SAMeDL:
Technical Report Appendix C – Developmental Environment Test Plan

September 1992

This document has been approved for public release and sale; its distribution is unlimited.

AIRMICS
115 O'Keefe Building
Georgia Institute of Technology
Atlanta, GA 30332-0800
This report details the research efforts into the SQL Ada Module Database Description Language (SAMD/L). Four compilers are presented (Oracle, Informix, XDB, and Sybase) that allow Ada application programs to access database using a standard SQL query language. Copies of the compiler can be obtained from the DoD Ada Joint Program Office 703/614-0209.
This research was performed by Statistica Inc., contract number DAKF11-91-C-0035, for the Army Institute for Research in Management Information, Communications, and Computer Sciences (AIRMICS), the RDTE organization of the U. S. Army Information Systems Engineering Command (USAISEC). This final report discusses a set of SAMeDL compilers and work environment that were developed during the contract. Request for copies of the compiler can be obtained from the DoD Ada Joint Program Office, 703/614/0209. This research report is not to construed as an official Army or DoD Position, unless so designated by other authorized documents. Material included herein is approved for public release, distribution unlimited. Not protected by copyright laws.

**THIS REPORT HAS BEEN REVIEWED AND IS APPROVED**
APPENDIX C

SAMeDL Development Environment
Test Plan
SAMeDL Development Environment
Test Plan
# Table Of Contents

## Chapter 1
### About This Manual
1. Purpose .......................................................... 1
1. Organization .................................................. 1
1. References .................................................. 1

## Chapter 2
### Module Manager Testing Procedure
2.1 sde.clearlib .................................................. 3
2.2 sde.creatlib .................................................. 3
2.3 sde.creatar .................................................. 4
2.4 sde.ls .......................................................... 4
2.5 sde.mkscript .................................................. 5
2.6 sde.purge .................................................... 5
2.7 sde.rm ......................................................... 5
2.8 sde.rmlib ..................................................... 6

## Chapter 3
### Compiler Testing Procedure
3.1 Introduction .................................................. 7
3.2 Correct Tests .................................................. 7
3.3 End-to-End Tests .......................................... 7
3.4 Error Tests ................................................... 8

## Chapter 4
### Compiler Testing Cross Reference
4.1 Diagram 1
Correct Testing Cross Reference .................................. 10
4.2 Diagram 2
End-to-End Testing Cross Reference .......................... 13
4.3 Diagram 3
Error Testing Cross Reference .................................. 14

## Appendix A
### Compiler Test Suite Source Code
A.1 Correct Tests .................................................. 15
A.1.1 tl/ci.sme .................................................. 15
A.1.2 tl/cii.sme .................................................. 15
A.1.3 tl/ciii.sme .................................................. 17
A.1.4 tl/civ.sme .................................................. 19
A.1.5 tl/cv.sme ................................................... 23
A.1.6 tl/cvi.sme .................................................. 28
A.1.7 tl/cvii.sme .................................................. 32
A.1.8 tl/cviii.sme ............................................... 36
A.1.9 tl/cix.sme ................................................... 40
A.1.10 tl/ct1.sme ............................................... 44
A.1.11 tl/ct2.sme ............................................... 46
A.1.12 tl/ct3.sme ............................................... 47
A.1.13 tl/ct4.sme ............................................... 59
A.1.14 tl/ct5.sme ............................................... 60
A.1.15 tl/ct6.sme ............................................... 62
A.1.16 tl/ct7.sme ............................................... 63
A.1.17 tl/ct8.sme ............................................... 64
A.1.18 tl/ct9.sme ............................................... 65
A.1.19 tl/ct10.sme .............................................. 66
A.1.20 tl/ct11.sme................................. 66
A.1.21 tl/ct12.sme................................. 67
A.1.22 tl/ct13.sme................................. 68
A.1.23 tl/ct14.sme................................. 70
A.1.24 tl/ct15.sme................................. 72
A.1.25 tl/ct16.sme................................. 75
A.1.26 t3/t1.sme................................. 77
A.1.27 t3/t2.sme................................. 81
A.1.28 t3/t3.sme................................. 82
A.1.29 t3/t4.sme................................. 83
A.1.30 t3/t5.sme................................. 87
A.1.31 t3/t6.sme................................. 90
A.1.32 t3/t7.sme................................. 91
A.1.33 t3/t8.sme................................. 93
A.1.34 t3/t9.sme................................. 94
A.1.35 t3/t10.sme................................. 96
A.1.36 t3/t11.sme................................. 97
A.1.37 t3/t12.sme................................. 99
A.1.38 t3/t13.sme................................. 99
A.1.39 t3/t14.sme................................. 101
A.2 End-to-End Tests.......................... 101
A.2.1 t2/t2.sme................................. 101
A.2.2 t2/driver.a................................. 114
A.2.3 t2/initbank.sql........................... 137
A.2.4 t2/initi.sql................................. 138
A.3 Error Tests................................. 138
A.3.1 t1/et1.sme................................. 138
A.3.2 t1/et2.sme................................. 140
A.3.3 t1/et3.sme................................. 140
A.3.4 t1/et4.sme................................. 140
A.3.5 t1/et5.sme................................. 142
A.3.6 t1/et6.sme................................. 147
A.3.7 t1/et7.sme................................. 148
A.3.8 t1/et8.sme................................. 157
A.3.9 t1/et9.sme................................. 159
A.3.10 t1/et10.sme............................... 160
A.3.11 t1/et11.sme............................... 161
A.3.12 t1/et12.sme............................... 165
A.3.13 t3/e1.sme................................. 168
A.3.14 t3/e2.sme................................. 170
A.3.15 t3/e3.sme................................. 171
Chapter 1  About This Manual

1.1 Purpose

The purpose of this manual is to describe the test plan for the SAMeDL Development Environment (SDE), consisting of the SAMeDL compiler and the Module Manager. These tools and their features are documented in the SAMeDL Development Environment User Manual [User]. The language supported by the SAMeDL compiler is defined in the SAMeDL Language Reference Manual [LRM].

1.2 Organization

The organization of this document is as follows:

- Chapter 2, Module Manager Testing Procedure, outlines the testing process to be followed in testing the SDE Module Manager commands.
- Chapter 3, Compiler Testing Procedure, contains an overview of the testing strategy to be followed for the SAMeDL compiler.
- Chapter 4, Compiler Testing Cross Reference, provides a cross reference of compiler testing objectives (in terms of [LRM] section numbers) against test source files.
- Appendix A, Compiler Test Suite Source Code, contains a listing of the source code files that comprise the SAMeDL compiler test suite.

1.3 References


Chapter 2 Module Manager Testing Procedure

Testing of the Module Manager consists of invoking each of the commands (sde.cleanlib, sde.creatar, sde.creatlib, sde.ls, sde.mks_mscript, sde.purge, sde.rm, and sde.rmlib) in a variety of scenarios and manually inspecting the results. The process to be followed for each of the tools is outlined below.

2.1 sde.cleanlib

1. No arguments (i.e., default current directory).
2. Pathname argument.
3. Incorrect number of arguments (error).
4. Incorrect options (error).
5. No samedl.lib directory in current directory and call with no arguments (error).
6. No samedl.lib directory in specified pathname directory (error).
7. On a locked library (simulate by creating ./samedl.lib/samedl.lock) (error).
8. No permission to write in samedl.lib (error).
9. No permission to write in the directory containing samedl.lib. (error).

2.2 sde.creatlib

1. No arguments (i.e., default current directory).
2. Pathname argument.
3. Incorrect number of arguments (error).
4. Incorrect options (error).
5. samedl.lib directory already exists in current directory (error).
6. samedl.lib directory already exists in specified directory (error).
7. On a locked library (simulate by creating ./samedl.lib/samedl.lock) (error).
8. No permission to write in samedl.lib (error).
9. No permission to write in the directory where to create samedl.lib (error).
2.3 sde.creatar

1. No arguments (error).
2. Only archive name specified (error).
3. Only archive name and non-abstract module name specified (error).
4. Only archive name and abstract module name specified.
5. Add library Pathname argument.
6. Incorrect options (error).
7. samedl.lib directory does not exists in current directory (error).
8. samedl.lib directory does not exist in specified directory (error).
9. On a locked library (simulate by creating ./samedl.lib/samedl.lock) (error).
10. No permission to write in samedl.lib (error).
11. No permission to write in the directory where to create archive (error).
12. Non-existent modules (error).

2.4 sde.ls

1. No arguments (i.e., default current directory).
2. Pathname argument.
3. Incorrect number of arguments (error).
4. Incorrect options (error).
5. No samedl.lib directory in current directory and call with no arguments (error).
6. No samedl.lib directory in specified pathname directory (error).
7. On a locked library (simulate by creating ./samedl.lib/samedl.lock) (error).
8. No permission to write in samedl.lib (error).
9. No permission to write in the directory containing samedl.lib (error).
10. Different permutations of the -i -a and -v options (error).
11. Multiple module names as arguments (error).
12. Non-existent modules names as arguments (error).
2.5 sde.mkscript

1. No pathname argument (current directory default) + def/abs module name.
2. Pathname argument + def/abs module name
3. Incorrect number of arguments (error).
4. Incorrect options (error).
5. No samedl.lib directory in current directory and call with no arguments (error).
6. No samedl.lib directory in specified pathname directory (error).
7. On a locked library (simulate by creating ./samedl.lib/samedl.lock) (error).
8. No permission to write in samedl.lib (error).
9. No permission to write in the directory containing samedl.lib (error).
10. Multiple module names as arguments.
11. Non-existent modules as arguments (error).

2.6 sde.purge

1. No arguments (i.e., default current directory).
2. Pathname argument.
3. Incorrect number of arguments (error).
4. Incorrect options (error).
5. No samedl.lib directory in current directory and call with no arguments (error).
6. No samedl.lib directory in specified pathname directory (error).
7. On a locked library (simulate by creating ./samedl.lib/samedl.lock) (error).
8. No permission to write in samedl.lib (error).
9. No permission to write in the directory containing samedl.lib (error).

2.7 sde.rm

1. No pathname argument (current directory default) + module name.
2. Pathname argument + module name
3. Incorrect number of arguments (error).
4. Incorrect options (error).
SAMeDL Development Environment - Test Plan

5. No samedl.lib directory in current directory and call with no arguments (error).
6. No samedl.lib directory in specified pathname directory (error).
7. On a locked library (simulate by creating ./samedl.lib/samedl.lock) (error).
8. No permission to write in samedl.lib (error).
9. No permission to write in the directory containing samedl.lib (error).
10. Multiple module names as arguments.
11. Non-existent modules names as arguments (error).
12. Interactive option (test yes/no).

2.8 sde.rmlib

1. No arguments (i.e., default current directory).
2. Pathname argument.
3. Incorrect number of arguments (error).
4. Incorrect options (error).
5. No samedl.lib directory in current directory and call with no arguments (error).
6. No samedl.lib directory in specified pathname directory (error).
7. On a locked library (simulate by creating ./samedl.lib/samedl.lock) (error).
8. No permission to write in samedl.lib (error).
9. No permission to write in the directory containing samedl.lib (error).
Chapter 3  Compiler Testing Procedure

3.1 Introduction

Testing of the SAMeDL compiler will be divided into three basic areas:

1. Verify that the SAMeDL compiler recognizes and processes proper (as defined by [LRM]) syntactical and semantic constructs submitted to it. These tests will be known as the Correct Tests.

2. Verify that the output of the SAMeDL compiler will functionally (as defined by [LRM]) interface with the target database. These tests will be known as the End-to-End Tests.

3. Verify that the SAMeDL compiler identifies improper syntactical and semantic constructs (as defined by [LRM]) as errors. These tests will be known as the Error Tests.

3.2 Correct Tests

Proper syntactic and semantic constructs will be tested using a series of SAMeDL program modules that will contain all facets of the SAMeDL language described in the LRM. Diagram 1 in Chapter 4 shows the tests and which sections of the LRM are validated after the successful compilation of the program module.

A correct test is said to pass if:

1. The SAMeDL source code for the test can be compiled by the SAMeDL compiler without issuing an error message; and

2. Where interface files should be generated (see [LRM], [User]), the interface files are correctly generated and can be compiled without error by the appropriate compiler and/or pre-compiler (i.e., Ada compiler, C/ESQL pre-compiler, C compiler, Ada/SQL Module Language compiler) without error. (Note: depending on the specific configuration of the SAMeDL compiler, not all of the above compilers/pre-compilers may be applicable, or if applicable, may be invoked transparently by the SAMeDL compiler. Refer to [User] to determine the situation that applies.)

Otherwise, a correct test is said to fail.

3.3 End-to-End Tests

To test the output of the SAMeDL compiler for proper interfaces with the database management system, a program will be written in Ada that will exercise all the Procedure and Cursor definitions from the SAMeDL modules. Upon its execution, the program will initialize the database and start the testing procedures. The initialization and verification routines will set up the database for the following tests while testing the data structure interface and basic SAMeDL procedure and cursor functionality. The more complex cursor and procedure tests will then be run individually. The Ada application program will be written such that it is self-checking.

At the conclusion of each step of initialization and testing the driver will:
2. If the test has completed successfully, the driver will progress to the next test (if any).
3. If the test has failed, the driver will exit the testing procedure.

This area is covered by the single `t2` test (see Section A.2) which consists of 6 subtests for procedures and 11 subtests for cursors. Diagram 2 in Chapter 4 shows the cross-reference matrix of the tests to the portions of the LRM they are testing. LRM Chapters 2 through 4 are not explicitly tested for functionality, but are implicitly tested during the other tests.

The database design used in the test suite was based on an example database from [DSC]. It consists of 5 tables defining basic banking information:

<table>
<thead>
<tr>
<th>CUSTOMER TABLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SAVINGS_ACCOUNT TABLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Account #</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CHECKING_ACCOUNT TABLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Account #</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>LOAN_ACCOUNT TABLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Account #</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>BRANCH_INFO TABLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Branch ID</td>
</tr>
</tbody>
</table>

This design was chosen because the full range of SAMeDL data types and data manipulation statements could be implemented in a meaningful fashion.

3.4 Error Tests

An error test consists of a single SAMeDL source code file containing one or more errors; errors will be marked in the source code through SAMeDL comments. These tests will be said to pass if the marked errors are appropriately detected when compiled by the SAMeDL compiler.

Diagram 3 in Chapter 4 shows the cross-reference matrix of the error tests to the portions of the LRM they are testing.
Chapter 4  Compiler Testing Cross Reference

This chapter provides a cross reference of compiler testing objectives (in terms of [LRM] section numbers) against test source files.
### 4.1 Diagram 1: Correct Testing Cross Reference

<table>
<thead>
<tr>
<th>LRM Section</th>
<th>t1/cXXXX Tests</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>i</td>
</tr>
<tr>
<td>2.1</td>
<td>x</td>
</tr>
<tr>
<td>2.2</td>
<td>x</td>
</tr>
<tr>
<td>2.3</td>
<td>x</td>
</tr>
<tr>
<td>2.4</td>
<td>x</td>
</tr>
<tr>
<td>2.5</td>
<td>x</td>
</tr>
<tr>
<td>2.6</td>
<td>x</td>
</tr>
<tr>
<td>3.1</td>
<td>x</td>
</tr>
<tr>
<td>3.2</td>
<td>x</td>
</tr>
<tr>
<td>3.3</td>
<td></td>
</tr>
<tr>
<td>3.4</td>
<td>x</td>
</tr>
<tr>
<td>3.5</td>
<td>x</td>
</tr>
<tr>
<td>3.6</td>
<td></td>
</tr>
<tr>
<td>3.7</td>
<td></td>
</tr>
<tr>
<td>4.1</td>
<td></td>
</tr>
<tr>
<td>4.1.1</td>
<td></td>
</tr>
<tr>
<td>4.1.2</td>
<td></td>
</tr>
<tr>
<td>4.1.3</td>
<td></td>
</tr>
<tr>
<td>4.1.4</td>
<td></td>
</tr>
<tr>
<td>4.1.5</td>
<td></td>
</tr>
<tr>
<td>4.1.6</td>
<td></td>
</tr>
<tr>
<td>4.1.7</td>
<td></td>
</tr>
<tr>
<td>4.1.8</td>
<td></td>
</tr>
<tr>
<td>4.2</td>
<td></td>
</tr>
<tr>
<td>4.2.1</td>
<td></td>
</tr>
<tr>
<td>4.2.2</td>
<td></td>
</tr>
<tr>
<td>4.3</td>
<td></td>
</tr>
<tr>
<td>5.1</td>
<td></td>
</tr>
<tr>
<td>5.2</td>
<td></td>
</tr>
<tr>
<td>5.3</td>
<td></td>
</tr>
<tr>
<td>5.4</td>
<td></td>
</tr>
<tr>
<td>5.5</td>
<td></td>
</tr>
<tr>
<td>5.6</td>
<td></td>
</tr>
<tr>
<td>5.7</td>
<td></td>
</tr>
<tr>
<td>5.8</td>
<td></td>
</tr>
<tr>
<td>5.9</td>
<td></td>
</tr>
<tr>
<td>5.10</td>
<td></td>
</tr>
<tr>
<td>5.11</td>
<td></td>
</tr>
<tr>
<td>5.11.1</td>
<td></td>
</tr>
<tr>
<td>5.11.2</td>
<td></td>
</tr>
<tr>
<td>5.11.3</td>
<td></td>
</tr>
<tr>
<td>5.11.4</td>
<td></td>
</tr>
<tr>
<td>5.11.5</td>
<td></td>
</tr>
<tr>
<td>5.11.6</td>
<td></td>
</tr>
<tr>
<td>5.11.7</td>
<td></td>
</tr>
<tr>
<td>5.12</td>
<td></td>
</tr>
<tr>
<td>5.13</td>
<td></td>
</tr>
<tr>
<td>LRM Section</td>
<td>t9</td>
</tr>
<tr>
<td>-------------</td>
<td>----</td>
</tr>
<tr>
<td>2.1</td>
<td></td>
</tr>
<tr>
<td>2.2</td>
<td></td>
</tr>
<tr>
<td>2.3</td>
<td></td>
</tr>
<tr>
<td>2.4</td>
<td></td>
</tr>
<tr>
<td>2.5</td>
<td></td>
</tr>
<tr>
<td>2.6</td>
<td></td>
</tr>
<tr>
<td>3.1</td>
<td></td>
</tr>
<tr>
<td>3.2</td>
<td></td>
</tr>
<tr>
<td>3.3</td>
<td></td>
</tr>
<tr>
<td>3.4</td>
<td></td>
</tr>
<tr>
<td>3.5</td>
<td></td>
</tr>
<tr>
<td>3.6</td>
<td></td>
</tr>
<tr>
<td>3.7</td>
<td></td>
</tr>
<tr>
<td>4.1</td>
<td></td>
</tr>
<tr>
<td>4.1.1</td>
<td></td>
</tr>
<tr>
<td>4.1.2</td>
<td></td>
</tr>
<tr>
<td>4.1.3</td>
<td></td>
</tr>
<tr>
<td>4.1.4</td>
<td></td>
</tr>
<tr>
<td>4.1.5</td>
<td></td>
</tr>
<tr>
<td>4.1.6</td>
<td></td>
</tr>
<tr>
<td>4.1.7</td>
<td></td>
</tr>
<tr>
<td>4.1.8</td>
<td></td>
</tr>
<tr>
<td>4.2</td>
<td></td>
</tr>
<tr>
<td>4.2.1</td>
<td></td>
</tr>
<tr>
<td>4.2.2</td>
<td></td>
</tr>
<tr>
<td>4.3</td>
<td></td>
</tr>
<tr>
<td>5.1</td>
<td></td>
</tr>
<tr>
<td>5.2</td>
<td></td>
</tr>
<tr>
<td>5.3</td>
<td></td>
</tr>
<tr>
<td>5.4</td>
<td></td>
</tr>
<tr>
<td>5.5</td>
<td></td>
</tr>
<tr>
<td>5.6</td>
<td></td>
</tr>
<tr>
<td>5.7</td>
<td></td>
</tr>
<tr>
<td>5.8</td>
<td></td>
</tr>
<tr>
<td>5.9</td>
<td></td>
</tr>
<tr>
<td>5.10</td>
<td></td>
</tr>
<tr>
<td>5.11</td>
<td></td>
</tr>
<tr>
<td>5.11.1</td>
<td></td>
</tr>
<tr>
<td>5.11.2</td>
<td></td>
</tr>
<tr>
<td>5.11.3</td>
<td></td>
</tr>
<tr>
<td>5.11.4</td>
<td></td>
</tr>
<tr>
<td>5.11.5</td>
<td></td>
</tr>
<tr>
<td>5.11.6</td>
<td></td>
</tr>
<tr>
<td>5.11.7</td>
<td></td>
</tr>
<tr>
<td>5.12</td>
<td></td>
</tr>
<tr>
<td>5.13</td>
<td></td>
</tr>
</tbody>
</table>
### SAMeDL Development Environment - Test Plan

<table>
<thead>
<tr>
<th>LRM Section</th>
<th>t3/XXXX Tests</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>t1</td>
</tr>
<tr>
<td>2.1</td>
<td></td>
</tr>
<tr>
<td>2.2</td>
<td></td>
</tr>
<tr>
<td>2.3</td>
<td></td>
</tr>
<tr>
<td>2.4</td>
<td></td>
</tr>
<tr>
<td>2.5</td>
<td></td>
</tr>
<tr>
<td>2.6</td>
<td></td>
</tr>
<tr>
<td>3.1</td>
<td></td>
</tr>
<tr>
<td>3.2</td>
<td></td>
</tr>
<tr>
<td>3.3</td>
<td></td>
</tr>
<tr>
<td>3.4</td>
<td>x</td>
</tr>
<tr>
<td>3.5</td>
<td>x</td>
</tr>
<tr>
<td>3.6</td>
<td>x</td>
</tr>
<tr>
<td>3.7</td>
<td></td>
</tr>
<tr>
<td>4.1</td>
<td></td>
</tr>
<tr>
<td>4.1.1</td>
<td></td>
</tr>
<tr>
<td>4.1.2</td>
<td></td>
</tr>
<tr>
<td>4.1.3</td>
<td></td>
</tr>
<tr>
<td>4.1.4</td>
<td></td>
</tr>
<tr>
<td>4.1.5</td>
<td></td>
</tr>
<tr>
<td>4.1.6</td>
<td></td>
</tr>
<tr>
<td>4.1.7</td>
<td></td>
</tr>
<tr>
<td>4.1.8</td>
<td></td>
</tr>
<tr>
<td>4.2</td>
<td></td>
</tr>
<tr>
<td>4.2.1</td>
<td></td>
</tr>
<tr>
<td>4.2.2</td>
<td></td>
</tr>
<tr>
<td>4.3</td>
<td></td>
</tr>
<tr>
<td>5.1</td>
<td></td>
</tr>
<tr>
<td>5.2</td>
<td></td>
</tr>
<tr>
<td>5.3</td>
<td></td>
</tr>
<tr>
<td>5.4</td>
<td></td>
</tr>
<tr>
<td>5.5</td>
<td></td>
</tr>
<tr>
<td>5.6</td>
<td></td>
</tr>
<tr>
<td>5.7</td>
<td></td>
</tr>
<tr>
<td>5.8</td>
<td></td>
</tr>
<tr>
<td>5.9</td>
<td></td>
</tr>
<tr>
<td>5.10</td>
<td></td>
</tr>
<tr>
<td>5.11</td>
<td></td>
</tr>
<tr>
<td>5.11.1</td>
<td></td>
</tr>
<tr>
<td>5.11.2</td>
<td></td>
</tr>
<tr>
<td>5.11.3</td>
<td></td>
</tr>
<tr>
<td>5.11.4</td>
<td></td>
</tr>
<tr>
<td>5.11.5</td>
<td></td>
</tr>
<tr>
<td>5.11.6</td>
<td></td>
</tr>
<tr>
<td>5.11.7</td>
<td></td>
</tr>
<tr>
<td>5.12</td>
<td></td>
</tr>
<tr>
<td>5.13</td>
<td></td>
</tr>
</tbody>
</table>

---

*Intermetrics, Inc.*
## Chapter 4 - Compiler Testing Cross Reference

### 4.2 Diagram 2: End-to-End Testing Cross Reference

<table>
<thead>
<tr>
<th>LRM Section</th>
<th>Procedure Tests (PTxxxx)</th>
<th>Cursor Tests (CTxxxx)</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.1</td>
<td>x x x x x x x x x x x x x x x</td>
<td>x x x x x x x x x x x x x x x</td>
</tr>
<tr>
<td>5.2</td>
<td>x x x x x x x x x x x x x x x</td>
<td>x x x x x x x x x x x x x x x</td>
</tr>
<tr>
<td>5.3</td>
<td>x x x x x x x x x x x x x x x</td>
<td>x x x x x x x x x x x x x x x</td>
</tr>
<tr>
<td>5.4</td>
<td>x x x x x x x x x x x x x x x</td>
<td>x x x x x x x x x x x x x x x</td>
</tr>
<tr>
<td>5.5</td>
<td>x x x x x x x x x x x x x x x</td>
<td>x x x x x x x x x x x x x x x</td>
</tr>
<tr>
<td>5.6</td>
<td>x x x x x x x x x x x x x x x</td>
<td>x x x x x x x x x x x x x x x</td>
</tr>
<tr>
<td>5.7</td>
<td>x x x x x x x x x x x x x x x</td>
<td>x x x x x x x x x x x x x x x</td>
</tr>
<tr>
<td>5.8</td>
<td>x x x x x x x x x x x x x x x</td>
<td>x x x x x x x x x x x x x x x</td>
</tr>
<tr>
<td>5.9</td>
<td>x x x x x x x x x x x x x x x</td>
<td>x x x x x x x x x x x x x x x</td>
</tr>
<tr>
<td>5.10</td>
<td>x x x x x x x x x x x x x x x</td>
<td>x x x x x x x x x x x x x x x</td>
</tr>
<tr>
<td>5.11</td>
<td>x x x x x x x x x x x x x x x</td>
<td>x x x x x x x x x x x x x x x</td>
</tr>
<tr>
<td>5.11.1</td>
<td>x x x x x x x x x x x x x x x</td>
<td>x x x x x x x x x x x x x x x</td>
</tr>
<tr>
<td>5.11.2</td>
<td>x x x x x x x x x x x x x x x</td>
<td>x x x x x x x x x x x x x x x</td>
</tr>
<tr>
<td>5.11.3</td>
<td>x x x x x x x x x x x x x x x</td>
<td>x x x x x x x x x x x x x x x</td>
</tr>
<tr>
<td>5.11.4</td>
<td>x x x x x x x x x x x x x x x</td>
<td>x x x x x x x x x x x x x x x</td>
</tr>
<tr>
<td>5.11.5</td>
<td>x x x x x x x x x x x x x x x</td>
<td>x x x x x x x x x x x x x x x</td>
</tr>
<tr>
<td>5.11.6</td>
<td>x x x x x x x x x x x x x x x</td>
<td>x x x x x x x x x x x x x x x</td>
</tr>
<tr>
<td>5.11.7</td>
<td>x x x x x x x x x x x x x x x</td>
<td>x x x x x x x x x x x x x x x</td>
</tr>
<tr>
<td>5.12</td>
<td>x x x x x x x x x x x x x x x</td>
<td>x x x x x x x x x x x x x x x</td>
</tr>
<tr>
<td>5.13</td>
<td>x x x x x x x x x x x x x x x</td>
<td>x x x x x x x x x x x x x x x</td>
</tr>
</tbody>
</table>

