALIZABLE END-EXEC
!
!
! ----------------------------------------------------------------
!                     IMPLEMENTOR-DEFINED SECTION
! ----------------------------------------------------------------
!
! Our implementor-defined data type for indicator variable is COMP.
sub *.pco /indic/ -i
/DISPLAY SIGN LEADING SEPARATE/COMP/
!
```

```
! Modify dml063 to declare vendor-reserved key words: WHERE and ROLLBACK.
!
del *dml063.pco /01  WHERE /
del *dml063.pco /:WHERE/MOVE WHERE TO/
del *dml063.pco /01  ROLLBACK /
del *dml063.pco /:ROLLBACK/MOVE ROLLBACK TO/
!
!
! Vendor-supplied extensions to demonstrate FIPS Flagging:
!
!     We don't support SUBSTR, but we can demonstrate LOWER:
sub *{flg005,xop709}.pco //
/SUBSTR(EMPNAME,1,3) = 'Ali'/LOWER (EMPNAME) = 'alice'/
!
!     No substitution for ABS(GRADE) = 12, it works for us.
!
! AUTHORIZATION ID max length is 8:
sub *.* /CANWEPARSELENGTH18/
/CANWEPARSELENGTH18/EIGHT_OK/
sub *.* /SULLIVAN1/
/SULLIVAN1/SULLIVN1/
!
! Change COBOL programs to specify correct source and object computers
sub *.[ps]co /COMPUTER/
/xyz/SuperPC-440/
!
! ----------------------------------------------------------------
!                      APPROVED CHANGES SECTION
! ----------------------------------------------------------------
!
! To lengthen concurrency test #0230 (according to documentation),
! Our DBMS uses "burst" mode CPU allocation:
sub *mp?001.[ps]* /keymax/ -i
/50/300/
!
! ----------------------------------------------------------------
!                      SYNTAX DEFICIENCIES SECTION
! ----------------------------------------------------------------
!
! We declare SQLCA which includes SQLCODE
! So replace SQLCODE with SQLCA, but outside of the DECLARE SECTION.
! 1.  Does not support ANSI SQLCODE declaration:
del *.pco /*01*SQLCODE*/ -p
ins> *.pco /WORKING-STORAGE/
           EXEC SQL INCLUDE sqlca END-EXEC
!
! our precompiler doesn't understand END-EXEC; it wants semicolon.
! 2.  Does not support COBOL END-EXEC:
sub *.pco /END-EXEC/
/END-EXEC/;/
```

```
!
! our numeric literals must not end with a decimal point.
! 3.  Does not support exact numeric literal ending in decimal point.
sub *dml005.pco /LONG_INT/
/00./00.0/
!

! ---------------------------------------------------------------
!                     DELETED TESTS SECTION
! ---------------------------------------------------------------
!
! UNION and UNION ALL not supported in same SQL statement
! Remove test number 160 and the paragraphs it performs
del *dml001.pco /BEGIN TEST0160/END TEST0160/
del *dml001.pco /P43./     ./
!
======================== END OF APPENDIX ==========================
```

Sample Printout from Program Execution

```
NIST SQL Test Suite, V6.0, Embedded COBOL, dml001.pco

BugFreeDB v.783, UNIX SYS X 2.2, SuperPC 440

Ted Version 5.1 5/17/95

Date run YYMMDD: 961231 at hhmmssff: 09514908

                    TEST0001
declare with ORDER BY < column specification > DESC
reference X3.135-1992 section 13.1 General Rules 3) a)
     - - - - - - - - - - - - - - - - - - -

   *** FOR SELECT EMPNUM,HOURS
   ***     FROM   WORKS
   ***     WHERE  PNUM='P2'
   ***     ORDER  BY EMPNUM DESC

           EMPNO=E4  and HOUR1=20
           EMPNO=E3  and HOUR1=20
           EMPNO=E2  and HOUR1=80
           EMPNO=E1  and HOUR1=20

          EMPNO=E1 , i=4
The answer should be EMPNO=E1, i=4 &
             order by EMPNO DESC.

                  *** pass ***
=====================================================


====================== END OF APPENDIX ============================
```

Sample Summary Reports
PROBLEMS Listing

```
************************************************************
************************ PROBLEMS ************************
************************************************************

>>>>>>>>>>>>>>>>>> FIPS 127-2 Entry SQL <<<<<<<<<<<<<<<<<<

--------------- FIPS 127-2 Entry Syntax Flags --------------

0296  Test 0296 in flg005:  FIPS Flagger - vendor provided character function
0296    MAD = fail

0832  Test 0832 in flg012:  FIPS Flagger - CREATE INDEX
0832    MAD = missing
0832    PC  = missing
0832    PCO = missing
0832    PFO = missing

---------------- ISO/IEC 9075:1992 Entry SQL ---------------

0003  Test 0003 in dml001:  CURSOR with ORDER BY DESC integer, named column
0003    PC  = missing
0003    PFO = missing

0005  Test 0005 in dml001:  CURSOR with UNION ALL
0005    MAD = fail

0007  Test 0007 in dml003:  Error for second consecutive CLOSE
0007    MAD = missing
0007    PC  = fail
0007    PCO = missing
0007    PFO = fail

0399  Test 0399 in ccc009:  C language storage class and class modifier comb.
0399  NOTE:  visual chk
0399    PC  = missing

0429  Test 0429 in ada005:  Ada reminder to check SQL_STANDARD package format
0429  NOTE:  visual chk
0429    MAD = missing

0503  Test 0503 in dml084:  SQLSTATE 42000: syntax error or access rule vio.1
0503  NOTE:  synvio_yes
0503    PFO = fail

Total number of problems:  15
```

```
0503  Test 0503 in dml084:  SQLSTATE 42000: syntax error or access rule vio.1
0503  NOTE:  synvio_yes
0503    PFO = fail