*Intermetrics, Inc.*
### 4.3 Diagram 3: Error Testing Cross Reference

<table>
<thead>
<tr>
<th>LRM Section</th>
<th>t1/XXXX Tests</th>
<th>t3/XXXX Tests</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>et1</td>
<td>et2</td>
</tr>
<tr>
<td>2.1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.1.1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.1.2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.1.3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.1.4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.1.5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.1.6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.1.7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.1.8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.2.1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.2.2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5.1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5.2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5.3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5.4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5.5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5.6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5.7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5.8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5.9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5.10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5.11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5.11.1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5.11.2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5.11.3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5.11.4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5.11.5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5.11.6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5.11.7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5.12</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5.13</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Appendix A  Compiler Test Suite Source Code

A.1  Correct Tests

A.1.1  tl/ci.sme

```vhdl
-- ******************************************************************************
-- ** Test I
-- ******************************************************************************

DEFINITION MODULE D_cI IS

-- the previous line tests the newline separator

-- testing full character set

DOMIAN Character_set_domain IS
  NEW SQL_CHAR(length => 43);
CONSTANT letters : character_set_domain
  IS  'the quick brown fox jumps over the lazy dog';
CONSTANT all_caps : character_set_domain
  IS  'THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG';
CONSTANT digit_as_char : character_set_domain
  IS  '1234567890';
CONSTANT digits_as_num
  IS  1234567890;

DOMIAN integer_domain IS
  NEW SQL_INT;
DOMIAN real_domain IS
  NEW SQL_REAL;
CONSTANT integer literal : integer_domain
  IS  (12-4+5*2);
CONSTANT real literal : real_domain
  IS  12.456/0.09 + 1. ;
CONSTANT float literal
  IS  (0.1E1) + (10.E-1) + (.1E+1) ;

ENUMERATION Loan_types IS
  ( mortgage,
    auto,
    personal);
DOMIAN Loan_type_domain IS
  NEW SQL_ENUMERATION_AS_CHAR
  (ENUMERATION => Loan_types, MAP => IMAGE);
CONSTANT personal loan : loan_type_domain
  IS  personal ;

END D_cI;
```

A.1.2  tl/cci.sme

```vhdl
-- ******************************************************************************
-- ** Test II
-- ******************************************************************************
```
DEFINITION MODULE D_cii IS
-- enumeration declarations

-- ENUMERATION Branches IS
  ( Bethesda,
    Silver_Spring,
    Gaithersburg,
    Potomac);

ENUMERATION Loan_types IS
  ( mortgage,
    auto,
    personal);

-- domain character declarations

DOMAIN Customer_name_domain IS
  NEW SQLCHAR(length => 50);
DOMAIN SSN_domain IS
  NEW SQLCHAR NOT NULL (length => 9);
DOMAIN Addr_domain IS
  NEW SQLCHAR(length => 25);
DOMAIN City_domain IS
  NEW SQLCHAR(length => 25);
DOMAIN State_domain IS
  NEW SQLCHAR(length => 2);
DOMAIN Branch_name_domain IS
  NEW SQLCHAR(length => 25);

-- domain integer declarations

DOMAIN ZIP_code_domain IS
  NEW SQLINT( FIRST => 0, LAST => 999999999);
DOMAIN ZIP2_code_domain IS
  NEW SQLINT NOT NULL;
DOMAIN Account_number_domain IS
  NEW SQLSMALLINT( FIRST => 0, LAST => 9999);
DOMAIN Account2_number_domain IS
  NEW SQLSMALLINT NOT NULL;

-- domain real declarations

DOMAIN Balance_domain IS
  NEW SQL_REAL;
DOMAIN Interest_rate_domain IS
  NEW SQL_REAL( FIRST => 0.0, LAST => 1.0);
DOMAIN Loan_payment_domain IS
  NEW SQL_REAL NOT NULL;
DOMAIN Branch_assets_domain IS
  NEW SQL_REAL NOT NULL (FIRST => 0.0, LAST => 1.0E+10);

-- domain enumeration declarations

DOMAIN Loan_type_domain IS
  NEW SQL_ENUMERATION_AS_CHAR
    (ENUMERATION => Loan_types, MAP => IMAGE);
DOMAIN Loan2_type_domain IS
NEW SQL_ENUMERATION AS CHAR NOT NULL
(ENUMERATION => Loan_types, MAP => IMAGE);
DOMAIN Branch_number_domain IS
NEW SQL_ENUMERATION AS INT
(ENUMERATION => Branches, MAP => POS);
DOMAIN Branch2_number_domain IS
NEW SQL_ENUMERATION AS INT NOT NULL
(ENUMERATION => Branches, MAP => POS);

-- record definitions

RECORD Customer_record IS
  Cust_Name : Customer_name_domain;
  SSN : SSN_domain;
  Street : Addr_domain NOT NULL;
  City : City_domain;
  State : State_domain;
  ZIP : ZIP_code_domain;
END customer_record;

A.1.3 t1/ciiisme

DEFINITION MODULE D_cIII IS

-- enumeration declarations

ENUMERATION Branches IS
  (Bethesda,
   Silver_Spring,
   Gaithersburg,
   Potomac);

ENUMERATION Loan_types IS
  (mortgage,
   auto,
   personal);

-- domain character declarations

DOMAIN Customer_name_domain IS
  NEW SQL_CHAR(length => 50);
DOMAIN SSN_domain IS
  NEW SQL_CHAR NOT NULL (length => 9);
DOMAIN Addr_domain IS
  NEW SQL_CHAR(length => 25);
DOMAIN City_domain IS
  NEW SQL_CHAR(length => 25);
DOMAIN State_domain IS
  NEW SQL_CHAR(length => 25);
DOMAIN Branch_name_domain IS
  NEW SQL_CHAR(length => 25);
domain integer declarations

DOMAIN ZIP_code_domain IS
  NEW SQL_INT( FIRST => 0, LAST => 999999999);
DOMAIN ZIP2_code_domain IS
  NEW SQL_INT NOT NULL;
DOMAIN Account_number_domain IS
  NEW SQL_SMALLINT( FIRST => 0, LAST => 9999);
DOMAIN Account2_number_domain IS
  NEW SQL_SMALLINT NOT NULL;

domain real declarations

DOMAIN Balance_domain IS
  NEW SQL_REAL;
DOMAIN Interest_rate_domain IS
  NEW SQL_REAL( FIRST => 0.0, LAST => 1.0);
DOMAIN Loan_payment_domain IS
  NEW SQL_REAL NOT NULL;
DOMAIN Branch_assets_domain IS
  NEW SQL_REAL NOT NULL (FIRST => 0.0, LAST => 1.0E+10);

domain enumeration declarations

DOMAIN Loan_type_domain IS
  NEW SQL_ENUMERATION_AS_CHAR
    (ENUMERATION => Loantypes, MAP => IMAGE);
DOMAIN Loan2_type_domain IS
  NEW SQL_ENUMERATION_AS_CHAR NOT NULL
    (ENUMERATION => Loantypes, MAP => IMAGE);
DOMAIN Branch_number_domain IS
  NEW SQL_ENUMERATION_AS_INT
    (ENUMERATION => Branches, MAP => POS);
DOMAIN Branch2_number_domain IS
  NEW SQL_ENUMERATION_AS_INT NOT NULL
    (ENUMERATION => Branches, MAP => POS);

record definitions

RECORD Customer_record IS
  Cust_Name : Customer_name_domain;
  SSN : SSN_domain;
  Street : .Addr_domain NOT NULL;
  City : City_domain;
  State : State_domain;
  ZIP : ZIP_code_domain;
END customer_record;

END D_cIII;
WITH D_cIII;
USE D_cIII;
SCHEMA MODULE Tl_III IS

-- Basic customer information

TABLE Customer IS
  Cust_Name : Customer_name_domain,
SSN not null : SSN_domain,
Street_addr : Addr_domain,
City_addr : addr_domain,
State_addr : State_domain,
ZIP_addr : ZIP_code_domain
END Customer;

-- Savings account

TABLE Savings_account IS
  SBBranch_number : Branch_number_domain,
  SAccount_number : Account_number_domain,
  SBalance : Balance_domain,
  SCustomer__SSN not null : SSN_domain
END Savings_account;

-- Checking account

TABLE Checking_account IS
  CBranch_number : Branch_number_domain,
  CAccount_number : Account_number_domain,
  CBalance : Balance_domain,
  CCustomer__SSN not null : SSN_domain
END Checking_account;

-- loan account

TABLE loan_account IS
  LBranch_number : Branch_number_domain,
  LAccount_number : Account_number_domain,
  LBalance : Balance_domain,
  LPayment not null : Loan_Payment_domain,
  LCustomer__SSN not null : SSN_domain
END loan_account;

-- Branch information

TABLE Branch_info IS
  Branch_name : Branch_name_domain,
  Branch_number : Branch_number_domain,
  Assets not null : Branch_assets_domain
END Branch_info;

END T1_III;

A.1.4 t1/civ.sme

-- Assotechmul shi

-- *** Test IV

DEFINITION MODULE D_cIV IS

-- enumeration declarations

ENUMERATION Branches IS
  ( Bethesda,
    Silver_Spring,
ENUMERATION Loan_types IS
  ( mortgage,
    auto,
    personal);

domain character declarations

DOMAIN Customer_name_domain IS
  NEW SQL_CHAR(length => 50);
DOMAIN SSN_domain IS
  NEW SQL_CHAR NOT NULL (length => 9);
DOMAIN Addr_domain IS
  NEW SQL_CHAR(length => 25);
DOMAIN City_domain IS
  NEW SQL_CHAR(length => 25);
DOMAIN State_domain IS
  NEW SQL_CHAR(length => 2);
DOMAIN Branch_name_domain IS
  NEW SQL_CHAR(length => 25);

domain integer declarations

DOMAIN ZIP_code_domain IS
  NEW SQL_INT( FIRST => 0, LAST => 999999999);
DOMAIN ZIP2_code_domain IS
  NEW SQL_INT NOT NULL;
DOMAIN Account_number_domain IS
  NEW SQL_SMALLINT( FIRST => 0, LAST => 9999);
DOMAIN Account2_number_domain IS
  NEW SQL_SMALLINT NOT NULL;

domain real declarations

DOMAIN Balance_domain IS
  NEW SQL_REAL;
DOMAIN Interest_rate_domain IS
  NEW SQL_REAL( FIRST => 0.0, LAST => 1.0);
DOMAIN Loan_payment_domain IS
  NEW SQL_REAL NOT NULL;
DOMAIN Branch_assets_domain IS
  NEW SQL_REAL NOT NULL ( FIRST => 0.0, LAST => 1.0E+10);

domain enumeration declarations

DOMAIN Loan_type_domain IS
  NEW SQL_ENUMERATION_AS_CHAR
    (ENUMERATION => Loan_types, MAP => IMAGE);
DOMAIN Loan2_type_domain IS
  NEW SQL_ENUMERATION_AS_CHAR NOT NULL
    (ENUMERATION => Loan_types, MAP => IMAGE);
DOMAIN Branch_number_domain IS
  NEW SQL_ENUMERATION_AS_INT
    (ENUMERATION => Branches, MAP => POS);
DOMAIN Branch2_number_domain IS
  NEW SQL_ENUMERATION_AS_INT NOT NULL;
(ENUMERATION => Branches, MAP => POS);

-- record definitions

RECORD Customer_record IS
  Cust_Name : Customer_name_domain;
  SSN : SSN_domain;
  Street_addr : Addr_domain;
  City_addr : City_domain;
  State_addr : State_domain;
  ZIP_addr : ZIP_code_domain;
END customer_record;

END D_cIV;

WITH D_cIV;
USE D_cIV;
SCHEMA MODULE Ti_III IS

-- Basic customer information

TABLE Customer IS
  Cust_Name : Customer_name_domain,
  SSN not null : SSN_domain ,
  Street_addr : Addr_domain,
  City_addr : City_domain,
  State_addr : State_domain,
  ZIP_addr : ZIP_code_domain
END Customer;

-- Savings account

TABLE Savings_account IS
  SBranch_number : Branch_number_domain,
  SAccount_number : Account_number_domain ,
  SBalance : Balance_domain,
  SCustomer_SSN not null: SSN_domain
END ;

-- Checking account

TABLE Checking_account IS
  CBranch_number : Branch_number_domain,
  CAccount_number : Account_number_domain ,
  CBalance : Balance_domain,
  CCustomer_SSN not null: SSN_domain
END Checking_account;

-- loan account

TABLE loan_account IS
  LBranch_number : Branch_number_domain,
  LAccount_number : Account_number_domain ,
  LBalance : Balance_domain,
  LPayment not null: Loan_Payment_domain,
  LCustomer_SSN not null: SSN_domain
END loan_account;
-- Branch information

--

TABLE Branch_info IS
Branch_name : Branch_name_domain,
Branch_number : Branch_number_domain,
Assets not null : Branch_assets_domain
END Branch_info;

END T1_III;

WITH D_cIV;
USE D_cIV;
ABSTRACT MODULE A_cIV IS
AUTHORIZATION T1_III

ENUMERATION Bool IS
  ( true,
    false);

STATUS Stat_Map1 USES Bool IS
  (0 => true, 100 => false);

STATUS Stat_Map2 NAMED Stat_Map2_Renamed USES Bool IS
  (0 => true, 100 => false);

--
-- procedures
--

-- commit statement

PROCEDURE Commit_work IS
  COMMIT WORK STATUS Stat_Map1;

-- delete statement

PROCEDURE Delete_customer_loan (loan_number_in : Account_number_domain) IS
  DELETE FROM
    T1_III.Loan_account
  WHERE
    T1_III.Loan_account.LAccount_number = loan_number_in
  STATUS Stat_Map1 NAMED Stat_Map1_Renamed;

PROCEDURE Delete_customers IS
  DELETE FROM
    T1_III.customer
  STATUS Stat_Map2;

-- rollback statement

PROCEDURE rollback_work IS
  ROLLBACK WORK STATUS Stat_Map2 NAMED Standard_Map;

-- update statement

PROCEDURE Update_savings_account_balance
APPENDIX A - COMPILER TEST SUITE SOURCE CODE

(account_number_in NAMED acct : account_number_domain NOT NULL;
transaction : balance_domain )

IS
UPDATE
T1_III.savings_account
SET
T1_III.savings_account.Sbalance =
T1_III.savings_account.Sbalance + transaction
WHERE
T1_III.savings_account.Saccount_number = account_number_in;

PROCEDURE Savings_and_loan_transaction IS
UPDATE
T1_III.loan_account
SET
T1_III.loan_account.Lbalance = 0.0;
-- insert statement (query)

PROCEDURE move_checking_to_savings
(account_num_in : account_number_domain)
IS
INSERT INTO
T1_III.savings_account
SELECT *
FROM
T1_III.checking_account
WHERE
T1_III.checking_account.Caccount_number >= account_num_in;
-- insert statement (values)

PROCEDURE New_customer IS
INSERT INTO
T1_III.customer
FROM
New_customer_info : customer_record
VALUES;
-- select statement

PROCEDURE Get_customer_profile (SSN_in : SSN_domain) IS
SELECT *
INTO
Customer_Profile : customer_record
FROM
customer
WHERE
customer.SSN = SSN_in;

END A_cIV;

A.1.5 t1/cv.sme

-- **********************************************************************
-- *** Test V
-- **********************************************************************

Intermetrics, Inc.
DEFINITION MODULE D_cv IS
  -- enumeration declarations
  --
  ENUMERATION Branches IS
    (Bethesda,
     Silver_Spring,
     Gaithersburg,
     Potomac);
  ENUMERATION Loan_types IS
    (mortgage,
     auto,
     personal);
  -- domain character declarations
  --
  DOMAIN Customer_name_domain IS
    NEW SQLCHAR(length => 50);
  DOMAIN SSN_domain IS
    NEW SQLCHAR NOT NULL (length => 9);
  DOMAIN Addr_domain IS
    NEW SQLCHAR(length => 25);
  DOMAIN City_domain IS
    NEW SQLCHAR(length => 25);
  DOMAIN State_domain IS
    NEW SQLCHAR(length => 2);
  DOMAIN Branch_name_domain IS
    NEW SQL_CHAR(length => 25);
  -- domain integer declarations
  --
  DOMAIN ZIP_code_domain IS
    NEW SQLINT(FIRST => 0, LAST => 999999999);
  DOMAIN ZIP2_code_domain IS
    NEW SQLINT NOT NULL;
  DOMAIN Account_number_domain IS
    NEW SQL_SMALLINT(FIRST => 0, LAST => 9999);
  DOMAIN Account2_number_domain IS
    NEW SQL_SMALLINT NOT NULL;
  -- domain real declarations
  --
  DOMAIN Balance_domain IS
    NEW SQL_REAL;
  DOMAIN Interest_rate_domain IS
    NEW SQL_REAL(FIRST => 0.0, LAST => 1.0);
  DOMAIN Loan_payment_domain IS
    NEW SQL_REAL NOT NULL;
  DOMAIN Branch_assets_domain IS
    NEW SQL_REAL NOT NULL (FIRST => 0.0, LAST => 1.0E+10);
  -- domain enumeration declarations
  --
  DOMAIN Loan_type_domain IS
    NEW SQL_ENUMERATION_AS_CHAR
    (ENUMERATION => Loan_types, MAP => IMAGE);
DOMA IN Loan2_type_domain IS
  NEW SQL_ENUMERATION AS CHAR NOT NULL
  (ENUMERATION => Loan_types, MAP => IMAGE);
DOMA IN Branch_number_domain IS
  NEW SQL_ENUMERATION AS INT
  (ENUMERATION => Branches, MAP => POS);
DOMA IN Branch2_number_domain IS
  NEW SQL_ENUMERATION AS INT NOT NULL
  (ENUMERATION => Branches, MAP => POS);

-- record definitions

RECORD Customer_record IS
  Cust_Name : Customer_name_domain;
  SSN : SSN_domain;
  Street_addr : Addr_domain NOT NULL;
  City_addr : City_domain;
  State_addr : State_domain;
  ZIP_addr : ZIP_code_domain;
END customer_record;

END D_cV;

WITH D_cV AS Def_Mod;
USE Def_Mod;
SCHEMA MODULE T1_III IS
  -- Basic customer information
  TABLE Customer IS
    Cust_Name : Def_Mod.Customer_name_domain,
    SSN not null : SSN_domain,
    Street_addr : Addr_domain,
    City_addr : addr_domain,
    State_addr : State_domain,
    ZIP_addr : ZIP_code_domain
END Customer;

  -- Savings account
  TABLE Savings_account IS
    SBranch_number : Branch_number_domain,
    SAccount_number : Account_number_domain,
    SBalance : Balance_domain,
    SCustomer_SSN not null : SSN_domain
END ;

  -- Checking account
  TABLE Checking_account IS
    CBranch_number : Branch_number_domain,
    CAccount_number : Account_number_domain,
    CBalance : Balance_domain,
    CCustomer_SSN not null : SSN_domain
END Checking_account;

  -- loan account

Intermetrics, Inc. 25
TABLE loan_account IS
   LBranch_number : Branch_number_domain,
   LAccount_number : Account_number_domain,
   LBalance : Balance_domain,
   LPayment not null : Loan_Payment_domain,
   LCustomer_SSN not null : SSN_domain
END loan_account;

-- Branch information

TABLE Branch_info IS
   Branch_name : Branch_name_domain,
   Branch_number : Branch_number_domain,
   Assets not null : Branch_assets_domain
END Branch_info;

END T1_II;

WITH D_cv AS Def_Mod;
USE Def_Mod;
ABSTRACT MODULE A_cv IS
   AUTHORIZATION T1_II
   -- cursors
   --
   CURSOR List_customers FOR
   SELECT *
   FROM
   T1_II.Customer
   ORDER BY
   T1_II.Customer.SSN ;
   --
   -- cursors with different predicates in the WHERE statement
   --
   --
   -- compound comparison predicate =
   --
   CURSOR customer_accounts(SSN_in : SSN_domain) FOR
   SELECT
   T1_II.customer.cust_name,
   T1_II.customer.street_addr,
   T1_II.customer.city_addr,
   T1_II.customer.state_addr,
   T1_II.customer.ZIP_addr,
   T1_II.savings_account.Saccount_number,
   T1_II.savings_account.Sbalance,
   T1_II.checking_account.Caccount_number,
   T1_II.checking_account.Cbalance
   FROM
   T1_II.Customer,
   T1_II.Savings_account,
   T1_II.Checking_account
   WHERE
   T1_II.customer.ssn = ssn_in AND
   T1_II.savings_account.Scustomer_ssn = ssn_in AND
   T1_II.checking_account.Ccustomer_ssn = ssn_in ;
Appendix A - Compiler Test Suite Source Code

-- comparison predicate >=
--
-- CURSOR loans_over(loan_balance_in : balance_domain) FOR
SELECT
  T1_III.Loan_account.Laccount_number,
  T1_III.Loan_account.Lbranch_number,
  T1_III.Loan_account.Lcustomer_ssn,
  T1_III.Loan_account.Lbalance
FROM
  T1_III.Loan_account
WHERE
  T1_III.Loan_account.Lbalance >= loan_balance_in
;

-- comparison predicate <=
--
-- CURSOR loans_under(loan_balance_in : balance_domain) FOR
SELECT
  T1_III.Loan_account.Laccount_number,
  T1_III.Loan_account.Lbranch_number,
  T1_III.Loan_account.Lcustomer_ssn,
  T1_III.Loan_account.Lbalance
FROM
  T1_III.Loan_account
WHERE
  T1_III.Loan_account.Lbalance <= loan_balance_in
;

-- comparison predicate >
--
-- CURSOR checking_balance_over (account_bal_in : Balance_domain) FOR
SELECT
  T1_III.checking_account.Caccount_number,
  T1_III.checking_account.Ccustomer_ssn,
  T1_III.checking_account.Cbalance
FROM
  T1_III.checking_account
WHERE
  T1_III.checking_account.Cbalance > account_bal_in
;

-- comparison predicate <
--
-- CURSOR savings_balance_under (account_bal_in : Balance_domain) FOR
SELECT
  T1_III.savings_account.Saccount_number,
  T1_III.savings_account.Scustomer_ssn,
  T1_III.savings_account.Sbalance
FROM
  T1_III.savings_account
WHERE
  T1_III.savings_account.Sbalance < account_bal_in
;

Intermetrics, Inc.
-- comparison predicate <>

CURSOR other_branch_names (branch_name_in : branch_name_domain) FOR
SELECT
  T1_III.branch_info.branch_name
FROM
  T1_III.branch_info
WHERE
  T1_III.branch_info.branch_name <> branch_name_in

END A_cV;

A.1.6 t1/cvi.smce

-- ******************************************
-- *** Test VI
-- ******************************************

DEFINITION MODULE D_cVI IS

-- enumeration declarations

ENUMERATION Branches IS
  (Bethesda, Silver_Spring, Gaithersburg, Potomac);

ENUMERATION Loan_types IS
  (mortgage, auto, personal);

-- domain character declarations

DOMAIN Customer_name_domain IS
  NEW SQL_CHAR(length => 50);
DOMAIN SSN_domain IS
  NEW SQL_CHAR NOT NULL (length => 9);
DOMAIN Addr_domain IS
  NEW SQL_CHAR(length => 25);
DOMAIN City_domain IS
  NEW SQL_CHAR(length => 25);
DOMAIN State_domain IS
  NEW SQL_CHAR(length => 2);
DOMAIN Branch_name_domain IS
  NEW SQL_CHAR(length => 25);

-- domain integer declarations

DOMAIN ZIP_code_domain IS
  NEW SQL_INT( FIRST => 0, LAST => 999999999);
DOMAIN ZIP2_code_domain IS
  NEW SQL_INT NOT NULL;
DOMAIN Account_number_domain IS
  NEW SQL_SMALLINT( FIRST => 0, LAST => 9999);
NEW SQL_SMALLINT NOT NULL;
--
-- domain real declarations
--
DOMAIN Balance_domain IS
NEW SQL_REAL;
DOMAIN Interest_rate_domain IS
NEW SQL_REAL( FIRST => 0.0, LAST => 1.0);
DOMAIN Loan_payment_domain IS
NEW SQL_REAL NOT NULL;
DOMAIN Branch_assets_domain IS
NEW SQL_REAL NOT NULL ( FIRST => 0.0, LAST => 1.0E+10);
--
-- domain enumeration declarations
--
DOMAIN Loan_type_domain IS
NEW SQL_ENUMERATION AS CHAR
(ENUMERATION => Loan_types, MAP => IMAGE);
DOMAIN Loan2_type_domain IS
NEW SQL_ENUMERATION AS CHAR NOT NULL
(ENUMERATION => Loan_types, MAP => IMAGE);
DOMAIN Branch_number_domain IS
NEW SQL_ENUMERATION AS INT
(ENUMERATION => Branches, MAP => POS);
DOMAIN Branch2_number_domain IS
NEW SQL_ENUMERATION AS INT NOT NULL
(ENUMERATION => Branches, MAP => POS);
--
-- record definitions
--
RECORD Customer_record IS
Cust_Name : Customer_name_domain;
SSN : SSN_domain;
Street : Addr_domain NOT NULL;
City : City_domain;
State : State_domain;
ZIP : ZIP_code_domain;
END customer_record;
END D_cVI;

WITH D_cVI;
USE D_cVI;
SCHEMA MODULE Ti_III IS
--
-- Basic customer information
--
TABLE Customer IS
Cust_Name : Customer_name_domain,
SSN not null : SSN_domain,
Street_addr : Addr_domain,
City_addr : addr_domain,
State_addr : State_domain,
ZIP_addr : ZIP_code_domain
END Customer;
--
-- Checking account
--

Intermetrics, Inc.
TABLE Checking_account IS
  CBranch_number : Branch_number_domain,
  CAccount_number : Account_number_domain,
  CBalance : Balance_domain,
  CCustomer_SSN not null : SSN_domain
END Checking_account;
--
-- loan account
--
TABLE loan_account IS
  LBranch_number : Branch_number_domain,
  LAccount_number : Account_number_domain,
  LBalance : Balance_domain,
  LPayment not null : Loan_Payment_domain,
  LCustomer_SSN not null : SSN_domain
END loan_account;
--
-- Branch information
--
TABLE Branch_info IS
  Branch_name : Branch_name_domain,
  Branch_number : Branch_number_domain,
  Assets not null : Branch_assets_domain
END Branch_info;
END T1_III;

WITH D_cVI;
SCHEMA MODULE T1_III_2 IS
--
-- Savings account
--
TABLE Savings_account IS
  SBranch_number : D_cVI.Branch_number_domain,
  SAccount_number : D_cVI.Account_number_domain,
  SBalance : D_cVI.Balance_domain,
  SCustomer_SSN not null : D_cVI.SSN_domain
END;
END T1_III_2;

WITH D_cVI;
WITH SCHEMA T1_III_2;
USE D_cVI;
ABSTRACT MODULE A_cVI IS
  AUTHORIZATION T1_III
--
-- cursors
--
--
-- between predicate
--
CURSOR large_deposits FOR
  SELECT *
  FROM T1_III_2.savings_account
  WHERE T1_III_2.savings_account.Sbalance
BETWEEN AVG(T1_III_2.savings_account.Sbalance) AND MAX(T1_III_2.savings_account.Sbalance) 

-- not between predicate 

CURSOR large_loans FOR SELECT 
T1_III.loan_account.Laccount_number, 
T1_III.loan_account.Lcustomer_ssn, 
T1_III.loan_account.Lbalance 
FROM 
T1_III.loan_account 
WHERE 
T1_III.loan_account.Lbalance 
NOT BETWEEN AVG(T1_III.loan_account.Lbalance) AND MIN(T1_III.loan_account.Lbalance) 

-- in predicate 

CURSOR Loan_count (Branch_in: branch_number_domain) FOR SELECT * 
FROM 
T1_III.Loan_account 
WHERE 
T1_III.Loan_account.LBranch_number IN (Branch_in) 

-- not in predicate 

CURSOR customer_count FOR SELECT * 
FROM 
T1_III.customer 
WHERE 
T1_III.customer.ssn 
NOT IN (SELECT T1_III.loan_account.Lcustomer_ssn FROM T1_III.loan_account) 

-- like predicate 

CURSOR find_customer (name_in: customer_name_domain) FOR SELECT 
T1_III.customer.cust_name 
FROM 
T1_III.customer 
WHERE 
T1_III.customer.cust_name LIKE name_in 

-- null predicate 

Intermetrics, Inc.
CURSOR find_empty_account FOR
SELECT
    T1_III.checking_account.Caccount_number,
    T1_III.checking_account.Ccustomer_SSN
FROM
    T1_III.checking_account
WHERE
    T1_III.checking_account.Cbalance IS NULL
;

CURSOR find_joint_accounts FOR
SELECT
    T1_III_2.savings_account.Scustomer_ssn,
    T1_III_2.savings_account.Saccount_number,
    checking_account.Caccount_number
FROM
    T1_III_2.savings_account, T1_III.checking_account
WHERE
    EXISTS
        (SELECT *
         FROM
             T1_III_2.savings_account, T1_III.checking_account
         WHERE
             T1_III_2.savings_account.Scustomer_ssn =
             T1_III.checking_account.Ccustomer_ssn)
;
END A_cVI;

A.1.7 t1/cvii.sme

DEFINITION MODULE D_cVII IS
--
-- enumeration declarations
--

ENUMERATION Branches IS
    ( Bethesda,
      Silver_Spring,
      Gaithersburg,
      Potomac);

ENUMERATION Loan_types IS
    ( mortgage,
      auto,
      personal);

--
-- domain character declarations
--

DOMAIN Customer_name_domain IS
    NEW SQL_CHAR(length => 50);

DOMAIN SSN_domain IS
NEW SQL_CHAR NOT NULL (length => 9);
DOMAIN Addr_domain IS
  NEW SQL_CHAR(length => 25);
DOMAIN City_domain IS
  NEW SQL_CHAR(length => 25);
DOMAIN State_domain IS
  NEW SQL_CHAR(length => 2);
DOMAIN Branch_name_domain IS
  NEW SQL_CHAR(length => 25);

-- domain integer declarations

DOMAIN ZIP_codedomain IS
  NEW SQLINT( FIRST => 0, LAST => 999999999);
DOMAIN ZIP2_code_domain IS
  NEW SQLINT NOT NULL;
DOMAIN Account_numberdomain IS
  NEW SQLSMALLINT( FIRST => 0, LAST => 9999);
DOMAIN Account2_number_domain IS
  NEW SQLSMALLINT NOT NULL;

-- domain real declarations

DOMAIN Balancemotion IS
  NEW SQLREAL;
DOMAIN Interest_rate_domain IS
  NEW SQLREAL( FIRST => 0.0, LAST => 1.0);
DOMAIN Loan_payment_domain IS
  NEW SQLREAL NOT NULL;
DOMAIN Branch_assets_domain IS
  NEW SQL_REAL NOT NULL (FIRST => 0.0, LAST => 1.0E+10);

-- domain enumeration declarations

DOMAIN Loan_type_domain IS
  NEW SQLENUMERATION_AS_CHAR
    (ENUMERATION => Loan_types, MAP => IMAGE);
DOMAIN Loan2_type_domain IS
  NEW SQLENUMERATION_AS_CHAR NOT NULL
    (ENUMERATION => Loan_types, MAP => IMAGE);
DOMAIN Branch_number_domain IS
  NEW SQLENUMERATION_AS_INT
    (ENUMERATION => Branches, MAP => POS);
DOMAIN Branch2_number_domain IS
  NEW SQLENUMERATION_AS_INT NOT NULL
    (ENUMERATION => Branches, MAP => POS);

-- record definitions

RECORD Customer_record NAMED Cust_Rec_Renamed IS
  Cust_Name : Customer_name_domain;
  SSN : SSN_domain;
  Street : Addr_domain NOT NULL;
  City : City_domain;
  State : State_domain;
  ZIP : ZIP_code_domain;
END customer_record;
END D_cvII;

WITH D_cvII;
USE D_cvII;
SCHEMA MODULE Ti_III IS
--
-- Basic customer information
--
TABLE Customer IS
  Cust_Name : Customer_name_domain,
  SSN not null : SSN_domain,
  Street_addr : Addr_domain,
  City_addr : addr_domain,
  State_addr : State_domain,
  ZIP_addr : ZIP_code_domain
END Customer;
--
-- Savings account
--
TABLE Savings_account IS
  SBrancl\number : Branch_number_domain,
  SAccount_number : Account_number_domain,
  SBalance : Balance_domain,
  SCustomer_SSN not null : SSN_domain
END;
--
-- Checking account
--
TABLE Checking_account IS
  CBranch_number : Branch_number_domain,
  CAccount_number : Account_number_domain,
  CBalance : Balance_domain,
  CCustomer_SSN not null SSN_domain
END Checking_account;
--
-- Loan account
--
TABLE loan_account IS
  LBranch_number : Branch_number_domain,
  LAccount_number : Account_number_domain,
  LBalance : Balance_domain,
  LPayment not null : Loan_Payment_domain,
  LCustomer_SSN not null : SSN_domain
END loan_account;
--
-- Branch information
--
TABLE Branch_info IS
  Branch_name : Branch_name_domain,
  Branch_number : Branch_number_domain,
  Assets not null : Branch_assets_domain
END Branch_info;
END Ti_III;

WITH D_cvII;
USE D_cvII;
ABSTRACT MODULE A_cvII IS
AUTHORIZATION T1_HII

-- cursors

-- quantified predicate = ALL

CURSOR List_Bethesda_checking FOR
SELECT *
FROM
T1_HII.checking_account
WHERE
T1_HII.checking_account.Cbranch_number = ALL
-- (SELECT * -- GDT: Result expr can't be
(SELECT Cbranch_number
FROM T1_HII.checking_account
WHERE T1_HII.checking_account.Cbranch_number =
bethesda)
;
IS
procedure Open_Cursor IS OPEN;
END List_Bethesda_checking;

-- quantified predicate <> ALL

CURSOR checking_only FOR
SELECT T1_HII.checking_account.Ccustomer_ssn
FROM T1_HII.checking_account
WHERE
T1_HII.checking_account.Ccustomer_ssn
<> ALL (SELECT T1_HII.savings_account.Scustomer_ssn
FROM T1_HII.savings_account)
;
IS
procedure Open_Curs IS OPEN checking_only;
END checking_only;

-- quantified predicate > ALL

CURSOR large_checking FOR
SELECT *
FROM
T1_HII.checking_account
WHERE
T1_HII.checking_account.Cbalance >
ALL (SELECT T1_HII.savings_account.Sbalance
FROM T1_HII.savings_account)
;

-- quantified predicate < ALL

CURSOR small_checking FOR
SELECT *
FROM
T1_HII.checking_account

Intermetrics, Inc.
WHERE
    T1_III.checking_account.Cbalance <
    ALL (SELECT T1_III.savings_account.Sbalance
    FROM T1_III.savings_account)

-- quantified predicate >= ALL

-- CURSOR largest_savings FOR
    SELECT *
    FROM
        T1_III.savings_account
    WHERE
        T1_III.savings_account.Sbalance >=
        ALL (SELECT T1_III.savings_account.Sbalance
    FROM T1_III.savings_account)

-- quantified predicate <= ALL

-- CURSOR smallest_savings FOR
    SELECT *
    FROM
        T1_III.savings_account
    WHERE
        T1_III.savings_account.Sbalance <=
        ALL (SELECT T1_III.savings_account.Sbalance
    FROM T1_III.savings_account)

-- quantified predicate = ANY

-- CURSOR loan_and_save FOR
    SELECT
        T1_III.savings_account.Scustomer_ssn
    FROM
        T1_III.savings_account
    WHERE
        T1_III.savings_account.Scustomer_ssn = ANY
        (SELECT T1_III.loan_account.Lcustomer_ssn
    FROM T1_III.loan_account)

-- quantified predicate <= SOME

-- CURSOR all_checking FOR
    SELECT
        T1_III.checking_account.Ccustomer_ssn
    FROM
        T1_III.checking_account
    WHERE
        T1_III.checking_account.Cbalance <=
        SOME (SELECT T1_III.checking_account.Cbalance
    FROM T1_III.checking_account)

END A_cVII;
A.1.8 tl/cvii.sme

DEFINITION MODULE D_cvIII IS

-- enumeration declarations

ENUMERATION Branches IS
  (Bethesda,
   Silver_Spring,
   Gaithersburg,
   Potomac);

ENUMERATION Loan_types IS
  (mortgage,
   auto,
   personal);

-- domain character declarations

DOMAIN Customer_name_domain IS
  NEW SQL_CHAR(length => 50);
DOMAIN SSN_domain IS
  NEW SQL_CHAR NOT NULL (length => 9);
DOMAIN Addr_domain IS
  NEW SQL_CHAR(length => 25);
DOMAIN City_domain IS
  NEW SQL_CHAR(length => 25);
DOMAIN State_domain IS
  NEW SQL_CHAR(length => 2);
DOMAIN Branch_name_domain IS
  NEW SQL_CHAR(length => 25);

-- domain integer declarations

DOMAIN ZIP_code_domain IS
  NEW SQL_INT( FIRST => 0, LAST => 999999999);
DOMAIN ZIP2_code_domain IS
  NEW SQLINT NOT NULL;
DOMAIN Account_number_domain IS
  NEW SQLSMALLINT( FIRST => 0, LAST => 9999);
DOMAIN Account2_number_domain IS
  NEW SQL_SMALLINT NOT NULL;

-- domain real declarations

DOMAIN Balance_domain IS
  NEW SQL_REAL;
DOMAIN Interest_rate_domain IS
  NEW SQL_REAL( FIRST => 0.0, LAST => 1.0);
DOMAIN Loan_payment_domain IS
  NEW SQL_REAL NOT NULL;
DOMAIN Branch_assets_domain IS
  NEW SQL_REAL NOT NULL ( FIRST => 0.0, LAST => 1.0E+10);
-- domain enumeration declarations

--

DOMAIN Loan_type_domain IS
NEW SQL_ENUMERATION AS CHAR
(ENUMERATION => Loan_types, MAP => IMAGE);

DOMAIN Loan2_type_domain IS
NEW SQL_ENUMERATION AS CHAR NOT NULL
(ENUMERATION => Loan_types, MAP => IMAGE);

DOMAIN Branch_number_domain IS
NEW SQL_ENUMERATION AS INT
(ENUMERATION => Branches, MAP => POS);

DOMAIN Branch2_number_domain IS
NEW SQL_ENUMERATION AS INT NOT NULL
(ENUMERATION => Branches, MAP => POS);

-- record definitions

--

RECORD Customer_record NAMED Cust_Rec IS
  Cust_Name : Customer_name_domain;
  SSN : SSN_domain;
  Street : Addr_domain NOT NULL;
  City : City_domain;
  State : State_domain;
  ZIP : ZIP_code_domain;
END customer_record;

END D_cVIII;

WITH D_cVIII;
USE D_cVIII;
SCHEMA MODULE T1_III IS

-- Basic customer information

--

TABLE Customer IS
  Cust_Name : Customer_name_domain,
  SSN not null : SSN_domain ,
  Street_addr : Addr_domain ,
  City_addr : City_domain,
  State_addr : State_domain,
  ZIP_addr : ZIP_code_domain
END Customer;

-- Savings account

--

TABLE Savings_account IS
  SBranch_number : Branch_number_domain,
  SAccount_number : Account_number_domain ,
  SBalance : Balance_domain ,
  SCustomer_SSN not null : SSN_domain
END ;

-- Checking account

--

TABLE Checking_account IS
  CBranch_number : Branch_number_domain,
  CAccount_number : Account_number_domain ,
  CBalance : Balance_domain,
CCustomer_SSN not null : SSN_domain
END Checking_account;

-- loan account

-- TABLE loan_account IS
  LBranch_number : Branch_number_domain,
  LAccount_number : Account_number_domain,
  LBalance : Balance_domain,
  LPayment not null : Loan_Payment_domain,
  LCustomer_SSN not null : SSN_domain
END loan_account;

-- Branch information

-- TABLE Branch_info IS
  Branch_name : Branch_name_domain,
  Branch_number : Branch_number_domain,
  Assets not null : Branch_assets_domain
END Branch_info;

END T1_III;

WITH D_cVIII;
USE D_cVIII;
ABSTRACT MODULE A_cVIII IS
  AUTHORIZATION T1_III
  -- cursors
  -- cursor procs

  CURSOR customer_list FOR
  SELECT *
  FROM
  T1_III.customer
  ORDER BY
  T1_III.customer.ssn;
  IS
  PROCEDURE open_customer IS
  OPEN customer_list;
  END open_customer;

  PROCEDURE close_customer IS
  CLOSE customer_list;
  END close_customer;

  PROCEDURE fetch_customer IS
  FETCH customer_list INTO : new customer_record;
  END fetch_customer;

  PROCEDURE update_customer( new_name : customer_name_domain ;
      new_ssn : ssn_domain ;
      new_street : addr_domain ;
      new_city : city_domain ;
      new_state : state_domain ;
      new_zip : zip_code_domain ) IS
  UPDATE T1_III.customer
  SET T1_III.customer.cust_name = new_name,
Tl_III.customer.ssn = new_ssn,
Tl_III.customer.street_addr = new_street,
Tl_III.customer.city_addr = new_city,
Tl_III.customer.state_addr = new_state,
Tl_III.customer.zip_addr = new_zip
WHERE CURRENT OF customer_list;

PROCEDURE delete_customer IS
DELETE FROM Tl_III.customer;
END customer_list;
END A_cVIII;

A.1.9 t1/cix.sme

-- ******************************************
-- *** Test IX
-- ******************************************

DEFINITION MODULE D_cIX_1 IS
--
-- enumeration declarations
--

ENUMERATION Branches IS
  (Bethesda,
   Silver_Spring,
   Gaithersburg,
   Potomac);

ENUMERATION Loan_types IS
  (mortgage,
   auto,
   personal);

--
-- domain character declarations
--

DOMAIN Customer_name_domain IS
  NEW SQL_CHAR(length => 50);
DOMAIN SSN_domain IS
  NEW SQL_CHAR NOT NULL (length => 9);
DOMAIN Addr_domain IS
  NEW SQL_CHAR(length => 25);
DOMAIN City-domain IS
  NEW SQL_CHAR(length => 25);
DOMAIN State_domain IS
  NEW SQL_CHAR(length => 2);
DOMAIN Branch_name_domain IS
  NEW SQL_CHAR(length => 25);
END D_cIX_1;

with D_cIX_1; use D_cIX_1;

DEFINITION MODULE D_cIX_2 IS
--
-- domain integer declarations
--

DOMAIN ZIP_code_domain IS
  NEW SQL_INT( FIRST => 0, LAST => 999999999);
Appendix A - Compiler Test Suite Source Code

DOMAIN ZIP2_code_domain IS
  NEW SQL_INT NOT NULL;
DOMAIN Account_number_domain IS
  NEW SQL_SMALLINT( FIRST => 0, LAST => 9999);
DOMAIN Account2_number_domain IS
  NEW SQL_SMALLINT NOT NULL;

-- domain real declarations

DOMAIN Balance_domain IS
  NEW SQL_REAL;
DOMAIN Interest_rate_domain IS
  NEW SQL_REAL( FIRST => 0.0, LAST => 1.0);
DOMAIN Loan_payment_domain IS
  NEW SQL_REAL NOT NULL;
DOMAIN Branch_assets_domain IS
  NEW SQL_REAL NOT NULL ( FIRST => 0.0, LAST => 1.0E+10);

-- domain enumeration declarations

DOMAIN Loan_type_domain IS
  NEW SQL_ENUMERATION AS CHAR
  (ENUMERATION => Loan_types, MAP => IMAGE);
DOMAIN Loan2_type_domain IS
  NEW SQL_ENUMERATION AS CHAR NOT NULL
  (ENUMERATION => Loan_types, MAP => IMAGE);
DOMAIN Branch_number_domain IS
  NEW SQL_ENUMERATION AS INT
  (ENUMERATION => Branches, MAP => POS);
DOMAIN Branch2_number_domain IS
  NEW SQL_ENUMERATION AS INT NOT NULL
  (ENUMERATION => Branches, MAP => POS);

-- record definitions

RECORD Customer_record IS
  Cust_Name : Customer_name_domain;
  SSN : SSN_domain;
  Street : Addr_domain;
  City : City_domain;
  State : State_domain;
  ZIP : ZIP_code_domain;
END customer_record;

END D_cIX_2;

WITH D_cIX_1, D_cIX_2;
USE D_cIX_1, D_cIX_2;
SCHEMA MODULE S_cIX_1 IS

-- Basic customer information

TABLE Customer IS
  Cust_Name : Customer_name_domain,
  SSN not null : SSN_domain,
  Street_addr : Addr_domain,
  City_addr : City_domain,
  State_addr : State_domain,
ZIP_addr : ZIP_code_domain
END Customer;

-- Savings account

-- TABLE Savings_account IS
    SBranch_number : Branch_number_domain,
    SAccount_number : Account_number_domain,
    SBalance : Balance_domain,
    SCustomer_SSN not null : SSN_domain
END ;

-- Checking account

-- TABLE Checking_account IS
    CBranch_number : Branch_number_domain,
    CAccount_number : Account_number_domain,
    CBalance : Balance_domain,
    CCustomer_SSN not null : SSN_domain
END Checking_account;

END S_cIX_1;

WITH D_cIX_1,D_cIX_2;
USE D_cIX_1,D_cIX_2;
SCHEMA MODULE S_cIX_2 IS

-- loan account

-- TABLE loan_account IS
    LBranch_number : Branch_number_domain,
    LAccount_number : Account_number_domain,
    LBalance : Balance_domain,
    LPayment not null : Loan_Payment_domain,
    LCustomer_SSN not null : SSN_domain
END loan_account;

-- Branch information

-- TABLE Branch_info IS
    Branch_name : Branch_name_domain,
    Branch_number : Branch_number_domain,
    Assets not null : Branch_assets_domain
END Branch_info;

END S_cIX_2;

WITH D_cIX_1,D_cIX_2;
USE D_cIX_1; USE D_cIX_2;
WITH SCHEMA S_cIX_2;
ABSTRACT MODULE A_cIX_1 IS
    AUTHORIZATION S_cIX_1

-- procedures

--

-- commit statement

--
PROCEDURE Commit_work IS
  COMMIT WORK;
--
-- delete statement
--
PROCEDURE Delete_customer_loan (loan_number_in : Account_number_domain) IS
  DELETE FROM
    S_cIX_2.Loan_account
  WHERE
    S_cIX_2.Loan_account.Laccount_number = loan_number_in;
PROCEDURE Delete_customers IS
  DELETE FROM
    S_cIX_1.Customer;
--
-- rollback statement
--
PROCEDURE rollback_work IS
  ROLLBACK WORK;
--
-- update statement
--
PROCEDURE Update_savings_account_balance
  (account_number_in : account_number_domain;
    transaction : balance_domain)
IS
  UPDATE
    S_cIX_1.Savings_account
  SET
    S_cIX_1.Savings_account.Sbalance
      = S_cIX_1.Savings_account.Sbalance + transaction
  WHERE
    S_cIX_1.Savings_account.Saccount_number = account_number_in;
END A_cIX_1;
WITH D_cIX_1, D_cIX_2;
USE D_cIX_1;
USE D_cIX_2;
WITH SCHEMA S_cIX_1;
ABSTRACT MODULE A_cIX_2 IS
  AUTHORIZATION S_cIX_2
PROCEDURE Savings_and_loan_transaction IS
  UPDATE
    S_cIX_2.loan_account
  SET
    S_cIX_2.loan_account.Lbalance = 0.0;
--
-- insert statement (query)
--
PROCEDURE move_checking_to_savings
  (account_num_in : account_number_domain)
IS
  INSERT INTO
    S_cIX_1.savings_account
  SELECT *
FROM
SAMeDL Development Environment - Test Plan

```
S_cIX_1.checking_account
WHERE
S_cIX_1.checking_account.Caccount_number >= account_num_in;
--
-- insert statement (values)
--
PROCEDURE New_customer IS
  INSERT INTO S_cIX_1.Customer
  FROM New_customer_info : new cust_record
  VALUES;
--
-- select statement
--
PROCEDURE Get_customer_profile (SSN_in : SSN_domain) IS
  SELECT *
  INTO Customer_Profile : customer_record
  FROM S_cIX_1.Customer
  WHERE S_cIX_1.Customer.SSN = SSN_in;
END A_cIX_2;

A.1.10  t1/ctl.sme

definition module d_ctl is
-- Member Information
  domain MemName is new SQL_CHAR Not Null (Length => 30);
  domain SSN is new SQL_CHAR Not Null (Length => 9);
  domain Age is new SQL_SMALLINT (FIRST => 1, LAST => 199);
  enumeration SexEnum is (F, M);
  domain Sex is new SQL_ENUMERATION_AS_INT (MAP => POS, ENUMERATION => SexEnum);
  domain Phone is new SQL_CHAR (Length => 8);
  domain Street is new SQL_CHAR (Length => 30);
  domain City is new SQL_CHAR (Length => 15);
  domain County is new SQL_CHAR Not Null (Length => 2);
  domain Club_Number is new SQL_SMALLINT Not Null;
  domain Sum_Domain is new SQL_SMALLINT Not Null;
  domain Count_Domain is new SQL_INT;
end d_ctl;

with d_ctl; use d_ctl;
schema module s_recdb is
  table Members is
    MemberName not null : MemName,
    MemberSSN not null : SSN,
    ClubNumber not null : Club_Number,
```

Intermetrics, Inc.
Appendix A - Compiler Test Suite Source Code

```
MemberAge : Age,
MemberSex : Sex,
MemberPhone : Phone,
MemberStreet : Street,
MemberCity : City,
MemberCnty not null : County
end Members;
end s_recdb;

with d_ctl; use d_ctl;
abstract module a_ctl is
  authorization s_recdb

  record MemberRec is
    R_MemberName : MemName;
    R_Sum : Sum_Domain;
    R_Count : Count_Domain;
  end;

  procedure P_MemberSelect (Req_MemberSSN : SSN) is
    select MemberName, SUM(MemberAge), COUNT(*)
    into Row : MemberRec
    from s_recdb.Members
    where s_recdb.Members.MemberSSN = Req_MemberSSN;

  procedure MP_MemberSelect (Req_MemberSSN : SSN) is
    select MemberName, Sum_Domain(SUM(MemberAge)),
    Count_Domain(COUNT(*))
    into Row : MemberRec
    from s_recdb.Members
    where s_recdb.Members.MemberSSN = Req_MemberSSN;

  procedure MPD_MemberSelect (Req_MemberSSN : SSN) is
    select MemberName, Sum_Domain(SUM(MemberAge)) named foo,
    Count_Domain(COUNT(*)) named ct
    from s_recdb.Members
    where s_recdb.Members.MemberSSN = Req_MemberSSN;

  cursor M_MemberSelect (Req_MemberSSN : SSN) for
    select MemberName, Sum_Domain(SUM(MemberAge)) named foo,
    Count_Domain(COUNT(*)) named ct
    from s_recdb.Members
    where s_recdb.Members.MemberSSN = Req_MemberSSN;
    is
    procedure FetchIt is
      fetch into Row : new MemRec;
    end M_MemberSelect;

  cursor MD_MemberSelect (Req_MemberSSN : SSN) for
    select MemberName, Sum_Domain(SUM(MemberAge)) named foo,
    Count_Domain(COUNT(*)) named ct
    from s_recdb.Members
    where s_recdb.Members.MemberSSN = Req_MemberSSN;
    is
    procedure FetchIt is
      fetch;
```

Intermetrics, Inc.
end MD_MemberSelect;
end a_ct1;

A.1.11 tl/ct2.sme

-- This test is the simple demo (input.sme)

definition module d_ct2 is
  -- Member Information
  domain MemName is new SQL_CHAR Not Null (Length => 30);
  domain SSN is new SQL_CHAR Not Null (Length => 9);
  domain Age is new SQL_SMALLINT (FIRST => 1, LAST => 199);

  enumeration SexEnum is (F, M);
  domain Sex is new SQL_ENUMERATION_AS_INT (MAP => POS, ENUMERATION => SexEnum);

  domain Phone is new SQL_CHAR (Length => 8);
  domain Street is new SQL_CHAR (Length => 30);
  domain City is new SQL_CHAR (Length => 15);
  domain County is new SQL_CHAR Not Null (Length => 2);
  domain ClubNumber is new SQL_SMALLINT Not Null;
end d_ct2;

with d_ct2; use d_ct2;

schema module s_ct2 is
  table Members is
    MemberName not null : MemName,
    MemberSSN not null : SSN,
    ClubNumber not null : ClubNumber,
    MemberAge : Age,
    MemberSex : Sex,
    MemberPhone : Phone,
    MemberStreet : Street,
    MemberCity : City,
    MemberCnty not null : County
  end Members;
end s_ct2;

with d_ct2; use d_ct2;

abstract module a_ct2 is
  authorization s_ct2
  record MemberRec is
    MemberName : MemName;
    MemberSSN : SSN;
    ClubNumber : ClubNumber;
    MemberAge : Age;
    MemberSex : Sex;
    MemberPhone : Phone;
    MemberStreet : Street;
end record MemberRec;

MemberCity : City;
MemberCnty : County;
end;

procedure CommitWork is
  commit work;
end;

procedure MemberInsert is
  insert into s_ct2.Members
  from Row : MemberRec
  values;

cursor MemberSelect (Req_MemberSSN : SSN) for
  select *
  from s_ct2.Members
  where s_ct2.Members.MemberSSN = Req_MemberSSN;

procedure FetchIt is
  fetch into Row : new MemRec;
end MemberSelect;
end a_ct2;

A.1.12 t1/ct3.sme

-- The big demo test (T2)

DEFINITION MODULE d_ct3 IS
--
-- enumeration declarations
--
ENUMERATION Branches IS
  ( Bethesda,
    Silver_Spring,
    Gaithersburg,
    Potomac);

ENUMERATION Loan_types IS
  ( mortgage,
    auto,
    personal);

--
-- domain character declarations
--
DOMAIN Customer_name_domain IS
  NEW SQL_CHAR(length => 15);
DOMAIN Addr_domain IS
  NEW SQL_CHAR(length => 15);
DOMAIN City_domain IS
  NEW SQL_CHAR(length => 15);
DOMAIN State_domain IS
  NEW SQL_CHAR(length => 2);

--
-- domain integer declarations
--
DOMAIN SSN_domain IS
  NEW SQL_INT NOT NULL ( FIRST => 0, LAST => 999999999);
DOMAIN acct_num_domain IS
    NEW SQL_SMALLINT NOT NULL (FIRST => 0, LAST => 9999);

--

domain real declarations

DOMAIN Balance_domain IS
    NEW SQL_REAL;
DOMAIN Interest_rate_domain IS
    NEW SQL_REAL( FIRST => 0.0, LAST => 1.0);
DOMAIN Loan_payment_domain IS
    NEW SQL_REAL;
DOMAIN Branch_assets_domain IS
    NEW SQL_REAL;
domain enumeration declarations

DOMAIN Loan_type_domain IS
  NEW SQL_ENUMERATION AS_int
  (MAP => POS, ENUMERATION => Loan_types);