Total number of problems:  15

====================== END OF APPENDIX ============================
```

Sample Summary Reports
TOTALS Listing

```
************************************************************
************************** TOTALS **************************
************************************************************

>>>>>>>>>>>>>>>>>>> FIPS 127-2 Entry SQL <<<<<<<<<<<<<<<<<<<

               MAD      PC     PCO     PFO
REQUIRED
  pass         431     440     442     433
  fail           2       1       0       2
  missing        3       3       2       2
  nogo           0       1       1       0
OPTIONAL
  pass           0       0       1       0
  fail           0       0       0       0
  missing        0       0       0       0
  nogo           0       0       0       0
UNDER REVIEW
  pass           0       0       0       0
  fail           0       0       0       0
  missing        0       0       0       0
  nogo           0       0       0       0

NA              50      41      40      49
DL               0       0       0       0
WD               0       0       0       0
               -----   -----   -----   -----
               486     486     486     486

Grand total = 1944
Problem total = 15

====================== END OF APPENDIX ============================
```

Sample Summary Reports
TEST RESULTS Listing

```
************************************************************
********************** TEST RESULTS **********************
************************************************************

>>>>>>>>>>>>>>>>>> FIPS 127-2 Entry SQL <<<<<<<<<<<<<<<<<<

--------------- FIPS 127-2 Entry Syntax Flags --------------

TESTNO MAD       PC        PCO       PFO
 0296  fail      pass      pass      pass
 0297  pass      pass      pass      pass
 0454  pass      pass      pass      pass
 0830  NA        pass      pass      pass
 0831  pass      nogo      nogo      pass
 0832  missing   missing   missing   missing
 0833  NA        pass      pass      pass

---------------- ISO/IEC 9075:1992 Entry SQL ---------------

TESTNO MAD       PC        PCO       PFO
 0001  pass      pass      pass      pass
 0002  pass      pass      pass      pass
 0003  pass      missing   pass      missing
 0004  pass      pass      pass      pass
 0005  fail      pass      pass      pass
 0006  pass      pass      pass      pass
 0007  missing   fail      missing   fail
 0008  pass      pass      pass      pass
 0009  pass      pass      pass      pass
 0010  pass      pass      pass      pass
 0012  pass      pass      pass      pass
 0013  pass      pass      pass      pass
 0014  pass      pass      pass      pass
 0015  pass      pass      pass      pass
 0016  pass      pass      pass      pass
 0017  pass      pass      pass      pass
 0018  pass      pass      pass      pass
...
 0563  pass      pass      pass      pass
 0564  pass      pass      pass      pass

Total number of tests:  486

** denotes OPTIONAL/INFORMATIONAL test

====================== END OF APPENDIX ============================
```

"FIPS Flaggers" Examples

Program FLG005:

```
$SQLPRE/COBOL/ansi/flag FLG005.PCO
         WHERE SUBSTRING(EMPNAME FROM 1 FOR 3) = 'Ali';
                1
%SQL-I-NONSTASYN92E, (1) Nonstandard SQL92 Entry-level syntax
         WHERE GRADE = '-12';
                       1
%SQL-I-NUMCMPTXT, (1) Numeric column compared with string literal as text
         WHERE GRADE = '-12';
                2
%SQL-I-NONSTACON, (2) The standard does not permit this data type conversion
```

Program FLG006:

```
$SQLPRE/COBOL/ansi/flag FLG006.PCO
             EXEC SQL INSERT INTO TABLEFGHIJKLMNOPQ19 VALUES (299)
                                  1
%SQL-I-NONSTALNM, (1) Nonstandard long name
               FROM   TABLEFGHIJKLMNOPQ19 END-EXEC
                      1
%SQL-I-NONSTALNM, (1) Nonstandard long name
             EXEC SQL SELECT COLUMN123456789IS19
                             1
%SQL-I-NONSTALNM, (1) Nonstandard long name
               FROM   VIEWABCDEFGHIKLMN19 END-EXEC
                      1
%SQL-I-NONSTALNM, (1) Nonstandard long name
```

Program FLG009:

```
$SQLPRE/COBOL/ansi/flag FLG009.PCO
   SELECT PTYPE, CITY FROM PROJ;
          1
%SQL-I-NONSTACUC, (1) The standard requires columns merged by UNION be identical
    SELECT EMPNUM, HOURS FROM WORKS;
                   1
%SQL-I-NONSTACUC, (1) The standard requires columns merged by UNION be identical
```

========== END OF APPENDIX ==========

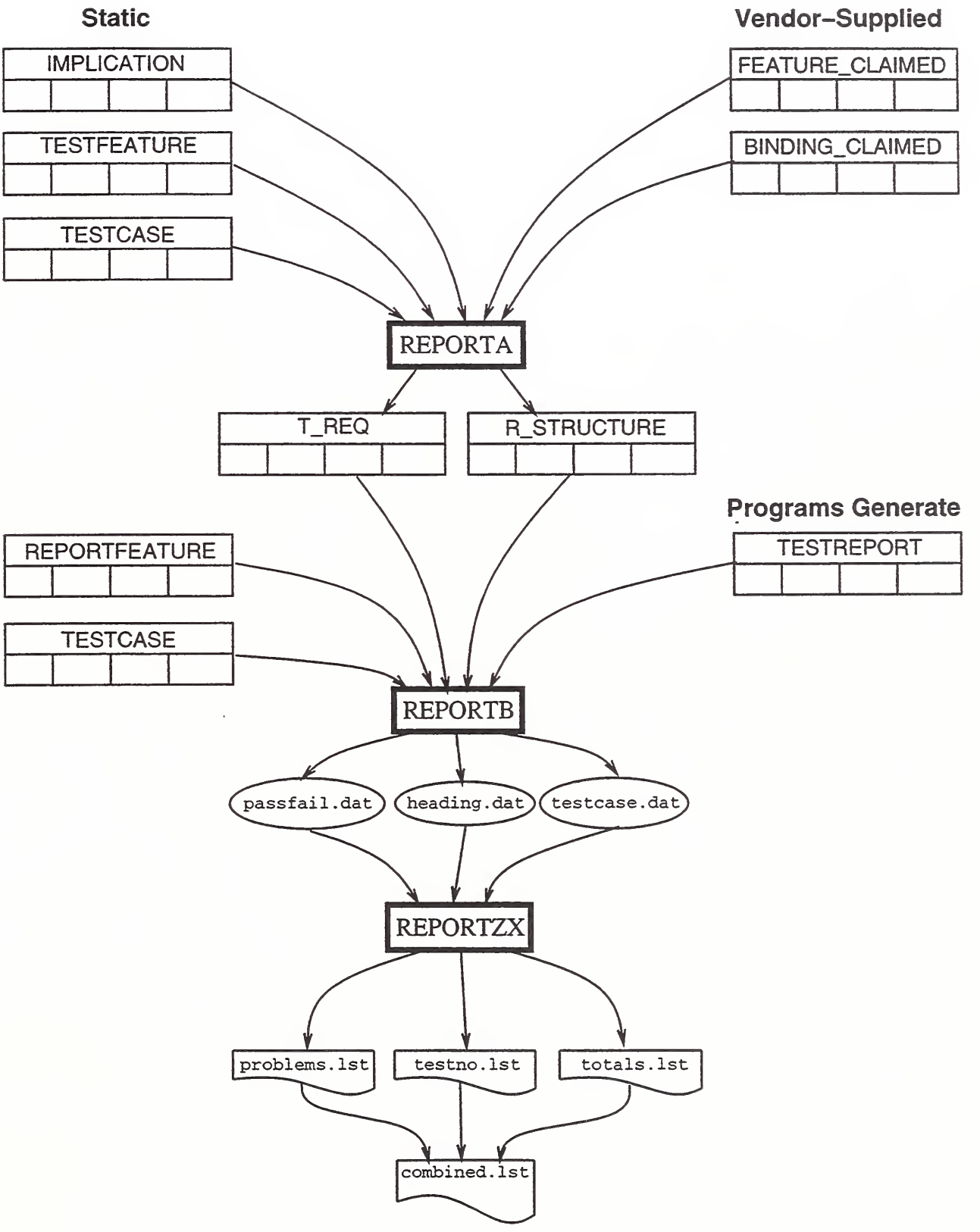
Automated Reporting System Diagrams
Table Definitions for Reporting System Diagrams

```
NIST SQL Test Suite, V6.0, report.sql

-- Schema for reporting structure of SQL Test Suite, Version 6.0
-- Followed by sample data and queries

-- =========================================================
--   Static tables to define the test suite structure
-- =========================================================

-- This table is an enumeration of all features
--    and collections of features (profiles) to be tested.
-- This is a reference table of codes (FEATURE1) and a
--    lookup table of names.
-- A profile has the value P in column1 of FEATURE1.
-- A logical profile has the value L in column1 of FEATURE1.
-- A lowest-level "leaf" feature is numeric in column1 of FEATURE1.
-- [A logical profile is a convenient grouping of features or
--   profiles, for purposes of recursion, but not reporting.].

   CREATE TABLE REPORTFEATURE
     (FEATURE1      CHAR(4) NOT NULL PRIMARY KEY,
      FEATURENAME   CHAR(30) NOT NULL);

-- This table describes the reporting structure for SQL testing;
--   i.e., the network of relationships between REPORTFEATURE rows.
-- Each row is a directed arc in the network.

-- Profiles and logical profiles are nodes in the hierarchy.

   CREATE TABLE IMPLICATION
     (PARENT_F  CHAR(4) NOT NULL REFERENCES REPORTFEATURE,
      CHILD_F   CHAR(4) NOT NULL REFERENCES REPORTFEATURE);

-- List of programs, authorization identifiers, and
--  special notes on how to run each the program.
-- P_NOTE indicates whether a test is a concurrency test,
--  requires a subroutine, etc.

   CREATE TABLE TESTPROG
     (PROG      CHAR(6) NOT NULL PRIMARY KEY,
      AUTHID    CHAR(18) NOT NULL,
      P_NOTE    CHAR(10));