DOMAIN branch_num_domain IS
  NEW SQL_ENUMERATION AS_INT
  (MAP => POS, ENUMERATION => Branches);

record definitions

RECORD Customer_record IS
  Cust_Name : Customer_name_domain;
  SSN : SSN_domain;
  Street : Addr_domain;
  City : City_domain;
  State : State_domain;
END customer_record;

RECORD Savings_entry IS
  branch_num : branch_num_domain;
  acct_num : acct_num_domain;
  Balance : Balance_domain;
  cust_ssn : SSN_domain;
END Savings_entry;

RECORD Chequeing_entry IS
  branch_num : branch_num_domain;
  acct_num : acct_num_domain;
  Balance : Balance_domain;
  cust_ssn : SSN_domain;
END Chequeing_entry;

RECORD loan_entry IS
  branch_num : branch_num_domain;
  acct_num : acct_num_domain;
  Balance : Balance_domain;
  Loan_type : Loan_type_domain;
  cust_ssn : SSN_domain;
END loan_entry;

RECORD Branch_entry IS
  branch_num : branch_num_domain;
  Assets : Branch_assets_domain;
END Branch_entry;

END d_ct3;
WITH d_ct3;
USE d_ct3;
SCHEMA MODULE s_ct3 IS
  -- Basic customer information
  --
  TABLE Cust IS
    Cust_Name : Customer_name_domain,
    SSN not null : SSN_domain,
    Street_addr : Addr_domain,
    City_addr : City_domain,
    State_addr : State_domain
  END cust;

  -- Checking account
  --
  TABLE cheque IS
    branch_num : branch_num_domain,
    acct_num not null : acct_num_domain,
    Balance : Balance_domain,
    cust_ssn not null : SSN_domain
  END cheque;

  -- Savings account
  --
  TABLE Save IS
    branch_num : branch_num_domain,
    acct_num not null : acct_num_domain,
    Balance : Balance_domain,
    cust_ssn not null : SSN_domain
  END Save;

  -- Loan account
  --
  TABLE loan IS
    branch_num : branch_num_domain,
    acct_num not null : acct_num_domain,
    Balance : Balance_domain,
    Loan_type : loan_type_domain,
    cust_ssn not null : SSN_domain
  END loan;

  -- Branch information
  --
  TABLE Branch IS
    num : branch_num_domain,
    Assets : Branch_assets_domain
  END Branch;

END s_ct3;
WITH d_ct3;
USE d_ct3;
ABSTRACT MODULE a_ct3 IS
  AUTHORIZATION s_ct3
  --
  -- procedures
  --
  -- commit statement
  --
  PROCEDURE Commit_work IS
    COMMIT WORK;
  --
  -- delete statement
  --
  PROCEDURE Delete_customer_loan
    (loan_number_in : acct_num_domain) IS
    DELETE FROM
      s_ct3.Loan
    WHERE
      s_ct3.Loan.acct_num = loan_number_in;
  --
  -- rollback statement
  --
  PROCEDURE rollback_work IS
    ROLLBACK WORK;
  --
  -- update statement
  --
  PROCEDURE Up_save_acct_bal
    (acct_num_in : acct_num_domain;
     transaction : balance_domain )
  IS
    UPDATE
      s_ct3.save
    SET
      s_ct3.save.balance =
      s_ct3.save.balance + transaction
    WHERE
      s_ct3.save.acct_num = acct_num_in;
  PROCEDURE S_and_L IS
    UPDATE
      s_ct3.Loan
    SET
      s_ct3.Loan.balance = 0.0;
  --
  -- insert statement (query)
  --
  PROCEDURE move_cheque_to_save
    (account_num_in : acct_num_domain)
  IS
    INSERT INTO
      s_ct3.save
    SELECT *
    FROM

Intermetrics, Inc.
s_ct3.cheque
WHERE
  s_ct3.cheque.acct_num >= account_num_in;
PROCEDURE Get_cust_profile (SSN_in : SSN_domain) IS
  SELECT *
  INTO Customer_Profile : customer_record
  FROM s_ct3.cust
  WHERE s_ct3.cust.SSN = SSN_in;

PROCEDURE Get_saverecord (acctnum_in : acct_num_domain) IS
  SELECT *
  INTO savings_record : savings_entry
  FROM s_ct3.save
  WHERE s_ct3.save.acct_num = acctnum_in;

CURSOR customer_accounts(SSN_in : SSN_domain) FOR
  SELECT s_ct3.save.cust_ssn,
         s_ct3.save.acct_num,
         s_ct3.save.balance
  FROM s_ct3.save
  WHERE s_ct3.save.cust_ssn = SSN_in
-- comparison predicate >=
--
CURSOR loans_over(loan_balance_in : balance_domain) FOR
SELECT
  s_ct3.Loan.acct_num,
  s_ct3.Loan.branch_num,
  s_ct3.Loan.cust_ssn,
  s_ct3.Loan.balance
FROM
  s_ct3.Loan
WHERE
  s_ct3.Loan.balance >= loan_balance_in
;

-- comparison predicate <=
--
CURSOR loans_under(loan_balance_in : balance_domain) FOR
SELECT
  s_ct3.Loan.acct_num,
  s_ct3.Loan.branch_num,
  s_ct3.Loan.cust_ssn,
  s_ct3.Loan.balance
FROM
  s_ct3.Loan
WHERE
  s_ct3.Loan.balance <= loan_balance_in
;

-- comparison predicate >
--
CURSOR cheque_bal_over(account_bal_in : Balance_domain) FOR
SELECT
  s_ct3.cheque.acct_num,
  s_ct3.cheque.balance
FROM
  s_ct3.cheque
WHERE
  s_ct3.cheque.balance > account_bal_in
;

-- comparison predicate <
--
CURSOR save_bal_under(account_bal_in : Balance_domain) FOR
SELECT
  s_ct3.save.acct_num,
  s_ct3.save.balance
FROM
  s_ct3.save
WHERE
  s_ct3.save.balance < account_bal_in
Appendix A - Compiler Test Suite Source Code

```
-- comparison predicate <>

CURSOR other_branches
  ( branch_num_in : branch_num_domain ) FOR
  SELECT
    s_ct3.Branch.num
  FROM
    s_ct3.Branch
  WHERE
    s_ct3.Branch.num <> branch_num_in
;

-- between predicate

CURSOR large_deposits
  ( lower_bound : balance_domain; upper_bound : balance_domain ) FOR
  SELECT *
  FROM
    s_ct3.save
  WHERE
    s_ct3.save.balance
    BETWEEN lower_bound
    AND upper_bound
;

-- not between predicate

CURSOR large_loans
  ( lower_bound : balance_domain; upper_bound : balance_domain ) FOR
  SELECT
    s_ct3.Loan.acct_num,
    s_ct3.Loan.balance,
    s_ct3.Loan.cust_ssn
  FROM
    s_ct3.Loan
  WHERE
    s_ct3.Loan.balance NOT BETWEEN lower_bound AND upper_bound
;
```

Intermetrics, Inc.
CURSOR find_customer (name_in : customer_name_domain) FOR
  SELECT
    s_ct3.cust.cust_name
  FROM
    s_ct3.cust
  WHERE
    s_ct3.cust.cust_name LIKE name_in
;

CURSOR Loan_count (Branch_in : branch_num_domain) FOR
  SELECT *
  FROM
    s_ct3.Loan
  WHERE
    s_ct3.Loan.Branch_num IN (Branch_in)
;

CURSOR customer_list FOR
  SELECT *
  FROM
    s_ct3.cust
;
IS
PROCEDURE open_customer IS
  OPEN customer_list;

PROCEDURE close_customer IS
  CLOSE customer_list;

PROCEDURE fetch_customer IS
  FETCH customer_list INTO next_customer : new c_record;

PROCEDURE update_customer (new_street : Addr_domain) IS
  UPDATE s_ct3.cust
  SET s_ct3.cust.street_addr = new_street
  WHERE CURRENT OF customer_list;

PROCEDURE delete_customer IS
  DELETE FROM s_ct3.cust;

END customer_list;
procedures and cursors used to initialize the database and
verify the contents of tables after test transactions

PROCEDURE New_customer IS
  INSERT INTO
    s_ct3.cust
  FROM
    New_customer_info : new_cust_record
  VALUES;

PROCEDURE New_chequeing IS
  INSERT INTO
    s_ct3.cheque
  FROM
    New_chequeing_info : chequeing_entry
  VALUES;

PROCEDURE New_savings IS
  INSERT INTO
    s_ct3.save
  FROM
    New_savings_info : savings_entry
  VALUES;

PROCEDURE New_loan IS
  INSERT INTO
    s_ct3.Loan
  FROM
    New_loan_info : loan_entry
  VALUES;

PROCEDURE New_branch IS
  INSERT INTO
    s_ct3.Branch
  FROM
    New_branch_info : new_b_entry
  VALUES;

PROCEDURE Delete_customers IS
  DELETE FROM
    s_ct3.cust;

PROCEDURE Delete_chequeing IS
  DELETE FROM
    s_ct3.cheque;

PROCEDURE Delete_savings IS
  DELETE FROM
    s_ct3.save;

PROCEDURE Delete_loans IS
  DELETE FROM
    s_ct3.Loan;

PROCEDURE Delete_Branches IS
DELETE FROM
    s_ct3.Branch;
CURSOR List_customers FOR
SELECT *
FROM s_ct3.cust
ORDER BY s_ct3.cust.SSN;

CURSOR List_chequeing FOR
SELECT *
FROM s_ct3.cheque
ORDER BY s_ct3.cheque.acct_num;

CURSOR List_savings FOR
SELECT *
FROM s_ct3.save
ORDER BY s_ct3.save.acct_num;

CURSOR List_loans FOR
SELECT *
FROM s_ct3.Loan
ORDER BY s_ct3.Loan.acct_num;

CURSOR List_branches FOR
SELECT *
FROM s_ct3.Branch
ORDER BY s_ct3.Branch.num;

END a_ct3;

A.1.13 t1/ct4.sme

-- Test named as phrases on status declarations
-- Test generation of status code and param names

definition module d_ct4 is
  domain MemName is new SQL_CHAR Not Null (Length => 30);
  [enumeration SQL_Code_Enum is (Ok, Bad, WhoKnows);
  status SQL_Status1 uses SQL_Code_Enum is (
    0 => ok,
    1 => bad,
    3 => WhoKnows);
  status SQL_Status2 named Status2 uses SQL_Code_Enum is (}
SAMeDL Development Environment - Test Plan

0 => ok,  
1 => bad,  
3 => WhoKnows);  

end d_ct4;

with d_ct4; use d_ct4;
schema module s_ct4 is  
  table Members is  
    MemberName not null : MemName  
  end Members;  
end s_ct4;

with d_ct4; use d_ct4;
abstract module a_ct4 is  
  authorization s_ct4
  procedure CommitWork1_1 is  
    commit work  
      status SQL_Status1;
    procedure CommitWork1_2 is  
      commit work  
        status SQL_Status1 named CW1_2;
    procedure CommitWork2_1 is  
      commit work  
        status SQL_Status2;
    procedure CommitWork2_2 is  
      commit work  
        status SQL_Status2 named CW2_2;
end a_ct4;

A.1.14  

t1/ct5.sme  

-- Tests AS PHRASES on context clauses

definition module d_ct5 is  
  -- Member Information  
  domain MemName is new SQL_CHAR Not Null (Length => 30);  
  domain SSN is new SQL_CHAR Not Null (Length => 9);  
  domain Age is new SQL_SMALLINT (FIRST => 1, LAST => 199);  
  enumeration SexEnum is (F, M);  
  domain Sex is new SQL_ENUMERATION_AS_INT (  
    MAP => POS, ENUMERATION => SexEnum);  
  domain Phone is new SQL_CHAR (Length => 8);  
  domain Street is new SQL_CHAR (Length => 30);  
  domain City is new SQL_CHAR (Length => 15);  
  domain County is new SQL_CHAR Not Null (Length => 2);  
  domain Club_Number is new SQL_SMALLINT Not Null;
with d_ct5 as SM;
use SM;
definition module d_ct5_2 is
  constant C_Name : SM.MemName is '123456789012345678901234567890';
  constant C_SSN : SM.SSN is '123456789';
  constant C_Club_Number : SM.Club_Number is 10;
  constant C_Age : SM.Age is 39;
  constant C_Sex : SM.Sex is SM.F;
  constant C_Phone : SM.Phone is '123456789';
  constant C_Street : SM.Street is '123456789012345678901234567890';
  constant C_City : SM.City is '123456789012345678901234567890';
end d_ct5_2;

with d_ct5 as SM;
use SM;
schema module s_ct5 is
  table Members is
    MemberName not null : MemName,
    MemberSSN not null : SSN,
    ClubNumber not null : Club_Number,
    MemberAge : Age,
    MemberSex : Sex,
    MemberPhone : Phone,
    MemberStreet : Street,
    MemberCity : City,
    MemberCnty not null : County
end Members;

  table Members2 is
    MemberName2 not null : SM.MemName,
    MemberSSN2 not null : SM.SSN,
    ClubNumber2 not null : SM.Club_Number,
    MemberAge2 : SM.Age,
    MemberSex2 : SM.Sex,
    MemberPhone2 : SM.Phone,
    MemberStreet2 : SM.Street,
    MemberCity2 : SM.City,
    MemberCnty2 not null : SM.County
end Members2;
end s_ct5;

with d_ct5 as SM;
use SM;
schema module Qs_ct5 is
  table QMembers is
    QMemberName not null : MemName,
    QMemberSSN not null : SSN,
    QClubNumber not null : Club_Number,
    QMemberAge : Age,
    QMemberSex : Sex,
    QMemberPhone : Phone,
    QMemberStreet : Street,
end QMembers;

Intermetrics, Inc.
SAMeDL Development Environment - Test Plan

QMemberCity : City,
QMemberCnty not null : County
end QMembers;

end Qs_ct5;

with d_ct5; use d_ct5;
with schema Qs_ct5 as QQ;
abstract module a_ct5 is
  authorization s_ct5
    procedure CommitWork1_1 is
      commit work;
    end;

    procedure MemberInsert is
      insert into QQ.QMembers
      values;
    end a_ct5;

A.1.15 t1/ct6.sme

DEFINITION MODULE D_ct6 IS
  DOMAIN Character_set_domain IS
    NEW SQL_CHAR(length => 43);
  DOMAIN integer_domain IS
    NEW SQL_INT;
  DOMAIN sm_integer_domain IS
    NEW SQL_SMALLINT;
  DOMAIN real_domain IS
    NEW SQL_REAL;

  ENUMERATION Loan_types IS
    ( mortgage,
      auto,
      personal);
  DOMAIN Loan_type_domain IS
    NEW SQL_ENUMERATION_AS_CHAR
      (ENUMERATION => Loan_types, map => image);

  CONSTANT personal_loan : loan_type_domain
    IS personal;

  CONSTANT one IS 1;
  CONSTANT Dmone IS integer_domain(one);
  CONSTANT one_dot_zero IS 1.0;
  CONSTANT one_float IS 10.0E-1;

  CONSTANT Done : sm_integer_domain IS 1;
  CONSTANT D_Dmone : integer_domain IS integer_domain(one);
  CONSTANT D_one_dot_zero : real_domain IS 1.0;
  CONSTANT D_one_float : real_domain IS 10.0E-1;

  CONSTANT u1 IS -1;
  CONSTANT u2 IS +1;
  CONSTANT u3 IS -1.0;
  CONSTANT u4 IS +1.0;
CONSTANT u5 IS -1.0E+00;
CONSTANT u6 IS +1.0E-00;

CONSTANT p1 IS (-1);
CONSTANT p2 IS (+1);
CONSTANT p3 : real_domain IS (-1.0);
CONSTANT p4 IS (+1.0);
CONSTANT p5 IS (-1.0E+00);
CONSTANT p6 : real_domain IS (+1.0E-00);

CONSTANT a1 IS 1+(-1);
CONSTANT a2 IS p1+(+1);
CONSTANT a3 IS 1.0E0+(-1.0);
CONSTANT a4 IS p3+(+1.0);
CONSTANT a5 IS 1.0+(-1.0E+00);
CONSTANT a6 IS p5+(+1.0E-00);
CONSTANT a7 IS real_domain(1.0) + real_domain((-1));

CONSTANT a8 IS p6 + real_domain((+1));
CONSTANT a9 IS real_domain(1.0E0) + real_domain((-1));
CONSTANT a0 IS real_domain(p3) + real_domain((+1));

CONSTANT s1 IS 1-(-1);
CONSTANT s2 IS p1-(-1);
CONSTANT s3 IS 1.0E0-(-1.0);
CONSTANT s4 IS p3-(-1.0);
CONSTANT s5 IS 1.0-(-1.0E+00);
CONSTANT s6 IS p5-(-1.0E-00);
CONSTANT s7 IS real_domain(1.0) - real_domain((-1));
CONSTANT s8 IS -p6 - real_domain((+1));
CONSTANT s9 IS real_domain(1.0E0) - real_domain((-1));
CONSTANT s0 IS real_domain(p3) - real_domain((-1));

CONSTANT m1 IS (4 - (-6))*(+5 + 5);
CONSTANT d1 IS (4 - (-6))/(+5 + 5);
CONSTANT m2 IS (4.0 - (-6.0))*(+5 + 5);
CONSTANT d2 IS (4.0 - (-6.0))/(+5 + 5);
CONSTANT m3 IS (4 - (-6))*(+5.0 + 5.0);
CONSTANT d3 IS real_domain(4 - (-6))/real_domain(+5.0 + 5.0);
CONSTANT m4 IS (4.0 - (-6.0))*(+5.0 + 5.0);
CONSTANT d4 IS (4.0 - (-6.0))/(+5.0 + 5.0);
CONSTANT m5 IS (4.0E0 - (-6.0))*(+5.0E0 + 5.0);
CONSTANT d5 IS (4.0 - (-6.0E0))/(+5.0E0 + 5.0E0);
CONSTANT m6 IS (4.0 - (-6.0))*(+5.0E0 + 5.0E0);
CONSTANT d6 IS (4.0 - (-6.0))/(+5.0 + 5.0E0);
CONSTANT m7 IS (4.0E0 - (-6.0))*(+5.0 + 5.0);
CONSTANT d7 IS (4.0 - (-6.0E0))/(+5.0 + 5.0);

END D_ct6;

A.1.16 t1/ct7.sme

DEFINITION MODULE D_ct7 IS

-- enumeration declarations

-- ENUMERATION Branches IS

Intermetrics, Inc. 63
Silver_Spring,
Gaithersburg,
Potomac);

ENUMERATION Loan_types IS
  ( mortgage,
    auto,
    personal);

-- domain character declarations
--

DOMAIN Customer_name_domain IS
  NEW SQL_CHAR(length => 50);
DOMAIN SSN_domain IS
  NEW SQL_CHAR NOT NULL (length => 9);
DOMAIN Addr_domain IS
  NEW SQL_CHAR(length => 25);
DOMAIN City_domain IS
  NEW SQL_CHAR(length => 25);
DOMAIN State_domain IS
  NEW SQL_CHAR(length => 2);
DOMAIN Branch_name_domain IS
  NEW SQL_CHAR(length => 25);
END D_ct7;

A.1.17 t1/ct8.sme

-- !reference d_ct7
with D_ct7; use D_ct7;
DEFINITION MODULE D_ct8 IS
--
-- domain integer declarations
--

DOMAIN ZIP_code_domain IS
  NEW SQL_INT( FIRST => 0, LAST => 999999999);
DOMAIN ZIP2_code_domain IS
  NEW SQL_INT NOT NULL;
DOMAIN Account_number_domain IS
  NEW SQL_SMALLINT( FIRST => 0, LAST => 9999);
DOMAIN Account2_number_domain IS
  NEW SQL_SMALLINT NOT NULL;

-- domain real declarations
--

DOMAIN Balance_domain IS
  NEW SQL_REAL;
DOMAIN Interest_rate_domain IS
  NEW SQL_REAL( FIRST => 0.0, LAST => 1.0);
DOMAIN Loan_payment_domain IS
  NEW SQL_REAL NOT NULL;
DOMAIN Branch_assets_domain IS
  NEW SQL_REAL NOT NULL ( FIRST => 0.0, LAST => 1.0E+10);

-- domain enumeration declarations
--

DOMAIN Loan_type_domain IS
  NEW SQL_ENUMERATION_AS_CHAR
    (ENUMERATION => Loan_types, MAP => IMAGE);
Appendix A - Compiler Test Suite Source Code

DOMAIN Loan2_type_domain IS
NEW SQL_ENUMERATION AS CHAR NOT NULL
(ENUMERATION => Loan_types, MAP => IMAGE);
DOMAIN Branch_number_domain IS
NEW SQL_ENUMERATION AS INT
(MAP => POS, ENUMERATION => Branches);
DOMAIN Branch2_number_domain IS
NEW SQL_ENUMERATION AS INT NOT NULL
(MAP => POS, ENUMERATION => Branches);

-- record definitions

RECORD Customer_record IS
Cust_Name : Customer_name-domain;
SSN : SSN_domain NOT NULL;
Street : Addr_domain;
City : City_domain;
State : State_domain;
ZIP : ZIP_code_domain;
END customer_record;
END D_ct8;

A.1.18 t1/ct9.sme

--!reference d_ct7
--!reference d_ct8
WITH D_ct7,D_ct8;
USE D_ct7,D_ct8;
SCHEMA MODULE S_ct9 IS

-- Basic customer information

TABLE Customer IS
Cust_Name : Customer_name_domain,
SSN not null : SSN_domain,
Street_addr : Addr_domain,
City_addr : City_domain,
State_addr : State_domain,
ZIP_addr : ZIP_code_domain
END Customer;

-- Savings account

TABLE Savings_account IS
SBranch_number : Branch_number_domain,
SAccount_number : Account_number_domain,
SBalance : Balance_domain,
SCustomer_SSN not null : SSN_domain
END;

-- Checking account

TABLE Checking_account IS
CBranch_number : Branch_number_domain,
CAccount_number : Account_number_domain,
CBalance : Balance_domain,
CCustomer_SSN not null : SSN_domain
END Checking_account;

END S_ct9;

A.1.19 tl/ct10.sme

--!reference s_ct9
WITH D_ct7,D_ct8;
USE D_ct7,D_ct8;
SCHEMA MODULE S_ct10 IS
--
-- loan account
--
TABLE loan_account IS
  LBranch_number : Branch_number_domain,
  LAccount_number : Account_number_domain,
  LBalance : Balance_domain,
  LPayment not null : Loan_Payment_domain,
  LCustomer_SSN not null : SSN_domain
END loan_account;
--
-- Branch information
--
TABLE Branch_info IS
  Branch_name : Branch_name_domain,
  Branch_number : Branch_number_domain,
  Assets not null : Branch_assets_domain
END Branch_info;

END S_ct10;

A.1.20 tl/ct11.sme

--!reference s_ct10
WITH D_ct7,D_ct8;
USE D_ct7; USE D_ct8;
WITH SCHEMA S_ct10;
ABSTRACT MODULE A_ct11 IS
  AUTHORIZATION S_ct9
--
-- procedures
--
-- commit statement
--
PROCEDURE Commit_work IS
  COMMIT WORK;
--
-- delete statement
--
PROCEDURE Delete_customer_loan (loan_number_in : Account_number_domain) IS
  DELETE FROM
    S_ct10.LOAN_account
  WHERE
    S_ct10.LOAN_account.Laccount_number = loan_number_in;

PROCEDURE Delete_customers IS
DELETE FROM
S_ct9.Customer;
-- rollback statement
--
PROCEDURE rollback_work IS
  ROLLBACK WORK;
--
-- update statement
--
PROCEDURE Update_savings_account_balance
  (account_number_in : account_number_domain;
   transaction : balance_domain)
IS
  UPDATE
    S_ct9.Savings_account
  SET
  WHERE
    S_ct9.Savings_account.Saccount_number = account_number_in;
END A_ct11;

A.1.21     t1/ct12.sme

--!reference a_ct11
WITH D_ct7,D_ct8;
USE D_ct7;
USE D_ct8;
WITH SCHEMA S_ct9;
ABSTRACT MODULE A_ct12 IS
  AUTHORIZATION S_ct10
  PROCEDURE Savings_and_loan_transaction IS
    UPDATE
      S_ct10.loan_account
    SET
      S_ct10.loan_account.Lbalance = 0.0;
  
PROCEDURE move_checking_to_savings
  (account_num_in : account_number_domain)
IS
  INSERT INTO
    S_ct9.savings_account
  SELECT *
  FROM
    S_ct9.checking_account
  WHERE
    S_ct9.checking_account.Caccount_number >= account_num_in;

PROCEDURE New_customer IS
  INSERT INTO
    S_ct9.Customer
FROM
  Intermetrics, Inc.
New_customer_info : new c_rec
VALUES;
--
-- select statement
--
PROCEDURE Get_customer_profile (SSN_in : SSN_domain) IS
SELECT *
INTO
Customer_Profile : customer_record
FROM
S_ct9.Customer
WHERE
S_ct9.Customer.SSN = SSN_in;
END A_ct12;

A.1.22  t1/c13.sme

definition module D_ct13 is
-- Member Information
domain MemName is new SQL_CHAR Not Null (Length => 30);
domain SSN is new SQL_CHAR Not Null (Length => 9);
domain Age is new SQL_SMALLINT ( FIRST => 1, LAST => 199);
enumeration SexEnum is (F, M);
domain Sex is new SQL_ENUMERATION_AS_INT
  MAP => POS, ENUMERATION => SexEnum);
domain Phone is new SQL_CHAR (Length => 8);
domain Street is new SQL_CHAR (Length => 30);
domain City is new SQL_CHAR (Length => 15);
domain County is new SQL_CHAR Not Null (Length => 2);
domain Club_Number is new SQL_SMALLINT Not Null;
domain Sum_Domain is new SQL_SMALLINT Not Null;
domain Count_Domain is new SQL_INT;
end D_ct13;

with D_ct13; use D_ct13;
schema module RecDB is

  table Members is
    MemberName not null : MemName,
    MemberSSN not null : SSN,
    ClubNumber not null : Club_Number,
    MemberAge : Age,
    MemberSex : Sex,
    MemberPhone : Phone,
    MemberStreet : Street,
    MemberCity : City,
    MemberCnty not null : County
end Members;

  table Members2 is
    MemberName2 not null : MemName,

Intermetrics, Inc.
Appendix A - Compiler Test Suite Source Code

MemberSSN2 not null : SSN,
ClubNumber2 not null : Club_Number,
MemberAge2 : Age,
MemberSex2 : Sex,
MemberPhone2 : Phone,
MemberStreet2 : Street,
MemberCity2 : City,
MemberCnty2 not null : County
end Members2;
end RecDB;

with D..ct13; use D..ct13;
abstract module A..ct13 is
  authorization RecDB
    record MemberRec is
      RMemberName : MemName;
      RSum : Sum_Domain;
      R_Count : Count_Domain;
    end;
  
  procedure P_MemberSelect (ReqMemberSSN : SSN) is
    select MemberName, SUM(MemberAge), COUNT(*)
    into Row : MemberRec
    from RecDB.Members
    where RecDB.Members.MemberSSN = ReqMemberSSN ;

  procedure MP_MemberSelect (ReqMemberSSN : SSN) is
    select MemberName, Sum_Domain(SUM(MemberAge)),
    Count_Domain(COUNT(*))
    into Row : MemberRec
    from RecDB.Members
    where RecDB.Members.MemberSSN = ReqMemberSSN;

  procedure MPD_MemberSelect (ReqMemberSSN : SSN) is
    select MemberName, Sum_Domain(SUM(MemberAge)) named x,
    CountDomain(COUNT(*)) named foo
    from RecDB.Members
    where RecDB.Members.MemberSSN = ReqMemberSSN ;

  cursor M_MemberSelect (ReqMemberSSN : SSN) for
    select MemberName, SumDomain(SUM(MemberAge)) named sm,
    CountDomain(COUNT(*)) named ct
    from RecDB.Members
    where RecDB.Members.MemberSSN = ReqMemberSSN ;
  is
    procedure FetchIt is
      fetch into Row : new MRec;
    end M_MemberSelect;

  cursor MD_MemberSelect (ReqMemberSSN : SSN) for
    select MemberName, SumDomain(SUM(MemberAge)) named msum,
    CountDomain(COUNT(*)) named foo
    from RecDB.Members
    where RecDB.Members.MemberSSN = ReqMemberSSN ;
  is
procedure FetchIt is
  fetch;
end MD_MemberSelect;

end A_ctl13;