-- List of test cases, descriptions, containing program, and
--  special notes on operational problems, such as
```

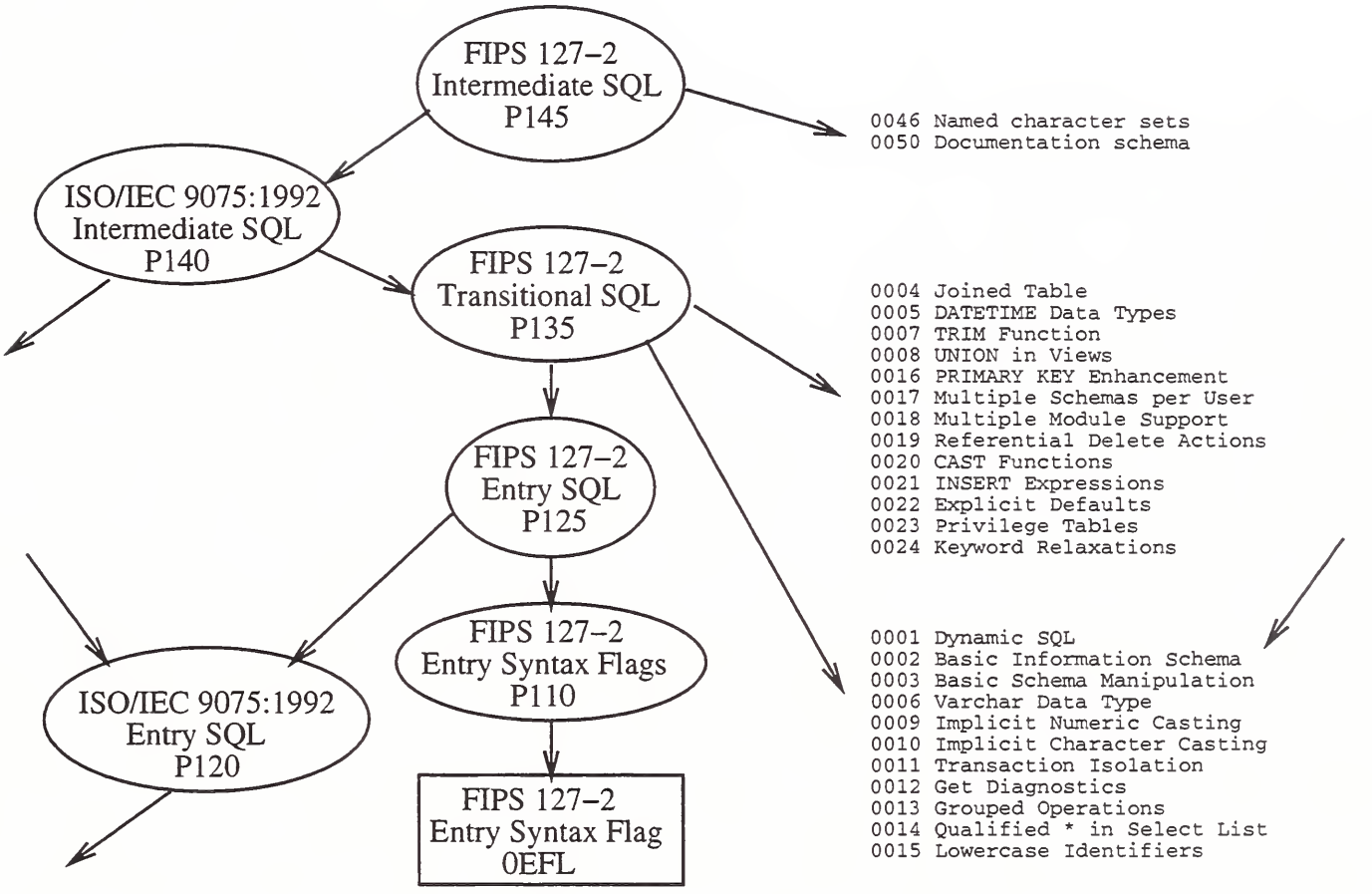
```
--  may not compile, may cause segmentation error, requires
--  visual inspection (ergo no pass/fail in TESTREPORT), etc.

   CREATE TABLE TESTCASE
     (TESTNO    CHAR(4) NOT NULL PRIMARY KEY,
      DESCR     CHAR(50) NOT NULL,
      PROG      CHAR(6) NOT NULL REFERENCES TESTPROG,
      T_NOTE    CHAR(10),
      ISQL_CT   DECIMAL(2) NOT NULL);

-- Each test is for one or more features.
-- This table describes the test cases in the programs.

   CREATE TABLE TESTFEATURE
     (TESTNO    CHAR(4) NOT NULL REFERENCES TESTCASE,
      FEATURE1  CHAR(4) NOT NULL REFERENCES REPORTFEATURE,
      PRIMARY KEY (TESTNO, FEATURE1));

-- On-line cross reference to SQL-92 (population deferred)
-- Sequence number is decimal to facilitate adding references
--  between existing references without renumbering.
-- [same idea as Dewey Decimal system used in libraries]

   CREATE TABLE TESTREFERENCE
     (TESTNO    CHAR(4) NOT NULL REFERENCES TESTCASE,
      SEQ       DECIMAL (6,4) NOT NULL,
      TESTREF   CHAR(50) NOT NULL,
      PRIMARY KEY (TESTNO, SEQ));

-- ========================================================
--   Tables to specify vendor claims follow:
-- ========================================================

-- Features supported for this testing:

   CREATE TABLE FEATURE_CLAIMED
     (FEATURE1  CHAR(4) NOT NULL UNIQUE REFERENCES REPORTFEATURE);

-- Bindings supported for this testing:

   CREATE TABLE BINDING_CLAIMED
     (BINDING1  CHAR(3) NOT NULL UNIQUE,
      CHECK (BINDING1 IN
          ('PCO','PFO','PC','PPA','PAD','PMU','PPL',
           'MCO','MFO','MC','MPA','MAD','MMU','MPL','SQL')));

-- ========================================================
--   Tables to generate vendor-specific requirements follow:
-- ========================================================
```

```
-- Features required, to be derived recursively,
--  including claim to be tested -- C1,
--            lowest reporting profile -- P1, and
--            lowest recursive link -- L1.
-- F1 is the feature to be tested.

   CREATE TABLE F_REQ
     (C1            CHAR(4) NOT NULL,
      P1            CHAR(4) NOT NULL,
      F1            CHAR(4) NOT NULL,
      LVL           INTEGER);

-- Working version of F_REQ,
--   needed because an INSERT cannot be self-referencing.

   CREATE TABLE F_TEMP
     (C1            CHAR(4) NOT NULL,
      P1            CHAR(4) NOT NULL,
      F1            CHAR(4),
      LVL           INTEGER);

   CREATE TABLE R_STRUCTURE
     (C1         CHAR(4) NOT NULL,
      P1         CHAR(4) NOT NULL,
      TESTNO     CHAR(4) NOT NULL,
      LVL        INTEGER);

-- Tests selected for this validation,
--   corresponding to the features selected and
--   corresponding to the bindings selected.
-- Result will be derived later from TESTREPORT.