A.1.23 t1/ct14.sme

-- Check replacement of constants and enum literals in embedded
-- C-code

DEFINITION MODULE d_ct14 IS
--
-- enumeration declarations
--
ENUMERATION Branches IS
  (Bethesda,
   Silver_Spring,
   Gaithersburg,
   Potomac);

ENUMERATION Loan_types IS
  (mortgage,
   auto,
   personal);
--
-- domain enumeration declarations
--
DOMAIN Loan_type_domain IS
  NEW SQL_ENUMERATION_AS_int
   (MAP => POS, ENUMERATION => Loan_types);

DOMAIN branch_num_domain IS
  NEW SQL_ENUMERATION_AS_Char
   (ENUMERATION => Branches, MAP => IMAGE);

  constant C1 : loan_type_domain is mortgage;
  constant C2 : loan_type_domain is loan_type_domain (loan_type_domain
    (auto));
  constant C3 : branch_num_domain is Bethesda;
  constant C4 : branch_num_domain is branch_num_domain
    (branch_num_domain (Silver_Spring));

END d_ct14;
WITH d_ctl4;
USE d_ctl4;
SCHEMA MODULE s_ctl4 IS

    TABLE Cust IS
      Col1 : loan_type_domain,
      Col2 : loan_type_domain,
      Col3 : branch_num_domain,
      Col4 : branch_num_domain,
      Col5 : loan_type_domain,
      Col6 : branch_num_domain
    END cust;

END s_ctl4;
WITH d_ctl14;
USE d_ctl14;
ABSTRACT MODULE a_ctl14 IS
  AUTHORIZATION s_ctl14

  PROCEDURE New_customer IS
    INSERT INTO
      s_ctl14.cust
    VALUES (C1, C2, C3, C4, personal, Gaithersburg);
  
  PROCEDURE Sel_Cust IS
    SELECT *
    FROM s_ctl14.cust
    where Col1 = C1 and
        Col2 = C2 and
        Col3 = C3 and
        Col4 = C4 and
        Col5 = loan_type_domain(personal) and
        Col6 = Gaithersburg;

  PROCEDURE Upd_Cust IS
    UPDATE s_ctl14.cust
    SET
        Col1 = C1,
        Col2 = C2,
        Col3 = C3,
        Col4 = C4,
        Col5 = personal,
        Col6 = Gaithersburg;

END a_ctl14;

A.1.24 t1/ctl15.sme

-- Various insert values tests

DEFINITION MODULE d_ctl15 IS
  --
  -- enumeration declarations
  --
  ENUMERATION Branches IS
    (Bethesda,
     Silver_Spring,
     Gaithersburg,
     Potomac);

  ENUMERATION Loan_types IS
    (mortgage,
     auto,
     personal);

  --
  -- domain character declarations
  --
  DOMAIN Customer_name_domain IS
    NEW SQL_CHAR(length => 15);
  DOMAIN Addr_domain IS
    NEW SQL_CHAR(length => 15);
Appendix A - Compiler Test Suite Source Code

DOMAIN City_domain IS
   NEW SQL_CHAR(length => 15);
DOMAIN State_domain IS
   NEW SQL_CHAR(length => 2);

-- domain integer declarations

DOMAIN SSN_domain IS
   NEW SQL_INT NOT NULL ( FIRST => 0, LAST => 999999999);
DOMAIN acct_num_domain IS
   NEW SQL_SMALLINT NOT NULL ( FIRST => 0, LAST => 9999);

-- domain real declarations

"DOMAIN Balance_domain IS
   NEW SQL_REAL;
DOMAIN Interest_rate_domain IS
   NEW SQL_REAL( FIRST => 0.0, LAST => 1.0);
DOMAIN Loan_payment_domain IS
   NEW SQL_REAL;
DOMAIN Branch_assets_domain IS
   NEW SQL_REAL;

-- domain enumeration declarations

DOMAIN Loan_type_domain IS
   NEW SQL_ENUMERATION_AS_INT
      (MAP => POS, ENUMERATION => Loan_types);
DOMAIN branch_num_domain IS
   NEW SQL_ENUMERATION_AS_INT
      (MAP => POS, ENUMERATION => Branches);
RECORD Customer_record IS
   Cust_Name : Customer_name_domain;
   SSN : SSN_domain;
   Street : Addr_domain;
   City : City_domain;
   State : State_domain;
END customer_record;

END d_ct15;

WITH d_ct15;
USE d_ct15;
SCHEMA MODULE s_ct15 IS

-- Basic customer information

TABLE Cust IS
   Cust_Name : Customer_name_domain,
   SSN not null : SSN_domain,
   Street_addr : Addr_domain,
   City_addr : City_domain,
   State_addr : State_domain
END cust;

-- Savings account

Intermetrics, Inc.
--

TABLE Save IS
    branch_num : branch_num_domain,
    acct_num not null : acct_num_domain,
    Balance : Balance_domain,
    cust_ssn not null : SSN_domain
END Save;

END s_ctl5;

WITH d_ctl5;
USE d_ctl5;
ABSTRACT MODULE a_ctl5 IS
    AUTHORIZATION s_ctl5

    RECORD customer_record_minus IS
        Cust_Name : Customer_name_domain;
        SSN : SSN_domain;
        City : City_domain;
        State : State_domain;
    END customer_record_minus;

    PROCEDURE New_customer IS
        INSERT INTO
            s_ctl5.cust (Cust_Name, SSN, Street_addr, City_addr,
            State_addr)
        FROM
            New_customer_info : new c_record
            VALUES;

    PROCEDURE New_customer1 IS
        INSERT INTO
            s_ctl5.cust (Cust_Name, SSN, Street_addr, City_addr,
            State_addr)
        FROM
            New_customer_info
            VALUES (Cust_Name, SSN, Street_addr, City_addr, State_addr);

    PROCEDURE New_customer2 IS
        INSERT INTO
            s_ctl5.cust (Cust_Name, SSN, Street_addr; City_addr,
            State_addr)
        FROM
            New_customer_info2
            VALUES (Cust_Name, SSN, NULL, City_addr, State_addr);

    PROCEDURE New_customer3 IS
        INSERT INTO
            s_ctl5.cust (Cust_Name, SSN, Street_addr, City_addr,
            State_addr)
        FROM
            New_customer_info3
            VALUES (Cust_Name, SSN, '11261 Col Pike', City_addr,
            State_addr);

    PROCEDURE New_customer4 IS
        INSERT INTO
Appendix A - Compiler Test Suite Source Code

```
s_ctl5.cust (Cust_Name, SSN, Street_addr, City_addr, State_addr)
FROM : new row_type1
VALUES;

PROCEDURE New_customer5 IS
    INSERT INTO
    s_ctl5.cust (Cust_Name, SSN, Street_addr, City_addr, State_addr)
    FROM : new row_type2
    VALUES (Cust_Name, SSN, Street_addr, City_addr, State_addr);

PROCEDURE New_customer6 IS
    INSERT INTO
    s_ctl5.cust (Cust_Name, SSN, Street_addr, City_addr, State_addr)
    FROM : new row_type3
    VALUES (Cust_Name, SSN, NULL, City_addr, State_addr);

PROCEDURE New_customer7 IS
    INSERT INTO
    s_ctl5.cust (Cust_Name, SSN, Street_addr, City_addr, State_addr)
    FROM : new row_type4
    VALUES (Cust_Name, SSN, '11261 Col Pike', City_addr, State_addr);

PROCEDURE New_customer8 IS
    INSERT INTO
    s_ctl5.cust (Cust_Name, SSN, Street_addr, City_addr, State_addr)
    FROM : new row_type5
    VALUES (Cust_Name, SSN, NULL, City_addr, State_addr);

END a_ctl5;

A.1.25
t1/ctl16.sme

definition module d_ctl16 is
    -- Member Information
    domain MemName is new SQL_CHAR Not Null (Length => 30);
    domain SSN is new SQL_CHAR Not Null (Length => 9);
    domain Age is new SQL_SMALLINT (FIRST => 1, LAST => 199);

    enumeration SexEnum is (F, M);
    domain Sex is new SQLENUMERATION_AS_INT (
        MAP => POS, ENUMERATION => SexEnum);

    domain Phone is new SQL_CHAR (Length => 8);
    domain Street is new SQL_CHAR (Length => 30);
    domain City is new SQL_CHAR (Length => 15);

    domain County is new SQL_CHAR Not Null (Length => 2);
    domain Club_Number is new SQL_SMALLINT Not Null;

    constant C_Name : MemName is '123456789012345678901234567890';
    constant C_SSN : SSN is '1234567890';
```

Intermetrics, Inc.
constant C_Club_Number : Club_Number is 10;
constant C_Age : Age is 39;
constant C_Sex : Sex is F;
constant C_Phone : Phone is '123456789';
constant C_Street : Street is '123456789012345678901234567890';
constant C_City : City is '12345678901234567890';
constant C_County : County is 'MO';
end d_ctl6;

with d_ctl6; use d_ctl6;
schema module RecDB is
  table Members is
    MemberName not null : MemName,
    MemberSSN not null : SSN,
    ClubNumber not null : Club_Number,
    MemberAge : Age,
    MemberSex : Sex,
    MemberPhone : Phone,
    MemberStreet : Street,
    MemberCity : City,
    MemberCnty not null : County
  end Members;
end RecDB;

with d_ctl6; use d_ctl6;
abstract module a_ctl6 is
  authorization RecDB
    record MemberRec named Named_MemberRec is
      -- record MemberRec is
      R_MemberName : MemName;
      R_MemberSSN : SSN;
      R_ClubNumber : Club_Number;
      R_MemberAge : Age;
      R_MemberSex : Sex;
      R_MemberPhone : Phone;
      R_MemberStreet : Street;
      R_MemberCity : City not null;
      R_MemberCnty : County;
    end;

cursor MemberSelect2 (Req_MemberSSN named Req_MemberSSN : SSN) for
  select
    MemberName named NS_MemberName,
    MemberSSN,
    ClubNumber,
    MemberAge,
    MemberSex,
    MemberPhone Not Null,
    MemberStreet named NS_MemberStreet Not Null,
    MemberCity,
    MemberCnty
  from RecDB.Members
    where RecDB.Members.MemberSSN = Req_MemberSSN
  UNION

Intermetrics, Inc.
select
    MemberName named NS_MemberName,
    MemberSSN,
    ClubNumber,
    MemberAge,
    MemberSex,
    MemberPhone Not Null,
    MemberStreet named NS_MemberStreet Not Null,
    MemberCity,
    MemberCnty
from RecDB.Members
where RecDB.Members.MemberSSN = Req_MemberSSN
UNION
select
    MemberName named NS_MemberName,
    MemberSSN,
    ClubNumber,
    MemberAge,
    MemberSex,
    MemberPhone Not Null,
    MemberStreet named NS_MemberStreet Not Null,
    MemberCity,
    MemberCnty
from RecDB.Members
where RecDB.Members.MemberSSN = Req_MemberSSN
UNION
select
    MemberName named NS_MemberName,
    MemberSSN,
    ClubNumber,
    MemberAge,
    MemberSex,
    MemberPhone Not Null,
    MemberStreet named NS_MemberStreet Not Null,
    MemberCity,
    MemberCnty
from RecDB.Members
where RecDB.Members.MemberSSN = Req_MemberSSN;

is

procedure FetchIt is
    -- fetch into Row_Name : MemberRec;
    -- fetch into : MemberRec;
    -- fetch into Row_Name;
    fetch ;
    -- fetch into Row_Name : new New_Row_Type;
    -- fetch into : new New_Row_Type;
end MemberSelect2;
end a_ctl6;

A.1.26 t3/t1.sme

definition module t_4 is
    -- Member Information
    domain MemberName is new SQL_CHAR Not Null (Length => 30);
    domain SSN is new SQL_CHAR Not Null (Length => 9);
    domain Age is new SQL_SMALLINT (FIRST => 1, LAST => 199);
enumeration SexEnum is (F, M);
domain Sex is new SQL_ENUMERATION AS INT (  
    Enumeration => SexEnum,  
    Map => Pos);

domain Phone is new SQL_CHAR (Length => 8);
domain Street is new SQL_CHAR (Length => 30);
domain City is new SQL_CHAR (Length => 15);

domain County is new SQL_CHAR Not Null (Length => 2);
domain Club_Number is new SQL_SMALLINT Not Null;

exception Record_Not_Found;

enumeration FailType is (Not_Logged_In, SQL_Ok, SQL_Fail);

status fetch_map named is_found uses Failtype is  
  -999 .. -300 => SQL_Fail,  
  -299, -298 => Not_Logged_In,  
    0 => SQL_Ok,  
   100 => raise record_not_found);

status bool_map uses boolean is  
  (100=>true, 0=>false);

end t_4;

with t_4; use t_4;
schema module s_4 is

  table Members is  
    MemberName not null : MemberName,  
    MemberSSN not null : SSN,  
    ClubNumber not null : Club_Number,  
    MemberAge : Age,  
    MemberSex : Sex,  
    MemberPhone : Phone,  
    MemberStreet : Street,  
    MemberCity : City,  
    MemberCnty not null : County  
  end Members;
end s_4;

with t_4; use t_4;
abstract module a_4 is
  authorization s_4

    record memberec2 named insertrec is  
      MemberName : MemberName,  
      MemberSex : Sex not null;  
    end memberec2;

    record memberec3 is  
      junk dblength named nameind : MemberName;  
      MemberSex : Sex;  
    end memberec3;

    record memberec is
Appendix A - Compiler Test Suite Source Code

```
MemberName : MemberName;
MemberSex : Sex;
end memberec;

procedure DeleteMember1 (Input_name : membername) is
  delete from members
  where Membername = Input_Name
  status bool_map;

procedure DeleteMember2 (Input_name : membername) is
  delete from members
  where Membername = Input_Name
  status bool_map named delete_status;

procedure DeleteMember3 (Input_name : membername) is
  delete from members
  where Membername = Input_Name
  status fetch_map;

procedure DeleteMember4 (Input_name : membername) is
  delete from members
  where Membername = Input_Name
  status fetch_map named delete_status;

procedure DeleteMember5 (Input_name named Delete_Me : membername) is
  delete from members
  where Membername = Input_Name
  status bool_map;

procedure SelectMember1 is
  select membersex not null, membername from s_4.Members;

procedure MemberInsert0 is
  insert into s_4.Members Values;

procedure MemberInsert1 is
  insert into s_4.Members (membername, membersex) Values;

procedure MemberInsert2 is
  insert into s_4.Members (Membername named myname, membersex
  named
  mysex not null) Values;

procedure MemberInsert3 is
  insert into s_4.Members (Membername named myname, membersex
  named
  mysex) Values (membername,membersex);

procedure MemberInsert4 is
  insert into s_4.Members (Membername named junk, membersex)
  from :memberec3 Values (membername,membersex);

procedure MemberInsert5 is
  insert into s_4.Members (Membername , membersex )
  from the_row : memberec Values (membername,membersex);

procedure MemberInsert6 is
```

Intermetrics, Inc.
insert into s_4.Members (Membername named myname, membersex named mysex) from the_row Values (membername,membersex);

procedure MemberInsert7 is
    insert into s_4.Members (Membername named myname, membersex named mysex) from :new rec7 values (membername,membersex);
end MemberInsert7;

procedure MemberInsert8 is
    insert into s_4.Members (s_4.members.Membername, membersex named mysex) from the_row :new rec8 values (membername,membersex);
end MemberInsert8;

procedure MemberInsert9 is
    insert into s_4.Members (Membername, membersex) from zmemberec2 values (membername,membersex);
end MemberInsert9;

procedure MemberInsert10 is
    insert into s_4.Members (s_4.members.Membername, membersex) from the_row :memberec2 values (membername,membersex);
end MemberInsert10;

procedure MemberInsert11 is
    insert into s_4.Members (Membername named myname, membersex named mysex) from :new rec11 values (membername,membersex);
end MemberInsert11;

procedure MemberInsert12 is
    insert into s_4.Members (Membername named myname, clubnumber named mycounty) from the_row :new rec12 values (membername,clubnumber);
end MemberInsert12;

cursor MemberSelect (Req_MemberSSN : SSN) is
    for
        select s_4.members.membername, membersex named mysex, clubnumber * 6 named club, memberssn named myssn
        from s_4.Members
        where memberssn = Req_MemberSSN;
end cursor MemberSelect;

cursor MemberSelect2 (Req_MemberSSN named myssn : SSN) is
    for
        select s_4.members.membername named junk, dblength named nameind, membersex
        from s_4.Members
        where memberssn = Req_MemberSSN;
end cursor MemberSelect2;

procedure Fetch1t is
    fetch into :memberec3 status standard_map;

procedure update1t is
    update members
    set s_4.members.membersex = null
    where current of memberselect2;
end update1t;

end a_4;
A.1.27  

t3/t2.sme

definition module t_10 is
    -- Member Information
    domain MemberName is new SQL_CHAR Not Null (Length => 30);
    domain SSN is new SQL_CHAR Not Null (Length => 9);
    domain Age is new SQL_SMALLINT (FIRST => 1, LAST => 199);

    enumeration SexEnum is (F, M);
    domain Sex is new SQL_ENUMERATION_AS_INT (Enumeration => SexEnum, Map => Pos);

    domain Phone is new SQL_CHAR (Length => 8);
    domain Street is new SQL_CHAR (Length => 30);
    domain City is new SQL_CHAR (Length => 15);
    domain County is new SQL_CHAR Not Null (Length => 2);
    domain Club_Number is new SQL_SMALLINT Not Null;

    exception Record_Not_Found;

    enumeration FailType is (Not_Logged_In, SQLOk, SQLFail);

    status fetch_map named is_found uses Failtype is
        (-999 .. -300 => SQLFail, -299, -298 => Not_Logged_In, 0 => SQLOk, 100 => raise recordnotfound);

    status bool_map uses boolean is
        (100=>true, 0=>false);

end t_10;

with t_10; use t_10;

abstract module a_10 is

end a_10;

with t_10; use t_10;

abstract module a_10 is
    authorization s_10

    record memberec2 named insertrec is
        MemberName : MemberName;

end a_10;

with t_10; use t_10;

abstract module a_10 is
    authorization s_10

    record memberec2 named insertrec is
        MemberName : MemberName;

end a_10;

with t_10; use t_10;

abstract module a_10 is
    authorization s_10

    record memberec2 named insertrec is
        MemberName : MemberName;

end a_10;

with t_10; use t_10;

abstract module a_10 is
    authorization s_10

    record memberec2 named insertrec is
        MemberName : MemberName;

end a_10;

with t_10; use t_10;

abstract module a_10 is
    authorization s_10

    record memberec2 named insertrec is
        MemberName : MemberName;

end a_10;
MemberSex : Sex not null;
end memberec2;

record memberec3 is
   junk dblength named nameind : MemberName;
   MemberSex : Sex;
end memberec3;

record memberec is
   MemberName : MemberName;
   MemberSex : Sex;
end memberec;

cursor MemberSelect2 (Req_MemberSSN named myssn SSN) for
   select s_10.members.membername named junk dblength named
   nameind,
   membersex
   from s_10.Members
   where memberssn = Req_MemberSSN
   union
   select members.membername named junk dblength named nameind,
   membersex
   from s_10.Members;
   is
   procedure FetchIt is
      fetch into :memberec3
      status standard_map;
   procedure updateit is
      update members
      set s_10.members.membersex = null
      where current of memberselect2;
   end MemberSelect2;
end a_10;

A.1.28 t3/t3sme

definition module t_11 is
   -- Member Information
   domain MemberName is new SQL_CHAR Not Null (Length => 30);
   domain SSN is new SQL_CHAR Not Null (Length => 9);
   domain Age is new SQLSMALLINT (FIRST => 1, LAST => 199);

   enumeration SexEnum is (F, M);
   domain Sex is new SQL_ENUMERATION_AS_INT (ENUMERATION => SexEnum, Map => POS);

   domain Phone is new SQL_CHAR (Length => 8);
   domain Street is new SQL_CHAR (Length => 30);
   domain City is new SQL_CHAR (Length => 15);

   domain County is new SQL_CHAR Not Null (Length => 2);
   domain Club_Number is new SQL_SMALLINT Not Null;

   exception Record_Not_Found;

Intermetrics, Inc.
Appendix A - Compiler Test Suite Source Code

enumeration FailType is (Not_Logged_In, SQL_OK, SQL_Fail);

status fetch_map named is_found uses Failtype is

-999 .. -300 => SQL_Fail,
-299, -298 => Not_Logged_In,
0 => SQL_OK,
100 => raise record_not_found);

end t_11;

with t_11; use t_11;
schema module s_11 is
table Members is
  MemberName not null : MemberName,
  MemberSSN not null : SSN,
  ClubNumber not null : Club_Number,
  MemberAge : Age,
  MemberSex : Sex,
  MemberPhone : Phone,
  MemberStreet : Street,
  MemberCity : City,
  MemberCnty not null : County
end Members;
end s_11;

with t_11; use t_11;
abstract module a_11 is
authorization s_11
record MemberRec is
  MemberName : MemberName;
  MemberSSN : SSN;
  ClubNumber : Club_Number;
  MemberAge : Age;
  MemberSex : Sex;
  MemberPhone : Phone;
  MemberStreet : Street;
  MemberCity : City;
  MemberCnty : County;
end;

cursor MemberSelect (Req_MemberSSN : SSN) for
  select MemberSSN, MemberName
  from s_11.Members as tl
  where tl.MemberSSN = '012345678';
end a_11;

A.1.29 t3/t4.sme

definition module t_8 is
  -- Member Information
domain MemberName is new SQL_CHAR Not Null (Length => 30);
domain SSN is new SQL_CHAR Not Null (Length => 9);

Intermetrics, Inc.
domain Age is new SQL_SMALLINT (FIRST => 1, LAST => 199);

enumeration SexEnum is (F, M);
domain Sex is new SQL_ENUMERATION_AS_INT (  
    Enumeration => SexEnum,  
    Map => Pos);

domain Phone is new SQL_CHAR (Length => 8);  
domain Street is new SQL_CHAR (Length => 30);  
domain City is new SQL_CHAR (Length => 15);

domain County is new SQL_CHAR Not Null (Length => 2);

domain Club_Number is new SQL_SMALLINT Not Null;

exception Record_Not_Found;

enumeration FailType is (NotLoggedIn, SQLOk, SQLFail);

status f_map named is_found is  
( 100 => raise record_not_found);

status fetch_map named is_found uses Failtype is  
(-999 .. -300 => SQLFail,  
-299 .. -298 => NotLoggedIn,  
0 => SQLOk,  
100 => raise record_not_found);

status bool_map uses boolean is  
(100=>true, 0=>false);

end t_8;

with t_8; use t_8;
schema module s_8 is  
    -- table mytable is  
    -- unique(ssn)
    -- end mytable;

table Members is  
    MemberName char(30) default 'jennifer' not null primary key:  
    MemberName,  
    MemberSSN character default user not null : SSN,  
    ClubNumber int not null : Club_Number,  
    MemberAge references mytable : Age,  
    MemberSex check(MemberSex <> f) : Sex,  
    MemberPhone : Phone,  
    MemberStreet : Street,  
    MemberCity : City,  
    MemberCnty not null unique : County,  
    foreign key (memberAge,membername) references another_table  
    (age,mname),
    primary key (memberssn),  
    unique (membername,memberphone),  
    check (membercnty <> 'PG')
end Members;
grant all privileges on jc.anothertable to jc, gdt, am with grant option;

grant select, insert, delete on informix.customer to jc;
end s_8;

with t_8; use t_8;
abstract module a_8 is
  authorization s_8

  record memberec2 named insertrec is
    MemberName : MemberName;
    MemberSex  : Sex not null;
  end memberec2;

  record memberec3 is
    junk dblength named nameind : MemberName;
    MemberSex : Sex;
  end memberec3;

  record memberec is
    MemberName : MemberName;
    MemberSex  : Sex;
  end memberec;

  procedure DeleteMember1 ( Input_name : membername) is
    delete from members
      where Membername = Input_Name
    status bool_map;
  end procedure;

  procedure DeleteMember2 ( Input_name : membername) is
    delete from members
      where Membername = Input_Name
    status bool_map named delete_status;
  end procedure;

  procedure DeleteMember3 ( Input_name : membername) is
    delete from members
      where Membername = Input_Name
    status f_map;
  end procedure;

  procedure DeleteMember4 ( Input_name : membername) is
    delete from members
      where Membername = Input_Name
    status fetch_map named delete_status;
  end procedure;

  procedure DeleteMember5 ( Input_name named Delete_Me : membername)
    is
      delete from members
        where Membername = Input_Name
      status bool_map;
  end procedure;

  procedure SelectMember1 is
select membersex not null, membername from s_8.Members;

procedure MemberInsert0 is
    insert into s_8.Members Values;
end MemberInsert0;

procedure MemberInsert1 is
    insert into s_8.Members (membername, membersex) Values;
end MemberInsert1;

procedure MemberInsert2 is
    insert into s_8.Members (Membername named myname, membersex
    named
    mysex not null) Values;
end MemberInsert2;

procedure MemberInsert3 is
    insert into s_8.Members (Membername named myname, membersex
    named
    mysex) Values (membername, membersex);
end MemberInsert3;

procedure MemberInsert4 is
    insert into s_8.Members (Membername named junk, membersex)
    from :memberrec3 Values (membername, membersex);
end MemberInsert4;

procedure MemberInsert5 is
    insert into s_8.Members (Membername, membersex)
    from the_row: memberrec Values (membername, membersex);
end MemberInsert5;

procedure MemberInsert6 is
    insert into s_8.Members (Membername named myname, membersex
    named
    mysex) from the_row Values (membername, membersex);
end MemberInsert6;

procedure MemberInsert7 is
    insert into s_8.Members (Membername named myname, membersex
    named
    mysex) from : new rec7 Values (membername, membersex);
end MemberInsert7;

procedure MemberInsert8 is
    insert into s_8.Members (s_8.members.Membername, membersex
    named
    mysex) from the_row: new rec8 Values (membername, membersex);
end MemberInsert8;

procedure MemberInsert9 is
    insert into s_8.Members (Membername, membersex)
    from : memberrec2 Values (membername, membersex);
end MemberInsert9;

procedure MemberInsert10 is
    insert into s_8.Members (s_8.members.Membername, membersex)
    from the_row: memberrec2 Values (membername, membersex);
end MemberInsert10;

procedure MemberInsert11 is
    insert into s_8.Members (Membername named myname, membersex
    named
    mysex) from : new rec11 Values (membername, membersex);
end MemberInsert11;

procedure MemberInsert12 is
    insert into s_8.Members (Membername named myname, clubnumber
    named
    myname) from : new rec12 Values (membername, membersex);
end MemberInsert12;
mycounty) from the_row : new rec12 Values (membername,clubnumber);

cursor MemberSelect (Req_MemberSSN : SSN) for
select a_8.members.membername , membersex named mysex,
clubnumber * 6 named club,
memberssn named myssn
from a_8.Members
where memberssn = a_8.memberselect.Req_MemberSSN;

cursor MemberSelect2 (Req_MemberSSN named myssn : SSN) for
select a_8.members.membername named junk dblength named nameind,
membersex
from a_8.Members
where memberssn = Req_MemberSSN;

is
procedure FetchIt is
fetch into :memberec3
status f_map ;

procedure updateit is
update members
set a_8.members.membersex = null
where current of memberselect2;
end MemberSelect2;
end a_8;