   CREATE TABLE T_REQ
     (TESTNO     CHAR(4) NOT NULL,
      PROG       CHAR(6) NOT NULL,
      BINDING1   CHAR(3) NOT NULL,
      REQOPTNA   CHAR(3) NOT NULL,
      RESULT     CHAR(4));

COMMIT WORK;

========================END OF APPENDIX ============================
```

TEST REPORTING STRUCTURE



SQL Testing Profiles: FIPS SQL Levels
1996-12-31



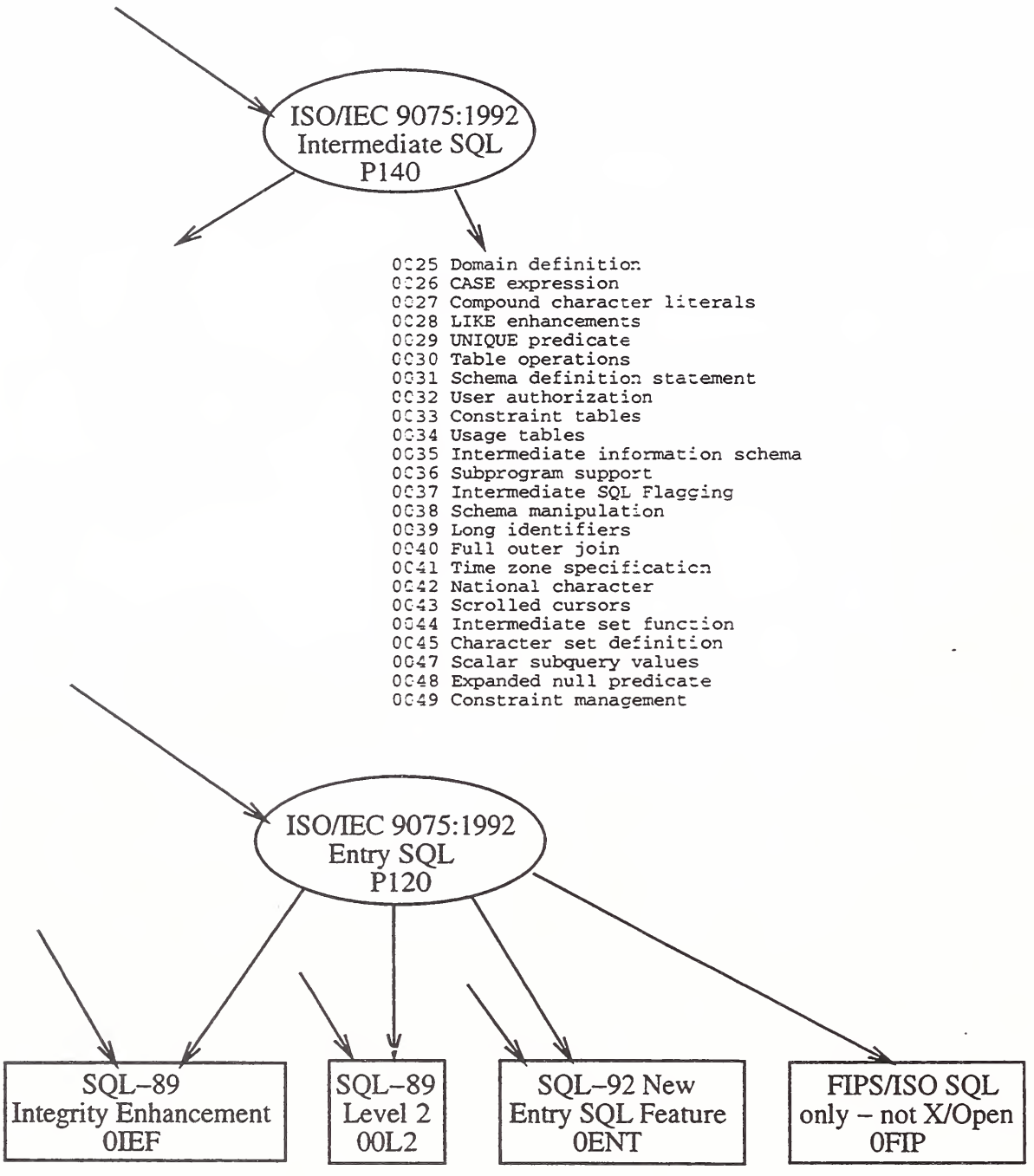
Analogous to Reporting Tables:

→ IMPLICATION

▭ ⬭ REPORTFEATURE

SQL Testing Profiles: ISO/IEC 9075:1992 Levels

1996-12-31



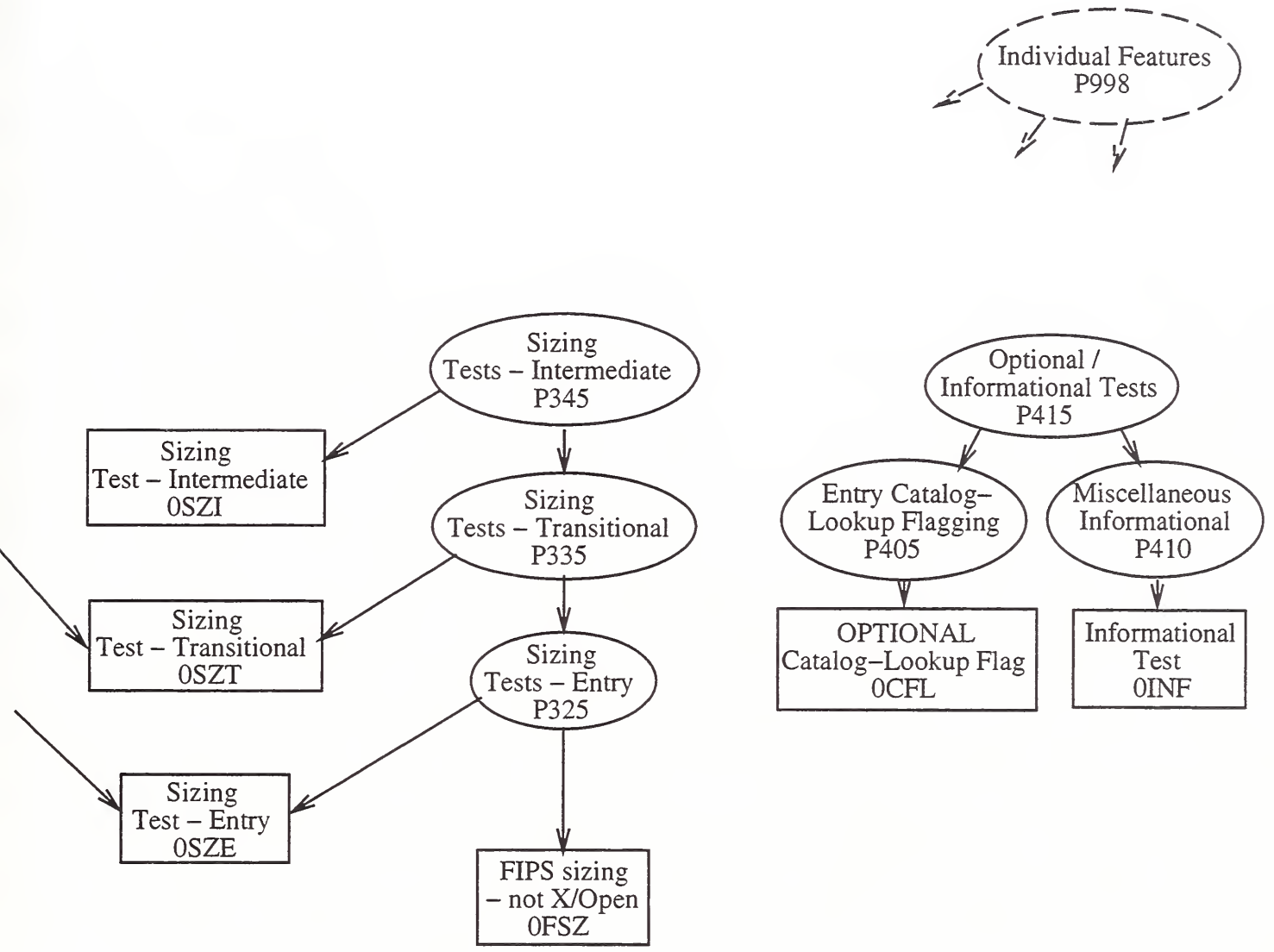
Analogous to Reporting Tables:

→ IMPLICATION

▭ ◯ REPORTFEATURE

SQL Testing Profiles: Sizing, Optionals, and Individual Features

1996–12–31



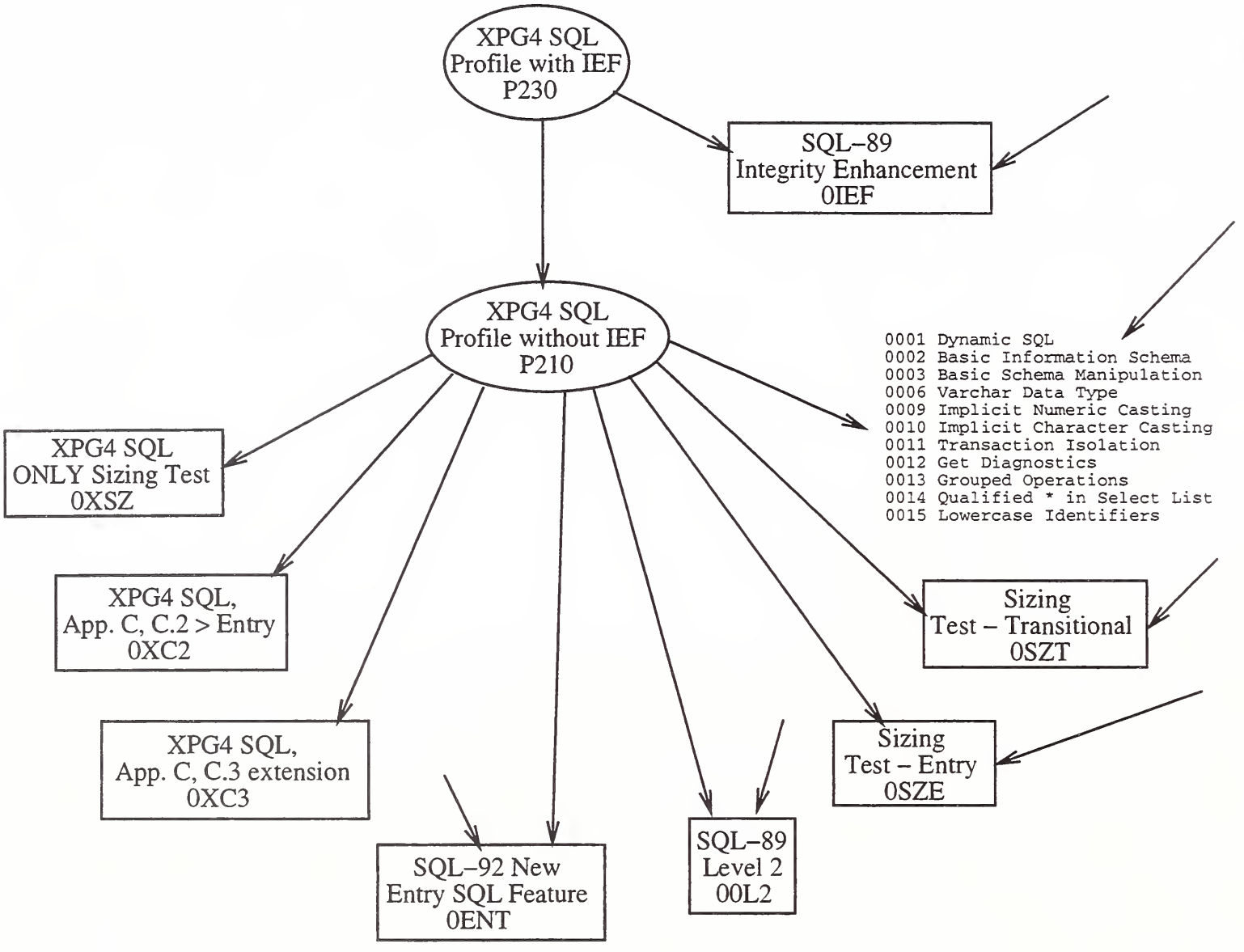
Analogous to Reporting Tables:

→ IMPLICATION

▭ ⬭ REPORTFEATURE

SQL Testing Profiles: X/Open

1996–12–31

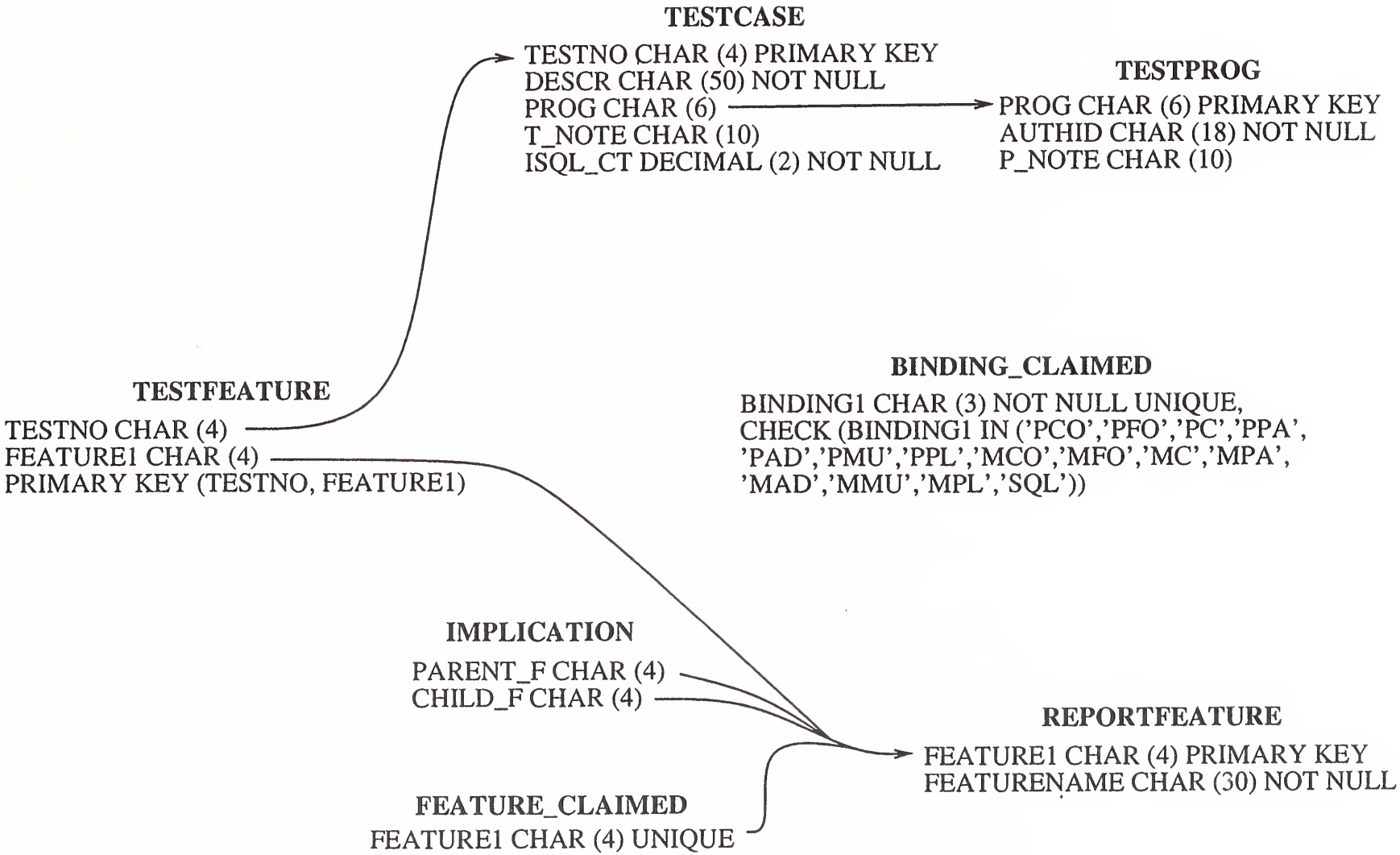


Analogous to Reporting Tables:

→ IMPLICATION

▭ ⬭ REPORTFEATURE

REPORTING SYSTEM TABLES



Informational Interactive Concurrency Test

Interactive SQL test files MPA001* and MPB001* through MPA008* and MPB008* have been superseded by the MPQUIC procedure, which is much more efficient. They are now considered optional. These files are still included in the Interactive SQL test suite because they may be helpful for SQL products which do not have a programming language interface. Each concurrency test has four parts. MPA*I.SQL initializes data in the test tables and is to be run first. MPA*T.SQL evaluates the test tables after the concurrency test and is to be run last. MPA*R.SQL and MPB*R.SQL are to be typed by two testers. These transactions are intended to be run concurrently. Print all of the MP*.SQL files and use them as worksheets to be filled out and retained as documentation. Instructions are provided for executing each test case.

In general, two testers are needed for each concurrency test, and the testers proceed as follows. For each concurrency test, one tester should run MPA*I.SQL. Then one tester should type as directed by the instructions in MPA*R.SQL while the other tester should type according to MPB*R.SQL. The testers should begin simultaneously. Then each tester should work at his own pace, and as the SQL implementation allows, given