A.1.30 t3/t5.sme

definition module t_9 is
-- Member Information
domain MemberName is new SQL_CHAR Not Null (Length => 30);
domain SSN is new SQL_CHAR Not Null (Length => 9);
domain Age is new SQL_SMALLINT (FIRST => 1, LAST => 199);

enumeration SexEnum is (F, M);
domain Sex is new SQL_ENUMERATION_AS_INT
    (Enumeration => SexEnum,
    Map => Pos);

domain Phone is new SQL_CHAR (Length => 8);
domain Street is new SQL_CHAR (Length => 30);
domain City is new SQL_CHAR (Length => 15);

domain County is new SQL_CHAR Not Null (Length => 2);
domain Club_Number is new SQL_SMALLINT Not Null;

exception Record_Not_Found;

enumeration FailType is (Not_Loged_In, SQL_Ok, SQL_Fail);

status fetch_map named is_found uses Failtype is
( -999 .. -300 => SQL_Fail,
  -299, -298 => Not_Loged_In,
  0 => SQL_Ok,
  100 => raise record_not_found);
status bool_map uses boolean is
(100=>true, 0=>false);
end t_9;

with t_9; use t_9;
schema module s_9 is
  table Members is
    MemberName not null : MemberName,
    MemberSSN not null : SSN,
    ClubNumber not null : Club_Number,
    MemberAge : Age,
    MemberSex : Sex,
    MemberPhone : Phone,
    MemberStreet : Street,
    MemberCity : City,
    MemberCnty not null : County
  end Members;
end s_9;

with t_9; use t_9;
abstract module a_9 is
  authorization s_9
    record memberec2 named insertrec is
      MemberName : MemberName;
      MemberSex : Sex not null;
    end memberec2;

    record memberec3 is
      junk dblength named nameind : MemberName;
      MemberSex : Sex;
    end memberec3;

    record memberec is
      MemberName : MemberName;
      MemberSex : Sex;
    end memberec;

    procedure DeleteMember1 (Input_name : membername) is
    delete from members
      where Membername = Input_Name
    status bool_map;

    procedure DeleteMember2 (Input_name : membername) is
    delete from members
      where Membername = Input_Name
    status bool_map named delete_status;

    procedure DeleteMember3 (Input_name : membername) is
    delete from members
      where Membername = Input_Name
    status fetch_map;

    procedure DeleteMember4 (Input_name : membername) is
    delete from members
      where Membername = Input_Name

Intermetrics, Inc.
status fetch_map named delete_status;

procedure DeleteMember (Input_name named Delete_Me : membername)

is

delete from members

where Membername = Input_Name

status bool_map;

procedure SelectMember is

select membersex not null, membername from s_9.Members;

procedure MemberInsert0 is

insert into s_9.Members Values;

procedure MemberInsert1 is

insert into s_9.Members (membername, membersex) Values;

procedure MemberInsert2 is

insert into s_9.Members (Membername named myname, membersex

named

mysex not null) Values;

procedure MemberInsert3 is

insert into s_9.Members (Membername named myname, membersex

named

mysex) Values (membername,membersex);

procedure MemberInsert4 is

insert into s_9.Members (Membername named junk, membersex)

from :memberec3 Values (membername,membersex);

procedure MemberInsert5 is

insert into s_9.Members (Membername , membersex )

from the_row : memberec Values (membername,membersex);

procedure MemberInsert6 is

insert into s_9.Members (Membername named myname, membersex

named

mysex) from the_row Values (membername,membersex);

procedure MemberInsert7 is

insert into s_9.Members (Membername named myname, membersex

named

mysex) from : new rec7 Values (membername,membersex);

procedure MemberInsert8 is

insert into s_9.Members (s_9.members.Membername , membersex

named

mysex) from the_row : new rec8 Values (membername,membersex);

procedure MemberInsert9 is

insert into s_9.Members (Membername, membersex)

from :memberec2 Values (membername,membersex);

procedure MemberInsert10 is

insert into s_9.Members (s_9.members.Membername , membersex )

from the_row : memberec2 Values (membername,membersex);

Intermetrics, Inc. 89
procedure MemberInsert11 is
  insert into s_9.Members (Membername named myname, membersex named mysex) from : new rec11 Values (membername,membersex);

procedure MemberInsert12 is
  insert into s_9.Members (Membername named myname, clubnumber named mycounty) from the_row : new rec12 Values (membername,clubnumber);

cursor MemberSelect (Req_MemberSSN SSN) for
  select s_9.members.membername , membersex named mysex, clubnumber * 6 named club,
  memberssn named myssn
  from s_9.Members
  where memberssn = a_9.memberselect.Req_MemberSSN;

cursor MemberSelect2 (Req.MemberSSN named myssn : SSN) for
  select s_9.members.membername named junk dblength named nameind, membersex
  from s_9.Members
  where memberssn = Req.MemberSSN
  union
  select s_9.members.membername named junk dblength named nameind, membersex
  from s_9.Members;

is
  procedure FetchIt is
    fetch into :memberec3
    status standard_map;
  procedure updateit is
    update members
    set s_9.members.membersex = null
    where current of memberselect2;
  end MemberSelect2;

end a_9;

A.1.31 13/16.sme

definition module t_12 is
  -- Member Information
domain MemberName is new SQL_CHAR Not Null (Length => 30);
domain SSN is new SQL_CHAR Not Null (Length => 9);
domain Age is new SQL_SMALLINT ( FIRST => 1, LAST => 199);

enumeration SexEnum is (F, M);
domain Sex is new SQL_ENUMERATION_AS_INT (Enumeration => SexEnum, Map => POS);

domain Phone is new SQL_CHAR (Length => 8);
domain Street is new SQL_CHAR (Length => 30);
domain City is new SQL_CHAR (Length => 15);

domain County is new SQL_CHAR Not Null (Length => 2);
domain Club_Number is new SQL_SMALLINT Not Null;
exception Record_Not_Found;

enumeration FailType is (NotLogged_In, SQL_OK, SQL_Fail);

status fetch_map named is_found uses Failtype is
  (-999 .. -300 => SQL_Fail,
   -299, -298 => NotLogged_In,
   0 => SQL_OK,
   100 => raise record_not_found);
end t_12;

with t_12; use t_12;
schema module s_12 is
  table Members is
    MemberName not null : MemberName,
    MemberSSN not null : SSN,
    ClubNumber not null : ClubNumber,
    MemberAge : Age,
    MemberSex : Sex,
    MemberPhone : Phone,
    MemberStreet : Street,
    MemberCity : City,
    MemberCnty not null : County
  end Members;

  table Members2 is
    MemberName not null : MemberName,
    MemberSSN not null : SSN,
    ClubNumber not null : ClubNumber
  end Members2;
end s_12;

with t_12; use t_12;
abstract module a_12 is
  authorization s_12

  record MemberRec is
    MemberName : MemberName;
    MemberSSN : SSN;
    ClubNumber : ClubNumber;
    MemberAge : Age;
    MemberSex : Sex;
    MemberPhone : Phone;
    MemberStreet : Street;
    MemberCity : City;
    MemberCnty : County;
  end;

cursor MemberSelect (Req_MemberSSN : SSN) for
  select MemberSSN, s_12.Members.Membername
  from s_12.Members
  where MemberSSN = (select MemberSSN
                      from Members2
                      ...);
SAMeDL Development Environment - Test Plan

where s_12.members2.membername = 'John');

end a_12;

A.1.32 t3/t7.sme

definition module t_13 is
    -- Member Information
    domain MemberName is new SQL_CHAR Not Null (Length => 30);
    domain SSN is new SQL_CHAR Not Null (Length => 9);
    domain Age is new SQL_SMALLINT (FIRST => 1, LAST => 199);

    enumeration SexEnum is (F, M);
    domain Sex is new SQL_ENUMERATION_AS_INT
        (Enumeration => SexEnum, Map => POS);

    domain Phone is new SQL_CHAR (Length => 8);
    domain Street is new SQL_CHAR (Length => 30);
    domain City is new SQLCHAR (Length => 15);
    domain County is new SQL_CHAR Not Null (Length => 2);
    domain ClubNumber is new SQL_SMALLINT Not Null;

    exception Record_Not_Found;

    enumeration FailType is (Not_Logged_In, SQL_Ok, SQL_Fail);

    status fetch_map named is_found uses Failtype is
        ( -999 .. -300 => SQL_Fail;
        -299, -298 => Not_Logged_In,
        0 => SQL_Ok,
        100 => raise record_not_found);

end t_13;

with t_13; use t_13;

schema module s_13 is

    table Members is
        MemberName not null : MemberName,
        MemberSSN not null : SSN,
        ClubNumber not null : ClubNumber,
        MemberAge : Age,
        MemberSex : Sex,
        MemberPhone : Phone,
        MemberStreet : Street,
        MemberCity : City,
        MemberCnty not null : County
    end Members;

    table Members2 is
        MemberName not null : MemberName,
        MemberSSN not null : SSN,
        ClubNumber not null : ClubNumber
    end Members2;

end s_13;
with t_13; use t_13;
abstract module a_13 is
  authorization s_13

  record MemberRec is
    MemberName : MemberName;
    MemberSSN : SSN;
    ClubNumber : ClubNumber;
    MemberAge : Age;
    MemberSex : Sex;
    MemberPhone : Phone;
    MemberStreet : Street;
    MemberCity : City;
    MemberCnty : County;
  end;

cursor MemberSelect (Req_MemberSSN : SSN) for
  select MemberSSN, Members.Membername
  from s_13.Members
    where MemberSSN = (select MemberSSN
        from Members2
            where s_13.members2.membername = 'John');
end a_13;

A.1.33 t3/t8.sme

definition module t_14 is
  -- Member Information
  domain MemberName is new SQL_CHAR Not Null (Length => 30);
  domain SSN is new SQLCHAR Not Null (Length => 9);
  domain Age is new SQLSMALLINT (FIRST => 1, LAST => 199);

  enumeration SexEnum is (F, M);
  domain Sex is new SQLENUMERATION_AS_INT 
      (Enumeration => SexEnum, Map => POS);

  domain Phone is new SQL_CHAR (Length => 8);
  domain Street is new SQL_CHAR (Length => 30);
  domain City is new SQL_CHAR (Length => 15);

  domain County is new SQL_CHAR Not Null (Length => 2);
  domain Club_Number is new SQL_SMALLINT Not Null;

  exception Record_Not_Found;

  enumeration FailType is (Not_Loocked_In, SQL_OK, SQL_Fail);

  status fetch_map named is_found uses Failtype is
    ( -999..-300 => SQL_Fail,
        -299, -298 => Not_Loocked_In,
        0 => SQL_OK,
        100 => raise record_not_found);
with t_14; use t_14;
schema module s_14 is
  table Members is
    MemberName not null : MemberName,
    MemberSSN not null : SSN,
    ClubNumber not null : ClubNumber,
    MemberAge : Age,
    MemberSex : Sex,
    MemberPhone : Phone,
    MemberStreet : Street,
    MemberCity : City,
    MemberCnty not null : County
  end Members;

  table Members2 is
    MemberName not null : MemberName,
    MemberSSN not null : SSN,
    ClubNumber not null : Club_Number
  end Members2;
end s_14;

with t_14; use t_14;
abstract module a_14 is
  authorization s_14
  record MemberRec is
    MemberName : MemberName;
    MemberSSN : SSN;
    ClubNumber : Club_Number;
    MemberAge : Age;
    MemberSex : Sex;
    MemberPhone : Phone;
    MemberStreet : Street;
    MemberCity : City;
    MemberCnty : County;
  end;

cursor MemberSelect (ReqMemberSSN SSN) for
  select MemberSSN, Members.Membername
  from s_14.Members
  where MemberSex = (select MemberSex
    from Members
    where s_14.members.membersex =
    t_14.f );
end a_14;

A.1.34  t3/t9.sme

definition module t_15 is
  -- Member Information
  domain MemberName is new SQL_CHAR Not Null (Length => 30);
  domain SSN is new SQL_CHAR Not Null (Length => 9);
  domain Age is new SQL_SMALLINT ( FIRST => 1, LAST => 199);
enumeration SexEnum is (F, M);
domain Sex is new SQL_ENUMERATION_AS_INT (Enumeration => SexEnum, Map => POS);

domain Phone is new SQL_CHAR (Length => 8);
domain Street is new SQL_CHAR (Length => 30);
domain City is new SQL_CHAR (Length => 15);

domain County is new SQL_CHAR Not Null (Length => 2);
domain Club_Number is new SQL_SMALLINT Not Null;

exception Record_Not_Found;

enumeration FailType is (Not_Logged_In, SQL_OK, SQL_Fail);

status fetch_map named is_found uses Failtype is
  -999 .. -300 => SQL_Fail,
  -299, -298 => Not_Logged_In,
  0 => SQL_OK,
  100 => raise record_not_found);

end t_15;

with t_15; use t_15;
schema module s_15 is
  table Members is
    MemberName not null : MemberName,
    MemberSSN not null : SSN,
    ClubNumber not null : Club_Number,
    MemberAge : Age,
    MemberSex : Sex,
    MemberPhone : Phone,
    MemberStreet : Street,
    MemberCity : City,
    MemberCnty not null : County
  end Members;

end s_15;

with t_15; use t_15;
abstract module a_15 is
  authorization s_15
  record MemberRec is
    MemberName : MemberName;
    MemberSSN : SSN;
    ClubNumber : Club_Number;
    MemberAge : Age;
    MemberSex : Sex;
    MemberPhone : Phone;
    MemberStreet : Street;
    MemberCity : City;
    MemberCnty : County;
  end;

Intermetrics, Inc.
procedure CommitWork is
    commit work;
end CommitWork;

procedure MemberInsert is
    insert into s_15.Members
    from Row : MemberRec VALUES;
cursor MemberSelect (Req_MemberSSN : SSN) for
    select *
    from s_15.Members
    where s_15.Members.MemberSSN = Req_MemberSSN;
    is
        procedure FetchIt is
            fetch into Row : new MemberRec
            status Fetch_Map named Rec_Status;
    end MemberSelect;
end a_15;

A.1.35 t3/t10.sme

definition module t_5 is
    -- Member Information
    domain MemberName is new SQL_CHAR Not Null (Length => 30);
    domain SSN is new SQL_CHAR Not Null (Length => 9);
    domain Age is new SQL_SMALLINT (FIRST => 1, LAST => 199);
    enumeration SexEnum is (F, M);
    domain Sex is new SQL_ENUMERATION_AS_INT (Enumeration => SexEnum, Map => Pos);
    domain Phone is new SQL_CHAR (Length => 8);
    domain Street is new SQL_CHAR (Length => 30);
    domain City is new SQL_CHAR (Length => 15);
    domain County is new SQL_CHAR Not Null (Length => 2);
    domain Club_Number is new SQL_SMALLINT Not Null;
    exception Record_Not_Found;
    enumeration FailType is (Not_LoggedIn, SQL_OK, SQL_Fail);
    status fetch_map named is_found uses Boolean is
        ( -999 .. -300 => False,
          0 => True,
          100 => raise record_not_found);
end t_5;

with t_5; use t_5;
schema module s_5 is
    table Members is
        MemberName not null : MemberName,
        MemberSSN not null : SSN,
ClubNumber not null : Club_Number,
MemberAge : Age,
MemberSex : Sex,
MemberPhone : Phone,
MemberStreet : Street,
MemberCity : City,
MemberCnty not null : County
end Members;
end s_5;

with t_5; use t_5;
abstract module a_5 is
  authorization s_5

  record MemberRec is
    MemberName : MemberName;
    MemberSSN : SSN;
    ClubNumber : Club_Number;
    MemberAge : Age;
    MemberSex : Sex;
    MemberPhone : Phone;
    MemberStreet : Street;
    MemberCity : City;
    MemberCnty : County;
  end;

  procedure CommitWork is
    commit work;

  procedure MemberInsert is
    insert into s_5.Members
    from Row : MemberRec VALUES;

  cursor MemberSelect (Req_MemberSSN : SSN) for
    select *
    from s_5.Members
    where s_5.Members.MemberSSN = Req_MemberSSN;
  is
    procedure FetchIt is
      fetch into Row : new MemberRec
        status Fetch_Map named Rec_Status;
  end MemberSelect;
end a_5;

A.1.36  t3/t11.sme

definition module t_6 is
  -- Member Information
  domain MemberName is new SQL_CHAR Not Null (Length => 30);
  domain SSN is new SQL_CHAR Not Null (Length => 9);
  domain Age is new SQL_SMALLINT (FIRST => 1, LAST => 199);
  enumeration SexEnum is (F, M);
  domain Sex is new SQL_ENUMERATION_AS_INT (}
Enumeration => SexEnum, Map => Pos);

domain Phone is new SQL_CHAR (Length => 8);
domain Street is new SQL_CHAR (Length => 30);
domain City is new SQL_CHAR (Length => 15);

domain County is new SQL_CHAR Not Null (Length => 2);
domain Club_Number is new SQL_SMALLINT Not Null;

exception Record_Not_Found;

enumeration FailType is (NotLoggedIn, SQLOk, SQLFail);

status fetch_map named is_found uses Boolean is
( -999 .. -300 => False,
  0 => True,
  100 => raise record_not_found);

end t_6;

with t_6; use t_6;

schema module s_6 is
  table Members is
    MemberName not null : MemberName,
    MemberSSN not null : SSN,
    ClubNumber not null : Club_Number,
    MemberAge : Age,
    MemberSex : Sex,
    MemberPhone : Phone,
    MemberStreet : Street,
    MemberCity : City,
    MemberCnty not null : County
  end Members;

end s_6;

with t_6; use t_6;

abstract module a_6 is
  authorization s_6

  record MemberRec is
    MemberName : MemberName;
    MemberSSN : SSN;
    ClubNumber : Club_Number;
    MemberAge : Age;
    MemberSex : Sex;
    MemberPhone : Phone;
    MemberStreet : Street;
    MemberCity : City;
    MemberCnty : County;
  end;

  procedure CommitWork is
    commit work;

end.
procedure MemberInsert is
   insert into s_6.Members
   from Row : MemberRec VALUES;

   cursor MemberSelect (Req_MemberSSN : SSN) for
   select *
   from s_6.Members
   where s_6.Members.MemberSSN = Req_MemberSSN;
end a_6;

A.1.37  t3/t12.sme

-- ********************************************************************
-- *** Test I
-- ********************************************************************

DEFINITION MODULE D_cI IS

-- the previous line tests the newline separator
--
-- testing full character set
--
DOMAIN Character_set_domain IS
   NEW SQL_CHAR(length => 43);
CONSTANT letters : character_set_domain
   IS 'the quick brown fox jumps over the lazy dog';
CONSTANT all_caps : character_set_domain
   IS 'THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG';
CONSTANT digit_as_char : Character_set_domain
   IS '1234567890';
CONSTANT digits_as_num
   IS 1234567890;

DOMAIN integer_domain IS
   NEW SQL_INT;
DOMAIN real_domain IS
   NEW SQL_REAL;
CONSTANT integerLiteral : integer_domain
   IS (12-4+5*2);
CONSTANT realLiteral : real_domain
   IS 12.4567/.09 + 1. ;
CONSTANT float_literal
   IS (0.1E1) + (10.E-1) + (.1E1) ;

ENUMERATION Loan_types IS
   ( mortgage,
     auto,
     personal);

DOMAIN Loan_type_domain IS
   NEW SQL_ENUMERATION_AS_CHAR
   (ENUMERATION => Loan_types, MAP => IMAGE);

   CONSTANT personal_loan : loan_type_domain
      IS personal ;

END D_cI;
definition module t_16 is
   -- Member Information
   domain MemberName is new SQL_CHAR Not Null (Length => 30);
   domain SSN is new SQL_CHAR Not Null (Length => 9);
   domain Age is new SQL_SMALLINT (FIRST => 1, LAST => 199);

   enumeration SexEnum is (F, M);
   domain Sex is new SQL_ENUMERATION_AS_INT (Enumeration => SexEnum, MAP => pos);

   domain Phone is new SQL_CHAR (Length => 8);
   domain Street is new SQL_CHAR (Length => 30);
   domain City is new SQL_CHAR (Length => 15);

   domain County is new SQL_CHAR Not Null (Length => 2);
   domain ClubNumber is new SQL_SMALLINT Not Null;

   exception Record_Not_Found;

   enumeration FailType is (NotLogged_In, SQL_Ok, SQL_Fail);

   status fetch_map named is_found uses Failtype is
      (-999 .. -300 => SQL_Fail,
       -299, -298 => NotLogged_In,
       0 => SQL_Ok,
       100 => raise record_not_found);

end t_16;

with t_16; use t_16;
schema module s_16 is
   table Members is
      MemberName not null : MemberName,
      MemberSSN not null : SSN,
      ClubNumber not null : ClubNumber,
      MemberAge : Age,
      MemberSex : Sex,
      MemberPhone : Phone,
      MemberStreet : Street,
      MemberCity : City,
      MemberCnty not null : County
end Members;

end s_16;

with t_16; use t_16;
abstract module a_16 is
   authorization s_16

   record MemberRec is
      MemberName : MemberName;
      MemberSSN : SSN;

   end record MemberRec;

end a_16;

Intermetrics, Inc.
Appendix A - Compiler Test Suite Source Code

ClubNumber : Club_Number;
MemberAge : Age;
MemberSex : Sex;
MemberPhone : Phone;
MemberStreet : Street;
MemberCity : City;
MemberCnty : County;
end;

cursor MemberSelect (Req_MemberSSN : SSN) for
select t1.MemberSSN, t2.MemberName
from s_16.Members as t1, s_16.Members as t2
where t1.MemberSSN = t2.MemberSSN;
end a_16;

A.1.39 t3/t14.sme

definition module t_7 is
-- Member Information
domain MemberName is new SQL_CHAR Not Null (Length => 30);
domain SSN is new SQL_CHAR Not Null (Length => 9);
domain Age is new SQL_SMALLINT (FIRST => 1, LAST => 199);

enumeration SexEnum is (F, M);
domain Sex is new SQL_ENUMERATION_AS_INT (Enumerations => SexEnum,
Map => Pos);

domain Phone is new SQL_CHAR (Length => 8);
domain Street is new SQL_CHAR (Length => 30);
domain City is new SQL_CHAR (Length => 15);

domain County is new SQL_CHAR Not Null (Length => 2);
domain Club_Number is new SQL_SMALLINT Not Null;

exception Record_Not_Found;

enumeration FailType is (Not_Logged_In, SQL_Ok, SQL_Fail);

status fetch_map named is_found uses Failtype is
( -999 .. -300 => SQL_Fail,
 -299, -298 => Not_Logged_In,
 0 => SQL_Ok,
 100 => raise record_not_found);

status bool_map named not_found uses boolean is
(100=>true, 0=>false);
end t_7;

A.2 End-to-End Tests

The end-to-end tests are comprised of the SAMeDL file t2.sme and the Ada application program file driver.a. Also included for convenience are the database schema files initbank.sql and initisql used for initially setting up the database.
A.2.1 t2/t2.sme

DEFINITION MODULE Bank_def IS

-- enumeration declarations

ENUMERATION Branches IS
  ( Bethesda, Silver_Spring, Gaithersburg, Potomac);

ENUMERATION Loan_types IS
  ( mortgage, auto, personal);

-- domain character declarations

DOMAIN Customer_name_domain IS
  NEW SQL_CHAR(length => 15);
DOMAIN Addr_domain IS
  NEW SQL_CHAR(length => 15);
DOMAIN City_domain IS
  NEW SQL_CHAR(length => 15);
DOMAIN State_domain IS
  NEW SQL_CHAR(length => 2);

-- domain integer declarations

DOMAIN SSN_domain IS
  NEW SQL_INT NOT NULL ( FIRST => 0, LAST => 999999999);
DOMAIN acct_num_domain IS
  NEW SQL_SMALLINT NOT NULL ( FIRST => 0, LAST => 999);

-- domain real declarations

DOMAIN Balance_domain IS
  NEW SQL_REAL;
DOMAIN Interest_rate_domain IS
  NEW SQL_REAL( FIRST => 0.0, LAST => 1.0);
DOMAIN Loan_payment_domain IS
  NEW SQL_REAL;
DOMAIN Branch_assets_domain IS
  NEW SQL_REAL;
--- domain enumeration declarations

DOMAIN Loan_type_domain IS
  NEW SQL_ENUMERATION AS int
  (ENUMERATION => Loan_types, MAP => POS);

DOMAIN branch_num_domain IS
  NEW SQL_ENUMERATION AS int
  (ENUMERATION => Branches, MAP => POS);

--- record definitions

RECORD Customer_record IS
  Cust_Name : Customer_name_domain;
  SSN : SSN_domain;
  Street : Addr_domain;
  City : City_domain;
  State : State_domain;
END customer_record;

RECORD New_Customer_record IS
  New_Name : Customer_name_domain;
  New_SSN : SSN_domain;
  New_Street : Addr_domain;
  New_City : City_domain;
  New_State : State_domain;
END new_customer_record;

RECORD Fetch_Customer_record IS
  Bank_Cust_Cust_Name : Customer_name_domain;
  Bank_Cust_SSN : SSN_domain;
  Bank_Cust_Street_Addr : Addr_domain;
  Bank_Cust_City_Addr : City_domain;
  Bank_Cust_State_Addr : State_domain;
END fetch_customer_record;

RECORD Savings_entry IS
  branch_num : branch_num_domain;
  acct_num : acct_num_domain;
  Balance : Balance_domain;
  cust_ssn : SSN_domain;
END Savings_entry;

RECORD Chequeing_entry IS
  branch_num : branch_num_domain;
  acct_num : acct_num_domain;
  Balance : Balance_domain;
  cust_ssn : SSN_domain;
END Chequeing_entry;

RECORD loan_entry IS
  branch_num : branch_num_domain;
  acct_num : acct_num_domain;
  Balance : Balance_domain;
  Loan_type : Loan_type_domain;
  cust_ssn : SSN_domain;
END loan_entry;
RECORD Branch_entry IS
  branch_num : branch_num_domain;
  Assets : Branch_assets_domain;
END Branch_entry;

RECORD new_Branch_entry IS
  num : branch_num_domain;
  Assets : Branch_assets_domain;
END new_Branch_entry;

ENUMERATION sql_enum IS (SQL_Found, SQL_Not_Found, SQL_Error);
STATUS fail_type NAMED Is_Found USES sql_enum IS
(-999..-1 => SQL_Error,
  0 => SQL_Found,
  100 => SQL_Not_Found);