there will be some locking. There are explicit instructions for each test. If ROLLBACK statements are needed and it appears that no progress is being made by either tester, then after a tester types a fifth ROLLBACK statement for a given test, the testers should alternate transactions, rather than executing them concurrently, in order to complete the workload. Finally, one of the testers should run MPA*T.SQL.

It is permissible for the SQL implementation to issue error messages if its strategy for serializability detects circumstances where it cannot deliver serializable transactions. It is NOT permissible for the SQL implementation to execute non-serialized transactions without warning. Hence, the testers should be prepared to terminate (ROLLBACK) a transaction which encounters error or warning messages and to retry the transaction.

For Each Program Pair

Print the MP*.SQL files and use as worksheets.

Run the stored procedure file MPA*I.SQL to initialize.

Type or run the MPA*R.SQL and MPB*R.SQL tests as described below, following the instructions in the worksheet.

Testers may use "up arrows," function keys, and other shortcuts commonly used in the environment being tested.

Run the MPA*T.SQL file to evaluate the test results.

Concurrency Test #1

The first concurrency test has 2 simultaneous transactions attempting to insert a row, assigning the next available key number (MAX(KEYNUM) + 1).

Each tester, MPA and MPB, types a SELECT to determine the current maximum key and then an INSERT to create a new row with the next highest key number. Each tester records on the worksheet the ANUM values committed.

If SQL issues a warning message for either SQL command in the transaction, the tester types a ROLLBACK and repeats the transaction; otherwise, the tester types COMMIT. If all appears to go well for the COMMIT, the tester considers the transaction to have terminated successfully.

After 5 successful transactions, the tester waits for the other tester to finish.

Concurrency Test #2

The second concurrency test has 2 simultaneous transactions attempting to transfer money into the same account.