END Bank_def;
WITH Bank_def;
USE Bank_def;
SCHEMA MODULE Bank IS
  -- Basic customer information
  TABLE Cust IS
    Cust_Name : Customer_name_domain,
    SSN not null : SSN_domain,
    Street_addr : Addr_domain,
    City_addr : City_domain,
    State_addr : State_domain
  END cust;
  -- Checking account
  TABLE cheque IS
    branch_num : branch_num_domain,
    acct_num not null : acct_num_domain,
    Balance : Balance_domain,
    cust_ssn not null : SSN_domain
  END cheque;
  -- Savings account
  TABLE savings IS
    branch_num : branch_num_domain,
    acct_num not null : acct_num_domain,
    Balance : Balance_domain,
    cust_ssn not null : SSN_domain
  END savings;
  -- loan account
  TABLE loan IS
    branch_num : branch_num_domain,
    acct_num not null : acct_num_domain,
    Balance : Balance_domain,
    Loan_type : loan_type_domain,
    cust_ssn not null : SSN_domain
  END loan;
  -- Branch information
  TABLE Branch IS
    num : branch_num_domain,
    Assets : Branch_assets_domain
  END Branch;
END Bank;
WITH Bank_def;
USE Bank_def;
EXTENDED ABSTRACT MODULE Bank_proc IS
  AUTHORIZATION Bank
  --
  -- procedures
  --
  -- commit statement
  --
  PROCEDURE Commit_work .IS
    COMMIT WORK;
  --
  -- delete statement
  --
  PROCEDURE Delete_customer_loan
    (loan_number_in : acct_num_domain) IS
    DELETE FROM
      Bank.Loan
    WHERE
      Bank.Loan.acct_num = loan_number_in;
  --
  -- rollback statement
  --
  PROCEDURE rollback_work IS
    ROLLBACK WORK;
  --
  -- update statement
  --
  PROCEDURE Up_save_acct_bal
    (acct_num_in : acct_num_domain;
      transaction : balance_domain )
    IS
    UPDATE
      Bank.savings
    SET
      Bank.savings.balance =
        Bank.savings.balance + transaction
    WHERE
      Bank.savings.acct_num = acct_num_in;

PROCEDURE S_and_L IS
  UPDATE
    Bank.Loan
  SET
    Bank.Loan.balance = 0.0;
  --
  -- insert statement (query)
  --
  PROCEDURE move_cheque_to_save
    (account_num_in : acct_num_domain)
  IS
    INSERT INTO
      Bank.savings
    SELECT *
    FROM
Bank.cheque
WHERE
  Bank.cheque.acct_num >= account_num_in;
PROCEDURE Get_cust_profile (SSN_in : SSN_domain) IS
  SELECT *
  INTO
    Customer_Profile : customer_record
  FROM
    Bank.cust
  WHERE
    Bank.cust.SSN = SSN_in;

PROCEDURE Get_save_record
  (acct_num_in : acct_num_domain) IS
  SELECT *
  INTO
    savings_record : savings_entry
  FROM
    Bank.savings
  WHERE
    Bank.savings.acct_num = acct_num_in;

CURSOR customer_accounts(SSN_in : SSN_domain) FOR
  SELECT
    Bank.savings.cust_ssn,
    Bank.savings.acct_num,
    Bank.savings.balance
  FROM
    Bank.savings
  WHERE
    Bank.savings.cust_ssn = SSN_in ;

CURSOR loans_over (loan_balance_in : balance_domain) FOR
  SELECT
    Bank.Loan.acct_num,
    Bank.Loan.branch_num,
    Bank.Loan.cust_ssn,
Appendix A - Compiler Test Suite Source Code

Bank.Loan.balance
FROM
Bank.Loan.
WHERE
Bank.Loan.balance >= loan_balance_in;
-- comparison predicate <=
--
CURSOR loans_under (loan_balance_in : balance_domain) FOR
SELECT
    Bank.Loan.acct_num,
    Bank.Loan.branch_num,
    Bank.Loan.cust_ssn,
    Bank.Loan.balance
FROM
    Bank.Loan
WHERE
    Bank.Loan.balance <= loan_balance_in;
-- comparison predicate >
--
CURSOR cheque_bal_over ( account_bal_in : Balance_domain ) FOR
SELECT
    Bank.cheque.acct_num,
    Bank.cheque.balance
FROM
    Bank.cheque
WHERE
    Bank.cheque.balance > account_bal_in;
-- comparison predicate <
--
CURSOR save_bal_under ( account_bal_in : Balance_domain ) FOR
SELECT
    Bank.savings.acct_num,
    Bank.savings.balance
FROM
    Bank.savings
WHERE
    Bank.savings.balance < account_bal_in;
-- comparison predicate <>
--
CURSOR other_branches
    ( branch_num_in : branch_num_domain ) FOR
SELECT
    Bank.Branch.num
FROM
    Bank.Branch
WHERE
    Bank.Branch.num <> branch_num_in;
-- between predicate
--
CURSOR large_deposits
    ( lower_bound : balance_domain; upper_bound : balance_domain ) FOR
SELECT *
FROM Bank.savings
WHERE Bank.savings.balance
    BETWEEN lower_bound AND upper_bound;

-- not between predicate
--
CURSOR large_loans
( lower_bound : balance_domain; upper_bound : balance_domain ) FOR
SELECT
    Bank.Loan.acct_num,
    Bank.Loan.balance,
    Bank.Loan.cust_ssn
FROM Bank.Loan
WHERE Bank.Loan.balance NOT BETWEEN lower_bound AND upper_bound;

-- like predicate
--
CURSOR find_customer ( name_in : customer_name_domain ) FOR
SELECT
    Bank.cust.cust_name
FROM Bank.cust
WHERE Bank.cust.cust_name LIKE name_in
ORDER BY Bank.cust.cust_name;

-- exists predicate/null predicate/subquery/quantified predicate
--
CURSOR Cust_Null_Cnt FOR
SELECT ssn
FROM Cust t1, Savings t2
WHERE t1.street_addr IS NULL AND
t1.ssn = t2.cust_ssn AND
t2.balance > SOME ( SELECT balance FROM bank.savings ) AND
EXISTS ( SELECT * FROM loan WHERE loan.cust_ssn=t1.ssn )
;

-- in predicate
--
CURSOR Loan_count ( Branch_in: branch_num_domain ) FOR
SELECT
FROM
Bank.Loan
WHERE
Bank.Loan.Branch_num IN (Branch_in);

-- cursor procs

CURSOR customer_list FOR
SELECT *
FROM
Bank.cust;
IS
PROCEDURE open_customer IS
OPEN customer_list;

PROCEDURE close_customer IS
CLOSE customer_list;

PROCEDURE fetch_customer IS
FETCH customer_list
INTO next_customer : fetch_customer_record
STATUS fail_type;

PROCEDURE update_customer (new_street : Addr_domain) IS
UPDATE Bank.cust
SET Bank.cust.street_addr = new_street
WHERE CURRENT OF customer_list;

PROCEDURE update_customer_null IS
UPDATE Bank.cust
SET Bank.cust.street_addr = null
WHERE CURRENT OF customer_list;

PROCEDURE delete_customer IS
DELETE FROM Bank.cust;

END customer_list;
procedures and cursors used to initialize the database and verify the contents of tables after test transactions

EXTENDED PROCEDURE Connect_Bank is
  CONNECT Bank;

PROCEDURE New_customer IS
  INSERT INTO
    Bank.cust (cust_name named new_name,
               ssn named new_ssn,
               street_addr named new_street,
               city_addr named new_city,
               state_addr named new_state)
  FROM
    New_customer_info : new_customer_record
  VALUES;

PROCEDURE New_chequeing IS
  INSERT INTO
    Bank.cheque
  FROM
    New_chequeing_info : chequeing_entry
  VALUES;

PROCEDURE New_savings IS
  INSERT INTO
    Bank.savings
  FROM
    New_savings_info : savings_entry
  VALUES;

PROCEDURE New_loan IS
  INSERT INTO
    Bank.Loan
  FROM
    New_loan_info : loan_entry
  VALUES;

PROCEDURE New_branch IS
  INSERT INTO
    Bank.Branch
  FROM
    New_branch_info : new_branch_entry
  VALUES (num, assets);

PROCEDURE Delete_customers IS
  DELETE FROM
    Bank.cust;

PROCEDURE Delete_chequeing IS
  DELETE FROM
    Bank.cheque;

PROCEDURE Delete_savings IS
  DELETE FROM
    Bank.savings;
PROCEDURE Delete_loans IS
    DELETE FROM
        Bank.Loan;

PROCEDURE Delete_Branches IS
    DELETE FROM
        Bank.Branch;
CURSOR List_customers FOR
SELECT *
FROM
    Bank.cust
ORDER BY
    Bank.cust.SSN
;

CURSOR List_chequeing FOR
SELECT *
FROM
    Bank.cheque
ORDER BY
    Bank.cheque.acct_num
;

CURSOR List_savings FOR
SELECT *
FROM
    Bank.savings
ORDER BY
    Bank.savings.acct_num
;

CURSOR List_loans FOR
SELECT *
FROM
    Bank.Loan
ORDER BY
    Bank.Loan.acct_num
;

CURSOR List_branches FOR
SELECT *
FROM
    Bank.Branch
ORDER BY
    Bank.Branch.num
;

END Bank_proc;

A.2.2 t2/driver.a

with text_io;
with Bank_def;
with Bank_proc;

procedure Test_driver is

use text_io;
use Bank_def;
use Bank_proc;
use customer_name_domain_ops;
use addr_domain_ops;
use city_domain_ops;
use state_domain_ops;
use balance_domain_ops;
use branch_assets_domain_ops;

-- I/O declarations

subtype prompt_length is integer range 1..60;
subtype prompt_line is string (1..prompt_length'last);
subtype prompt_index is integer range 1..9;
type test_count is range 1..18;
type prompt_array is array(prompt_index) of prompt_line;
beginning : constant string := "Beginning SAmeDL T2 test program";
heading : constant string := "TEST DESCRIPTION TEST DESCRIPTION";
testlist : constant prompt_array :=
  prompt_array'((OPT1 CT4
  (*PT2 CT5
  (*PT3 CT6
  (*PT4 CT7
  (*PT5 CT8
  (*PT6 CT9
  (*CT1 CT10
  (*CT2 CT11
  (*CT3

prompt1: constant string :=
  "Enter starting test number or A to run entire test set:";
answer: string(1..80):=
  "

line: prompt_index;
prompt2: constant string := "Try again (test letters must be caps):";
prompt_count : integer range 1..10 := 1;
length: natural range 0..80;
test_number : test_count := 1;
found : boolean := false;

-- database interaction declarations

type customer_index is range 1..16;
type checking_index is range 1..14;
type savings_index is range 1..17;
type loan_index is range 1..12;
type branch_index is range 1..4;
type customer_rec is
  record
    name : customer_name_domain_not_null;
    ssn : ssn_domain_not_null;
    street : addr_domain_not_null;
    city : city_domain_not_null;
    state : state_domain_not_null;
  end record;

type Savings_rec is
  record
    acct_num : acct_num_domain_not_null;
    balance : balance_domain_not_null;
  end record;

Intermetrics, Inc. 115
cust_ssn : ssn_domain_not_null;
branch_num : branch_num_domain_not_null;
end record;

type Checking_rec IS
record
acct_num : acct_num_domain_not_null;
balance : balance_domain_not_null;
cust_ssn : ssn_domain_not_null;
branch_num : branch_num_domain_not_null;
end record;

type loan_rec IS
record
acct_num : acct_num_domain_not_null;
balance : balance_domain_not_null;
loan_type : loan_type_domain_not_null;
cust_ssn : ssn_domain_not_null;
branch_num : branch_num_domain_not_null;
end record;

type Branch_rec IS
record
branch_num : branch_num_domain_not_null;
assets : branch_assets_domain_not_null;
end record;

type customer_array is array (customer_index) of customer_rec;
type checking_array is array (checking_index) of checking_rec;
type savings_array is array (savings_index) of savings_rec;
type loan_array is array (loan_index) of loan_rec;
type branch_array is array (branch_index) of branch_rec;

customers : constant customer_array :=
  ("Scarlett",111111111,"Byrd","Potomac"),
  ("Smith",111345678,"North","Gaithersburg"),
  ("Glenn",123456789,"Sand Hill","Potomac"),
  ("Green",123546789,"Walnut","Bethesda"),
  ("Brown",123987654,"Chestnut","Alexandria"),
  ("Plum",132549876,"Bethesda","Bethesda"),
  ("Mustard",222222222,"Spice","Hyattsville"),
  ("Jones",222345678,"Main","Arlington"),
  ("Hayes",333345678,"Bridge","Potomac"),
  ("Curry",444345678,"Tree","Columbia"),
  ("Lindsay",555345678,"Park","Bethesda"),
  ("Turner",666345678,"Putnam","Fairfax"),
  ("Va"),...
Appendix A - Compiler Test Suite Source Code

("Williams", 777345678, "Nassau", "Gaithersburg",
"Adams", 888345678, "Spring", "Silver Spring",
"Brooks", 987654321, "Senator", "Washington",
"Johnson", 999345678, "Alma", "Silver Spring")

checking_list : constant checking_array := checking_array'
((1.000.0, 222345678, bethesda),
(2, 50.0, 111345678, gaithersburg),
(3, 50000.0, 333345678, potomac),
(4, 300.0, 444345678, silver_spring),
(5, 250.0, 555345678, bethesda),
(6, 500.0, 777345678, gaithersburg),
(7, 1500.0, 888345678, silver_spring),
(8, 15000.0, 123456789, potomac),
(9, 1250.0, 987654321, bethesda),
(10, -10.0, 123546789, silver_spring),
(11, 150.0, 123546789, gaithersburg),
(12, 350.0, 123987654, potomac),
(13, 4500.0, 132549876, bethesda),
(14, 40000.0, 222222222, gaithersburg));

savings_list : constant savings_array := savings_array'
((101, 3000.0, 222345678, bethesda),
(102, 4000.0, 111345678, gaithersburg),
(103, 2000.0, 111345678, potomac),
(104, 50.0, 333345678, potomac),
(105, 1500.0, 444345678, silver_spring),
(106, 50000.0, 555345678, bethesda),
(107, 50000.0, 555345678, silver_spring),
(108, 50000.0, 555345678, potomac),
(109, 200.0, 666345678, gaithersburg),
(110, 300.0, 777345678, gaithersburg),
(111, 200.0, 777345678, silver_spring),
(112, 4000.0, 999345678, silver_spring),
(113, 200.0, 123456789, potomac),
(114, 60000.0, 987654321, bethesda),
(115, 10000.0, 123546789, bethesda),
(116, 50000.0, 132549876, potomac),
(117, 20.0, 222222222, gaithersburg));

loan_list : constant loan_array := loan_array'
((201, 3000.0, personal, 111345678, gaithersburg),
(202, 300000.0, mortgage, 333345678, potomac),
(203, 32000.0, auto, 444345678, bethesda),
(204, 150000.0, mortgage, 555345678, bethesda),
(205, 1500.0, personal, 777345678, gaithersburg),
(206, 1500.0, personal, 999345678, potomac),
(207, 50000.0, mortgage, 123456789, potomac),
(208, 320000.0, mortgage, 987654321, bethesda),
(209, 180000.0, mortgage, 123546789, bethesda),
(210, 5000.0, auto, 123987654, potomac),
(211, 240000.0, mortgage, 132549876, potomac),
(212, 14000.0, auto, 111111111, silver_spring));
SAMeDL Development Environment - Test Plan

branch_list : constant branch_array := branch_array'(
(bethesda,814250.0),
silver_spring,71490.0),
(gaithersburg,49720.0),
potomac,764100.0));

fetch_results : boolean := false;
test_results : boolean := false;
current_test : string(1..4) := " ";

procedure init_customer_db is

customer_row : customer_index;
next_customer: customer_rec;
new_cust : new_customer_record;

begin
  init_customer: for customer_row in customer_index loop
    next_customer := customers(customer_row);
    assign (new_cust.new_name, with_null(next_customer.name));
    new_cust.new_ssn := next_customer.ssn;
    assign (new_cust.new_street, with_null(next_customer.street));
    assign (new_cust.new_city, with_null(next_customer.city));
    assign (new_cust.new_state, with_null(next_customer.state));
    new_customer (new_cust);
  end loop init_customer;
  commit_work;
end init_customer_db;

procedure verify_customer_db(verified: out boolean) is

begin
  list_customers.open;
  check_customer: for customer_row in customer_index loop
    list_customers.fetch(db_customer, fetch_results);
    in_customer := customers(customer_row);
    is_same := is_same and
    (db_customer.bank_cust_cust_name
     = with_null(in_customer.name)) and
    (db_customer.bank_cust_ssn = in_customer.ssn) and
    (db_customer.bank_cust_street_addr
     = with_null(in_customer.street)) and
    (db_customer.bank_cust_city_addr
     = with_null(in_customer.city)) and
    (db_customer.bank_cust_state_addr
     = with_null(in_customer.state));
  end loop check_customer;
  verified := is_same;
  list_customers.close;
  commit_work;
  return is_same;
end verify_customer_db;
end verify_customer_db;

Procedure init_checking_db is
  input_entry : checking_rec;
  db_entry : chequeing_entry;
  checking_row : checking_index;
begin
  init_checking: for checking_row in checking_index loop
    input_entry := checking_list(checking_row);
    db_entry.acct_num := input_entry.acct_num;
    db_entry.cust_ssn := input_entry.cust_ssn;
    assign (db_entry.balance , with_null(input_entry.balance));
    assign (db_entry.branch_num ,
      with_null(input_entry.branch_num));
    new_chequeing (db_entry);
  end loop init_checking;
  commit_work;
end init_checking_db;

-- procedure verify_checking_db (verified : out boolean ) is
function verify_checking_db return boolean is
  checking_row : checking_index;
  db_checking: list_chequeing.row_type;
  in_checking: checking_rec;
  is_same : boolean := true;
begin
  list_chequeing.open;
  check_checking: for checking_row in checking_index loop
    in_checking := checking_list(checking_row);
    list_chequeing.fetch(db_checking, fetch_results);
    is_same := is_same and
      (db_checking.bank_cheque.acct_num = in_checking.acct_num) and
      (db_checking.bank_cheque.cust_ssn = in_checking.cust_ssn) and
      (db_checking.bank_cheque.balance = with_null(in_checking.balance))
    and
      (db_checking.bank_cheque.branch_num = with_null(in_checking.branch_num));
  end loop check_checking;
  --
  verified := is_same;
  list_chequeing.close;
  commit_work;
  return is_same;
end verify_checking_db;

procedure init_savings_db is
  input_entry : savings_rec;
  db_entry : savings_entry;
  savings_row : savings_index;
begin
  init_savings: for savings_row in savings_index loop
    input_entry := savings_list(savings_row);
    db_entry.acct_num := input_entry.acct_num;
    db_entry.cust_ssn := input_entry.cust_ssn;
    assign (db_entry.balance , with_null(input_entry.balance));
assign (db_entry.branch_num, with_null(input_entry.branch_num));
new_savings (db_entry);
end loop init_savings;
commit_work;
end init_savings_db;

-- procedure verify_savings_db (verified : out boolean) is
function verify_savings_db return boolean is

savings_row : savings_index;
db_savings: list_savings.row_type;
in_savings: savings_rec;
is_same : boolean := true;

begin
list_savings.open;
check_savings: for savings_row in savings_index loop
in_savings := savings_list(savings_row);
list_savings.fetch(db_savings, fetch_results);
is_same := is_same and
(db_savings.bank_savings_acct_num = in_savings.acct_num) and
(db_savings.bank_savings_cust_ssn = in_savings.cust_ssn) and
(db_savings.bank_savings_balance = with_null(in_savings.balance))
and
(db_savings.bank_savings_branch_num =
with_null(in_savings.branch_num));
end loop check_savings;
list_savings.close;
commit_work;
return is_same;
end verify_savings_db;

procedure init_loan_db is

input_entry : loan_rec;
db_entry : loan_entry;
loan_row : loan_index;

begin
init_loan: for loan_row in loan_index loop
input_entry := loan_list(loan_row);
db_entry.acct_num := input_entry.acct_num;
db_entry.cust_ssn := input_entry.cust_ssn;
assign (db_entry.loan_type,
       with_null(input_entry.loan_type));
assign (db_entry.balance, with_null(input_entry.balance));
assign (db_entry.branch_num,
       with_null(input_entry.branch_num));
new_loan (db_entry);
end loop init_loan;
commit_work;
end init_loan_db;

-- procedure verify_loan_db (verified : out boolean) is
function verify_loan_db return boolean is

loan_row : loan_index;
Appendix A - Compiler Test Suite Source Code

```plaintext
db_loan: list_loans.row_type;
in_loan: loan_rec;
is_same : boolean := true;

begin
list_loans.open;
check_loan: for loan_row in loan_index loop
  in_loan := loan_list(loan_row);
  list_loans.fetch(db_loan, fetch_results);
  is_same := is_same and
    (db_loan.bank_loan_acct_num = in_loan.acct_num) and
    (db_loan.bank_loan_cust_ssn = in_loan.cust_ssn) and
    (db_loan.bank_loan_loan_type = with_null(in_loan.loan_type)) and
    (db_loan.bank_loan_balance = with_null(in_loan.balance)) and
    (db_loan.bank_loan_branch_num = with_null(in_loan.branch_num));
end loop check_loan;
list_loans.close;
conunit_work;
return is_same;
end verify_loan_db;

procedure init_branch_db is
  input_entry : branch_rec;
  db_entry : new_branch_entry;
  branch_row : branch_index;

begin
  init_branch: for branch_row in branch_index loop
    input_entry := branch_list(branch_row);
    assign (db_entry.num, with_null(input_entry.branch_num));
    assign (db_entry.assets, with_null(input_entry.assets));
    new_branch (db_entry);
  end loop init_branch;
  commit_work;
end init_branch_db;

-- procedure verify_branch_db (verified : out boolean) is
function verify_branch_db return boolean is
  branch_row : branch_index;
  db_branch: list_branches.row_type;
  in_branch: branch_rec;
is_same : boolean := true;

begin
list_branches.open;
check_branch: for branch_row in branch_index loop
  in_branch := branch_list(branch_row);
  list_branches.fetch(db_branch, fetch_results);
  is_same := is_same and
    (db_branch.bank_branch_num = with_null(in_branch.branch_num))
    and
    (db_branch.bank_branch_assets = with_null(in_branch.assets));
```

Intermetrics, Inc. 121
end loop check_branch;
-- verified := is_same;
list_branches.close;
commit_work;
return is_same;
end verify_branch_db;

procedure PT_1 (results : out boolean) is
-- verify delete procedure
--
subtype PT1_index is loan_index range 1..11;
type PT1_array is array (PT1_index) of loan_rec;
loan_list : constant PT1_array := PT1_array'((
(201, 3000.0, personal, 111345678, gaithersburg),
(202, 300000.0, mortgage, 333345678, potomac),
(203, 32000.0, auto, 444345678, bethesda),
(204, 150000.0, mortgage, 555345678, bethesda),
(206, 1500.0, personal, 999345678, potomac),
(207, 50000.0, mortgage, 123456789, potomac),
(208, 320000.0, mortgage, 987654321, bethesda),
(209, 180000.0, mortgage, 123546789, bethesda),
(210, 5000.0, auto, 123987654, potomac),
(211, 240000.0, mortgage, 132549876, potomac),
(212, 14000.0, auto, 111111111, silver_spring));
loan_number : acct_num_domain_not_null := 205;
in_loan : loan_rec;
db_loan : list_loans.row_type;
loan_row : PT1_index := 1;
good_check : boolean := true;
begin
delete_customer_loan (loan_number);
commit_work;
list_loans.open;
while good_check and (loan_row < PT1_index'last) loop
    in_loan := loan_list(loan_row);
    list_loans.fetch(db_loan, fetch_results);
    good_check := good_check and
    (db_loan.bank_loan_acct_num = in_loan.acct_num) and
    (db_loan.bank_loan_cust_ssn = in_loan.cust_ssn) and
    (db_loan.bank_loan_loan_type = with_null(in_loan.loan_type)) and
    (db_loan.bank_loan_balance = with_null(in_loan.balance)) and
    (db_loan.bank_loan_branch_num = with_null(in_loan.branch_num));
    loan_row := loan_row + 1;
end loop;
list_loans.close;
delete_loans;
commit_work;
init_loan_db;
results := good_check;
end PT_1;

procedure PT_2 (results : out boolean) is
-- verify rollback procedure
--
begin
delete_customers;
Appendix A - Compiler Test Suite Source Code

rollback_work;
results := verify_customer_db; --(results);
end PT_2;

procedure PT_3 (results: out boolean) is
--
-- verify select procedure
--
SSN : SSN_domain_not_null := 123456789;
in_profile : customer_rec :=
("Glenn -,123456789,"Sand Hill -,"Potomac
", "Md") ;
db_customer : customer_record;
begi
get_cust_profile (SSN, db_customer);
results := (db_customer.cust_name = with_null(in_profile.name))
and
(db_customer.ssn = in_profile.ssn) and
(db_customer.street = with_null(in_profile.street)) and
(db_customer.city = with_null(in_profile.city)) and
(db_customer.state = with_null(in_profile.state)) ;
rollback_work;
end PT_3;

procedure PT_4 (results: out boolean) is
--
-- verify update procedure (single row)
--
acct_num : acct_num_domain_not_null := 103;
transaction : balance_domain_not_null := 100.0;
savings_record : savings_entry;
new_balance : balance_domain_not_null := 2100.0;
begi
Up_save_acct_bal(acct_num, with_null(transaction));
get_save_record(acct_num, savings_record);
results := with_null(new_balance) = savings_record.balance;
rollback_work;
end PT_4;

procedure PT_5 ( results : out boolean ) is
--
-- verify update procedure (entire table)
--
loan_record : list_loans.row_type;
zero : balance_domain_not_null := 0.0;
is_zero : boolean := true;
begi
s_and_l;
list_loans.open;
verify : for loan_row in loan_index loop
list_loans.fetch(loan_record, fetch_results);
is_zero := is_zero and (loan_record.bank_loan_balance
= with_null(zero));
end loop verify;