Each tester, MPA and MPB, types a SELECT statement to determine the dollar balance for row number 25. The tester then adds 10 to the dollar balance and types an UPDATE statement to set the balance to the new amount. Each tester records on the worksheet the DOLLARS values committed.

If SQL issues a warning message for either SQL command in the transaction, the tester types a ROLLBACK and repeats the transaction; otherwise, the tester types COMMIT. If all appears to go well for the COMMIT, the tester considers the transaction to have terminated successfully.

After 5 successful transactions, the tester waits for the other tester to finish.

Concurrency Test #3

The third concurrency test has 2 simultaneous transactions attempting to transfer money from one account to another.

Each tester, MPA and MPB, types two UPDATE statements to transfer money from a common account into separate accounts. If SQL issues a warning message for either SQL command in the transaction, the tester types a ROLLBACK and repeats the transaction; otherwise, the tester types COMMIT. If all appears to go well for the COMMIT, the tester considers the transaction to have terminated successfully. The tester then checks a box on the worksheet to count the successful transaction.

After 5 successful transactions, the tester waits for the other tester to finish.

Concurrency Test #4

The fourth concurrency test has 2 simultaneous transactions headed towards deadlock.

The two testers run their test files simultaneously. If SQL issues a warning message to a tester, then the tester types ROLLBACK and repeats the transaction; otherwise, the tester types COMMIT. If all appears to go well for the COMMIT, the tester stops.

These tests must be typed rather than run as a stored procedure if it appears that one transaction completes entirely before the other begins. There are instructions for shortening the test case if it is actually being typed.

If both testers deadlock twice, then MPB004R.SQL should wait until MPA004R.SQL completes a third attempt and commits before attempting again.

Concurrency Test #5

The fifth concurrency test determines whether uncommitted inserted rows are visible to another transaction.

The fifth concurrency test starts out with 25 rows in a table. MPA then alternately deletes and inserts batches of 5 rows until MPB has completed its workload. The MPB transaction alternately reads and counts the rows in the table. The MPB transaction expects to see results with either 20 or 25 rows.

If SQL issues a warning message for any SQL command in either of the transactions, the tester affected types a ROLLBACK and repeats the transaction; otherwise, the tester types COMMIT. If all appears to go well for the COMMIT, the tester considers the transaction to have terminated successfully.

After 6 successful typings of the MPB script, the MPB worksheet is completed and both testers may stop.

Concurrency Test #6

The sixth concurrency test is a rerun of the fourth test with the intention of reviewing the deadlock message, if any, to ensure that it notifies the use of transaction rollback or statement rejection (statement rollback), as appropriate.

The two testers type their test scripts simultaneously. If SQL issues a warning message to a tester, another query is typed to determine whether there was a statement rejection or a transaction rollback. The message must indicate which action is taken and must be accurate.

Concurrency Test #7

The seventh concurrency test starts with 1 row in table AA and 1 row in table BB. MPA attempts to transfer (READ/INSERT/DELETE) all rows from table AA to table BB; whereas MPB attempts to transfer all rows from table BB to table AA.

If SQL issues a warning message for any SQL command in the transaction, the tester types a ROLLBACK and repeats the transaction; otherwise, the tester types COMMIT. If all appears to go well for the COMMIT, the tester considers the transaction to have terminated successfully. The tester then checks a box on the worksheet to count the successful transaction.

After 2 successful transactions, the tester waits for the other tester to finish.

Concurrency Test #8

The eighth concurrency test determines whether MPB can insert a row with primary key which MPA has deleted, but not committed.

MPA deletes row with primary key value of 3. MPA then inserts a different row with primary key 3 and COMMITs. MPA repeats this scenario twice and should see no error messages. MPB meanwhile tries to insert a row with primary key value of 3. MPB should never be successful because the primary key value of 3 already exists. MPB notes error messages and quits after three attempts.