Intermetrics, Inc.
results := is_zero;
rollback_work;
end pt_5;

procedure PT_6 ( results : out boolean ) is
  -- verify insert procedure (query)
  type PT6_index is range 1 .. 20;
  type PT6_array is array (PT6_index) of savings_rec;
  answer_array : constant PT6_array := PT6_array'(
    (12, 350.0, 123987654, potomac),
    (13, 4500.0, 132549876, bethesda),
    (14, 40000.0, 222222222, gaithersburg),
    (101, 3000.0, 222345678, bethesda),
    (102, 4000.0, 111345678, gaithersburg),
    (103, 2000.0, 111345678, potomac),
    (104, 50.0, 333345678, potomac),
    (105, 1500.0, 444345678, silver_spring),
    (106, 50000.0, 555345678, bethesda),
    (107, 50000.0, 555345678, silver_spring),
    (108, 50000.0, 555345678, potomac),
    (109, 200.0, 666345678, gaithersburg),
    (110, 300.0, 777345678, gaithersburg),
    (111, 200.0, 777345678, silver_spring),
    (112, 4000.0, 999345678, silver_spring),
    (113, 200.0, 123456789, potomac),
    (114, 60000.0, 987654321, bethesda),
    (115, 10000.0, 123546789, bethesda),
    (116, 50000.0, 132549876, potomac),
    (117, 20.0, 222222222, gaithersburg));
  lower_bound : acct_num_domain_not_null := 12;
  row_index : PT6_index;
  is_same : boolean := true;
  db_savings : list_savings.row_type;
  in_savings : savings_rec;
begin
  move_cheque_to_save (lower_bound);
  list_savings.open;
  verify : for row_index in PT6_index loop
    in_savings := answer_array (row_index);
    list_savings.fetch (db_savings, fetch_results);
    is_same := is_same and
      (db_savings.bank_savings_acct_num = in_savings.acct_num) and
      (db_savings.bank_savings_cust_ssn = in_savings.cust_ssn) and
      (db_savings.bank_savings_balance = with_null (in_savings.balance))
    and
      (db_savings.bank_savings_branch_num =
        with_null (in_savings.branch_num));
  end loop verify;
  list_savings.close;
  rollback_work;
  results := is_same;
end PT_6;

procedure CT_1 ( results : out boolean ) is

-- verify cursor select with comparison predicate

    type CT1_index is range 1..2;
    type CT1_row is
        record
            ssn : ssn_domain_not_null;
            account_num : acct_num_domain_not_null;
            balance : balance_domain_not_null;
        end record;
    type CT1_array is array (CT1_index) of CT1_row;
    answer_array : constant CT1_array := CT1_array'(
        (111345678,102,4000.0),
        (111345678,103,2000.0));
    row_num : CT1_index;
    db_row : customer_accounts.row_type;
    in_row : CT1_row;
    ssn := 111345678;
    verified := true;
    begin
        customer_accounts.open(ssn);
        verify : for row_num in CT1_index loop
            customer_accounts.fetch(db_row, fetch_results);
            in_row := answer_array(row_num);
            verified := verified and
                (db_row.cust_ssn = in_row.ssn) and
                (db_row.acct_num = in_row.account_num) and
                (db_row.balance = with_null(in_row.balance));
        end loop verify;
        results := verified;
        customer_accounts.close;
        rollback_work;
    end CT_1;

procedure CT_2 ( results : out boolean) is
-- verify cursor select with >= predicate

    type CT2_index is range 1..5;
    type CT2_row is
        record
            acct_num : acct_num_domain_not_null;
            balance : balance_domain_not_null;
            ssn : ssn_domain_not_null;
            branch_num : branch_num_domain_not_null;
        end record;
    type CT2_array is array (ct2_index) of CT2_row;
    answer_array : constant CT2_array := CT2_array'(
        (202,300000.0,333345678, potomac),
        (204,150000.0,555345678, bethesda),
        (208,320000.0,987654321, bethesda),
        (209,180000.0,123546789, bethesda),
        (211,240000.0,132549876, potomac));
    row_num : ct2_index;
    in_row : ct2_row;
    db_row : loans_over.row_type;
    verified := true;
    lower_bound : balance_domain_not_null := 150000.0;
begin
loans_over.open(with_null(lower_bound));
verify : for row_num in CT2_index loop
  loans_over.fetch(db_row,fetch_results);
in_row := answer_array(row_num);
verified := verified and
  (db_row.acct_num = in_row.acct_num) and
  (db_row.balance = with_null(in_row.balance)) and
  (db_row.cust_ssn = in_row.ssn) and
  (db_row.branch_nuin = with_null(in_row.branch_nuin));
end loop verify;
results := verified;
loans_over.close;
rollback_work;
end CT_2;

procedure CT_3 (results : out boolean) is
-- verify cursor select with <= predicate
--
type CT3_index is range 1..4;
type CT3_row is
record
  acct_num : acct_num_domain_not_null;
  balance : balance_domain_not_null;
  ssn : ssn_domain_not_null;
  branch_num : branch_num_domain_not_null;
end record;
type CT3_array is array (ct3_index) of CT3_row;
answer_array : constant CT3_array := ct3_array'(
  (201,3000.0,111345678,gaithersburg),
  (205,1500.0,777345678,gaithersburg),
  (206,1500.0,999345678,potomac),
  (210,5000.0,123987654,potomac));
verified : boolean := true;
row_num : ct3_index;
db_row : loans_under.row_type;
in_row : ct3_row;
upper_bound : balance_domain_not_null := 5000.0;
begin
loans_under.open(with_null(upper_bound));
verify : for row_num in CT3_index loop
  loans_under.fetch(db_row,fetch_results);
in_row := answer_array(row_num);
verified := verified and
  (db_row.acct_num = in_row.acct_num) and
  (db_row.balance = with_null(in_row.balance)) and
  (db_row.cust_ssn = in_row.ssn) and
  (db_row.branch_nuin = with_null(in_row.branch_nuin));
end loop verify;
results := verified;
loans_under.close;
rollback_work;
end CT_3;

procedure CT_4 (results : out boolean) is

126

Intermetrics, Inc.
Appendix A - Compiler Test Suite Source Code

-- verify cursor select with > predicate

  type CT4_index is range 1..7;
type CT4_row is record
    acct_num : acct_num_domain_not_null;
    balance : balance_domain_not_null;
  end record;
type CT4_array is array (ct4_index) of CT4_row;
answer_array : constant ct4_array := ct4_array'(
  (3,50000.0),
  (5,2500.0),
  (7,1500.0),
  (8,15000.0),
  (9,1250.0),
  (13,4500.0),
  (14,40000.0));
db_row : cheque_bal_over.row_type;
in_row : ct4_row;
row_num : ct4_index;
verified : boolean := true;
lower_bound : balance_domain_not_null := 1001.0;
begin
  cheque_bal_over.open(with_null(lower_bound));
  verify : for row_num in CT4_index loop
    in_row := answer_array(row_num);
    cheque_bal_over.fetch(db_row, fetch_results);
    verified := verified and
       (db_row.acct_num = in_row.acct_num) and
       (db_row.balance = with_null(in_row.balance)) ;
  end loop verify;
  results := verified;
  cheque_bal_over.close;
  rollback_work;
end CT_4;

procedure CT_5 (results : out boolean) is

-- verify cursor select with < predicate

  type CT5_index is range 1..6;
type CT5_row is record
    acct_num : acct_num_domain_not_null;
    balance : balance_domain_not_null;
  end record;
type CT5_array is array (ct5_index) of CT5_row;
answer_array : constant ct5_array := ct5_array'(
  (104,50.0),
  (109,200.0),
  (110,300.0),
  (111,200.0),
  (113,200.0),
  (117,20.0));
db_row : save_bal_under.row_type;
in_row : ct5_row;
row_num : ct5_index;
verified : boolean := true;
upper_bound : balance_domain_not_null := 1500.0;

begin
save_bal_under.open(with_null(upper_bound));
verify : for row_num in CT5_index loop
  in_row := answer_array(row_num);
  save_bal_under.fetch(db_row.fetch_results);
  verified := verified and
    (db_row.acct_num = in_row.acct_num) and
    (db_row.balance = with_null(in_row.balance));
end loop verify;
results := verified;
save_bal_under.close;
rollback_work;
end CT_5;

procedure CT_6 (results : out boolean) is
  -- verify cursor select with <> predicate
  type CT6_index is range 1..3;
  type CT6_array is array (ct6_index) of branch_num_domain_not_null;
  answer_array : constant ct6_array := ct6_array('(
bethesda),
  (silver_spring),
  (gaithersburg));
  db_row : other_branches.row_type;
  in_row : branch_num_domain_not_null;
  row_num : ct6_index;
  verified : boolean := true;
  branch : branch_num_domain_not_null := potomac;
begin
other_branches.open(with_null(branch));
verify : for row_num in CT6_index loop
  other_branches.fetch(db_row.fetch_results);
  in_row := answer_array(row_num);
  verified := verified and
    (db_row.num = with_null(in_row));
end loop verify;
results := verified;
other_branches.close;
rollback_work;
end CT_6;

procedure CT_7 (results : out boolean) is
  -- verify cursor select with between predicate
  type CT7_index is range 1..5;
  type CT7_array is array (ct7_index) of savings_rec;
  answer_array : constant ct7_array := ct7_array('(
  (106,50000.0,555345678,bethesda),
  (107,50000.0,555345678,silver_spring),
  (108,50000.0,555345678,potomac),
  (114,60000.0,987654321,bethesda),

Intermetrics, Inc.
Appendix A - Compiler Test Suite Source Code

(116,50000.0;132549876,potomac));

db_row : large_deposits.row_type;
upper_bound : balance_domain_not_null := 60000.0;
lower_bound : balance_domain_not_null := 40000.0;
in_row : savings_rec;
row_num : ct7_index;
verified : boolean := true;

begin

large_deposits.open(with_null(lower_bound),with_null(upper_bound))

verify : for row_num in CT7_index loop
    in_row := answer_array(row_num);
    large_deposits.fetch(db_row,fetch_results);
    verified := verified and
    (db_row.bank_savings_acct_num = in_row.acct_num) and
    (db_row.bank_savings_cust_ssn = in_row.cust_ssn) and
    (db_row.bank_savings_balance = with_null(in_row.balance)) and
    (db_row.bank_savings_branch_num = with_null(in_row.branch_num));
end loop verify;
results := verified;
large_deposits.close;
rollback_work;
end CT_7;

procedure CT_8 (results : out boolean) is
--
-- verify cursor select with NOT BETWEEN predicate
--

type CT8_index is range 1..5;
type CT8_row is
    record
        acct_num : acct_num_domain_not_null;
        balance : balance_domain_not_null;
        ssn : ssn_domain_not_null;
    end record;
type CT8_array is array (ct8_index) of CT8_row;
answer_array : constant ct8_array := ct8_array'(
    (202,300000.0,333345678),
    (204,150000.0,555345678),
    (208,320000.0,987654321),
    (209,180000.0,123546789),
    (211,240000.0,132549876));
upper_bound : balance_domain_not_null := 60000.0;
lower_bound : balance_domain_not_null := 20.0;
db_row : large_loans.row_type;
in_row : CT8_row;
row_num : ct8_index;
verified : boolean := true;

begin

large_loans.open(with_null(lower_bound),with_null(upper_bound));

verify : for row_num in CT8_index loop
    in_row := answer_array (row_num);
    large_loans.fetch(db_row,fetch_results);
    verified := verified and
    (db_row.acct_num = in_row.acct_num) and

(db_row.cust_ssn = in_row.ssn) and
(db_row.balance = with_null(in_row.balance));
end loop verify;
results := verified;
large_loans.close;
rollback_work;
end CT_8;

procedure CT_9 (results : out boolean) is

-- verify cursor IN predicate
--
type CT9_index is range 1..5;
type CT9_array is array (ct9_index) of loan_rec;
answer_array := constant ct9_array := ct9_array'
  (202,300000.0,mortgage,333345678,potomac),
  (206,1500.0,personal,999345678,potomac),
  (207,50000.0,mortgage,123456789,potomac),
  (210,5000.0,auto,123987654,potomac),
  (211,240000.0,mortgage,132549876,potomac));
verified := true;
in_row := loan_rec;
db_row := loan_count.row_type;
row_num := ct9_index;
begin
    loan_count.open(with_null(branch));
verify : for row__numi in CT9_index loop
        in_row := answer_array(row__numi);
        loan_count.fetch(db_row,fetch_results);
        verified := verified and
        (db_row.bank_loan_acct_num = in_row.acct_num) and
        (db_row.bank_loan_cust_ssn = in_row.cust_ssn) and
        (db_row.bank_loan_loan_type = with_null(in_row.loan_type)) and
        (db_row.bank_loan_balance = with_null(in_row.balance)) and
        (db_row.bank_loan_branch_num = with_null(in_row.branch_num));
end loop verify;
results := verified;
loan_count.close;
rollback_work;
end CT_9;

procedure CT_10 (results : out boolean) is

-- verify cursor like predicate
--
type CT10_index is range 1..2;
type CT10_array is array (ct10_index) of customer_name_domain_not_null;
answer_array := constant ct10_array := ct10_array'
  (*Johnson
   (*Jones

(*johnson
(*jones

db_row := find_customer.row_type;
in_row := customer_name_domain_not_null;
row_num := ct10_index;
verified := true;
name_in := customer_name_domain_not_null := "J"
Appendix A - Compiler Test Suite Source Code

-- name_in : customer_name_domain_not_null := "J__________";

begin
find_customer.open(with_null(name_in));
verify : for row_num in CT10_index loop
  find_customer.fetch(db_row,fetch_results);
  in_row := answer_array(row_num);
  verified := verified and
  (db_row.cust_name = with_null(in_row));
end loop verify;
results := verified;
find_customer.close;
rollback_work;
end CT_10;

procedure CT_11 (results : out boolean) is
  -- verify cursor procedures
  new_record : constant customer_rec := customer_rec'
  ("Smith", "11345678", "South", "Gaithersburg","Md");
  search_ssn : ssn_domain_not_null := 11345678;
  db_street : addr_domain_type;
  new_street : addr_domain_not_null := "South";
  customer_row : customer_index := 1;
  verified : boolean := false;
  db_row : fetch_customer_record;
  sqlval : sql_enum;

begin
  customer_list.open_customer;
  customer_list.fetch_customer(db_row,sqlval);
  -- Check Status Code
  if sqlval = SQL_NOT_FOUND or sqlval = SQL_ERROR then
  results := false;
  return;
  elsif sqlval = SQL_FOUND then
  customer_row := customer_row + 1;
  else
  results := false;
  return;
  end if;
  while (db_row.bank_cust_ssn /= search_ssn) and
  (customer_row < customer_index'last) loop
  customer_list.fetch_customer(db_row,sqlval);
  -- Check Status Code
  if sqlval = SQL_NOT_FOUND or sqlval = SQL_ERROR then
  results := false;
  return;
  elsif sqlval = SQL_FOUND then
  customer_row := customer_row + 1;
  else
  results := false;
  return;
  end if;
end loop;
if db_row.bank_cust_ssn = search_ssn then
assign (db_street, with_null(new_street));
customer_list.update_customer(db_street);
customer_list.close_customer;
commit_work;
customer_list.open_customer;
customer_row := 1;
customer_list.fetch_customer(db_row, sqlval);

-- Check Status Code
if sqlval = SQL_NOT_FOUND or sqlval = SQL_ERROR then
   results := false;
   return;
elsif sqlval = SQL_FOUND then
   null;
else
   results := false;
   return;
end if;

while (db_row.bank_cust_ssn /= search_ssn) and
   (customer_row < customer_index'last) loop
   customer_list.fetch_customer(db_row, sqlval);

-- Check Status Code
if sqlval = SQL_NOT_FOUND or sqlval = SQL_ERROR then
   results := false;
   return;
elsif sqlval = SQL_FOUND then
   customer_row := customer_row + 1;
else
   results := false;
   return;
end if;
end loop;
verified := (db_row.bank_cust_ssn = search_ssn) and
   (db_row.bank_cust_street_addr = db_street);
customer_list.close_customer;
else
   verified := false;
end if;

-- Update the customer street addr to null
if verified then
   customer_list.open_customer;
customer_list.fetch_customer(db_row, sqlval);

-- Check Status Code
if sqlval = SQL_NOT_FOUND or sqlval = SQL_ERROR then
   results := false;
   return;
elsif sqlval = SQL_FOUND then
   customer_row := customer_row + 1;
else
   results := false;
   return;
end if;

while (db_row.bank_cust_ssn /= search_ssn) and
   (customer_row < customer_index'last) loop
   customer_list.fetch_customer(db_row, sqlval);

-- Check Status Code
if sqlval = SQL_NOT_FOUND or sqlval = SQL_ERROR then
   results := false;
   return;
elsif sqival = SQL_FOUND then
    customer_row := customer_row + 1;
else
    results := false;
    return;
end if;
end loop;
if db_row.bank_cust_ssn = search_ssn then
    customer_list.update_customer_null;
    customer_list.close_customer;
    commit_work;
    customer_list.open_customer;
    customer_row := 1;
    customer_list.fetch_customer(db_row, sqival);

-- Check Status Code
if sqival = SQL_NOT_FOUND or sqival = SQL_ERROR then
    results := false;
    return;
elsif sqival = SQL_FOUND then
    null;
else
    results := false;
    return;
end if;
while (db_row.bank_cust_ssn /= search_ssn) and
      (customer_row < customer_index'last) loop
    customer_list.fetch_customer(db_row, sqival);

-- Check Status Code
if sqival = SQL_NOT_FOUND or sqival = SQL_ERROR then
    results := false;
    return;
elsif sqival = SQL_FOUND then
    customer_row := customer_row + 1;
else
    results := false;
    return;
end if;
end loop;
verified := (db_row.bank_cust_ssn = search_ssn) and
            is_null(db_row.bank_cust_street_addr);
customer_list.close_customer;
else
    verified := false;
end if;
end if;
if verified then
    declare
cust_row : cust_null_count.row_type;
isok : boolean;
begin
cust_null_count.open;
cust_null_count.fetch(cust_row, isok);
if isok = FALSE then
    results := false;
    return;
end if;
results := (cust_row.ssn = search_ssn);
end;
else
    results := false;
end if;
rollback_work;
end CT_11;

begin
put_line(beginning);
put_line(heading);
loop1: for line in prompt_index loop
    put_line(test_list(line));
end loop loop1;
put(prompt1);
while not found and (prompt_count < 10) loop
    prompt_count := prompt_count + 1;
    get_line(answer, length);
case length is
    when 1 =>
        case answer(1) is
            when 'a' | 'A' =>
                found := true;
                test_number := 1;
            when others =>
                put(prompt2);
        end case;
    when 2 =>
        if (answer(1..2) = "al") or
            (answer(1..2) = "Al") or
            (answer(1..2) = "aL") or
            (answer(1..2) = "AL") then
            found := true;
            test_number := 1;
        else
            put(prompt2);
        end if;
    when 3 =>
        if answer(1..3) = "PT1" then
            found := true;
            test_number := 1;
        elsif answer(1..3) = "PT2" then
            found := true;
            test_number := 2;
        elsif answer(1..3) = "PT3" then
            found := true;
            test_number := 3;
        elsif answer(1..3) = "PT4" then
            found := true;
            test_number := 4;
        elsif answer(1..3) = "PT5" then
            found := true;
            test_number := 5;
        elsif answer(1..3) = "PT6" then
            found := true;
            test_number := 6;
        elsif answer(1..3) = "CT1" then
            found := true;
            test_number := 7;
        else
            put(prompt2);
        end if;
elsif answer(1..3) = "CT2" then
    found := true;
    test_number := 8;
elsif answer(1..3) = "CT3" then
    found := true;
    test_number := 9;
elsif answer(1..3) = "CT4" then
    found := true;
    test_number := 10;
elsif answer(1..3) = "CT5" then
    found := true;
    test_number := 11;
elsif answer(1..3) = "CT6" then
    found := true;
    test_number := 12;
elsif answer(1..3) = "CT7" then
    found := true;
    test_number := 13;
elsif answer(1..3) = "CT8" then
    found := true;
    test_number := 14;
elsif answer(1..3) = "CT9" then
    found := true;
    test_number := 15;
else
    put (prompt2);
end if;

when 4 =>
    if answer(1..4) = "CT10" then
        found := true;
        test_number := 16;
    elsif answer(1..4) = "CT11" then
        found := true;
        test_number := 17;
    else
        put (prompt2);
    end if;
end case;
end loop;
if prompt_count >= 10 then
    put_line("exceeded attempt limit: try again later");
else
    connect_bank;
    init_customer_db;
    test_results := verify_customer_db; --(test_results);
    if test_results then
        init_checking_db;
        test_results := verify_checking_db; --(test_results);
        if test_results then
            init_savings_db;
            test_results := verify_savings_db; --(test_results);
            if test_results then
                init_loan_db;
                test_results := verify_loan_db; --(test_results);
                if test_results then

Intermetrics, Inc.
SAMeDL Development Environment - Test Plan

init_branch_db;
test_results := verify_branch_db; --(test_results);
if test_results then
    put_line
        ("databases initialize successfully");
else
    put_line
        ("Branch database failed to initialize");
end if;
else
    put_line
        ("Loan database failed to initialize");
end if;
else
    put_line
        ("Savings database failed to initialize");
end if;
else
    put_line ("Checking database failed to initialize");
end if;
else
    put_line ("Customer database failed to initialize");
end if;

if test_results = true then
    while test_number < test_count'last loop
        case test_number is
            when 1 =>
                current_test := "PT1 ";
                PT_1(test_results);
            when 2 =>
                current_test := "PT2 ";
                PT_2(test_results);
            when 3 =>
                current_test := "PT3 ";
                PT_3(test_results);
            when 4 =>
                current_test := "PT4 ";
                PT_4(test_results);
            when 5 =>
                current_test := "PT5 ";
                PT_5(test_results);
            when 6 =>
                current_test := "PT6 ";
                PT_6(test_results);
            when 7 =>
                current_test := "CT1 ";
                CT_1(test_results);
            when 8 =>
                current_test := "CT2 ";
                CT_2(test_results);
            when 9 =>
                current_test := "CT3 ";
                CT_3(test_results);
            when 10 =>
                current_test := "CT4 ";
                CT_4(test_results);
        end case;
    end while;
end if;
current_test := "CT5 ";
CT_5(test_results);
when 12 =>
current_test := "CT6 ";
CT_6(test_results);
when 13 =>
current_test := "CT7 ";
CT_7(test_results);
when 14 =>
current_test := "CT8 ";
CT_8(test_results);
when 15 =>
current_test := "CT9 ";
CT_9(test_results);
when 16 =>
current_test := "CT10 ";
CT_10(test_results);
when 17 =>
current_test := "CT11 ";
CT_11(test_results);
when others =>
put_line ("we should not be here");
end case;

put (current_test);
if test_results = false then
  put_line(" has failed");
  exit;
else
  put_line(" has passed");
test_number := test_number +1;
end if;
end loop;
put_loop (" the test set is complete");
end if;
end if;
end Test_driver;

A.2.3 t2/initbank.sql

create table cust
  (cust_name CHAR(15),
   SSN INTEGER NOT NULL,
   Street_addr CHAR(15),
   City_addr CHAR(15),
   State_addr CHAR(15));
create table savings
  (Branch_num SMALLINT,
   acct_num SMALLINT NOT NULL,
   Balance DECIMAL(12,2),
   cust_ssn INTEGER NOT NULL);
create table cheque
  (Branch_num SMALLINT,
   acct_num SMALLINT NOT NULL,
   Balance DECIMAL(12,2),
   cust_ssn INTEGER NOT NULL);
create table loan
  (Branch_num SMALLINT,
A.2.4 t2/initisql

create table bank.cust
( Name CHAR(15),
  SSN INTEGER NOT NULL,
  Street_addr CHAR(15),
  City_addr CHAR(15),
  State_addr CHAR(15));
create table bank.savings
( Branch_num SMALLINT,
  acct_num SMALLINT NOT NULL,
  Balance DECIMAL(12,2),
  cust_ssn INTEGER NOT NULL);
create table bank.cheque
( Branch_num SMALLINT,
  acct_num SMALLINT NOT NULL,
  Balance DECIMAL(12,2),
  cust_ssn INTEGER NOT NULL);
create table bank.loan
( Branch_num SMALLINT,
  acct_num SMALLINT NOT NULL,
  Balance DECIMAL(12,2),
  Loan_type SMALLINT,
  cust_ssn INTEGER NOT NULL);
create table bank.branch
( Num SMALLINT,
  Assets DECIMAL(12,2) );

A.3 Error Tests

A.3.1 t1/et1.sme

definition module d_etl is
   -- Member Information
   domain MemName is new SQL_CHAR Not Null (Length => 30);
   domain SSN is new SQL_CHAR Not Null (Length => 9);
   domain Age is new SQL_SMALLINT (FIRST => 1, LAST => 199);

   enumeration SexEnum is (F, M);
   domain Sex is new SQL_ENUMERATION_AS_INT (Enumeration => SexEnum, Map => Pos);

   domain Phone is new SQL_CHAR (Length => 8);
   domain Street is new SQL_CHAR (Length => 30);
   domain City is new SQL_CHAR (Length => 15);

   domain County is new SQL_CHAR Not Null (Length => 2);
   domain Club_Number is new SQL_SMALLINT Not Null;
domain Sum_Domain is new SQL_SMALLINT Not Null;
domain Count_Domain is new SQL_INT;
end d_et1;

with d_et1; use d_et1;
schema module s_recdb is
  table Members is
    MemberName not null : MemName,
    MemberSSN not null : SSN,
    ClubNumber not null : Club_Number,
    MemberAge : Age,
    MemberSex : Sex,
    MemberPhone : Phone,
    MemberStreet : Street,
    MemberCity : City,
    MemberCnty not null : County
  end Members;
end s_recdb;

with d_et1; use d_et1;
abstract module a_etl is
  authorization s_recdb
  record MemberRec is
    R_MemberName : MemName;
    R_Sum : Sum_Domain;
    R_Count : Count_Domain;
  end;

  -- Check row record conformance
  procedure PD_MemberSelect (ReqMemberSSN : SSN) is
    select MemberName, SUM(MemberAge), COUNT(*)
    from s_recdb.Members
    where s_recdb.Members.MemberSSN = ReqMemberSSN;
  end MemberSelect;

  cursor MemberSelect (ReqMemberSSN : SSN) for
    select MemberName, SUM(MemberAge), COUNT(*)
    from s_recdb.Members
    where s_recdb.Members.MemberSSN = ReqMemberSSN;
  is
    procedure FetchIt is
      fetch into Row : MemberRec;
    end FetchIt;
  end MemberSelect;

  cursor D_MemberSelect (ReqMemberSSN : SSN) for
    select MemberName, SUM(MemberAge), COUNT(*)
    from s_recdb.Members
    where s_recdb.Members.MemberSSN = ReqMemberSSN;
  is
    procedure FetchIt is

A.3.2 t1/et2.sme

DEFINITION MODULE d_et2 IS
   DOMAIN Branch_assets_domain IS
      NEW SQL_REAL NOT NULL (L => 0.0, R => 1E+10);
   END d_et2;

A.3.3 t1/et3.sme

DEFINITION MODULE E_et3 IS
   DOMAIN Customer_name_domain IS
      NEW SQL_CHAR(length => 50);
   RECORD Customer_record IS
      CName : Customer_name_domain;
   END customer_record;
   END E_et3;
   WITH E_et3;
   USE E_et3;
   SCHEMA MODULE T1_III IS
      TABLE Customer IS
         CName : Customer_name_domain
      END Customer;
   END T1_III;
   WITH E_et3;
   USE E_et3;
   ABSTRACT MODULE A_et3 IS
      AUTHORIZATION T1_III
      CURSOR find_customer (name_in : customer_name_domain) FOR
         SELECT T1_III.customer.Cname
         FROM T1_III.customer
      WHERE name_in LIKE T1_III.customer.Cname
         -- ~~~~~~~~~~~~~~~~~~~ Can only be input parm, ~
         -- literal, or USER
   END A_et3;

A.3.4 t1/et4.sme

definition module D_et4 is
   -- Member Information
domain MemName is new SQL_CHAR Not Null (Length => 30);
domain SSN is new SQL_CHAR Not Null (Length => 9);
domain Age is new SQL_SMALLINT (FIRST => 1, LAST => 199);

enumeration SexEnum is (F, M);
domain Sex is new SQL_ENUMERATION_AS_INT
  (Enumeration => SexEnum, MAP => POS);

domain Phone is new SQL_CHAR (Length => 8);
domain Street is new SQL_CHAR (Length => 30);
domain City is new SQL_CHAR (Length => 15);

domain County is new SQL_CHAR Not Null (Length => 2);

domain Club_Number is new SQL_SMALLINT Not Null;
domain Sum_Domain is new SQL_SMALLINT Not Null;
domain Count_Domain is new SQL_INT;

end D_et4;

with D_et4; use D_et4;
schema module RecDB is
  table Members is
    MemberName not null : MemName,
    MemberSSN not null : SSN,
    ClubNumber not null : Club_Number,
    MemberAge : Age,
    MemberSex : Sex,
    MemberPhone : Phone,
    MemberStreet : Street,
    MemberCity : City,
    MemberCnty not null : County
  end Members;

  table Members2 is
    MemberName2 not null : MemName,
    MemberSSN2 not null : SSN,
    ClubNumber2 not null : Club_Number,
    MemberAge2 : Age,
    MemberSex2 : Sex,
    MemberPhone2 : Phone,
    MemberStreet2 : Street,
    MemberCity2 : City,
    MemberCnty2 not null : County
  end Members2;

end RecDB;

with D_et4; use D_et4;
abstract module A_et4 is
  authorization RecDB

  record MemberRec is
    R_MemberName : MemName;
    R_Sum : Sum_Domain;
    R_Count : Count_Domain;

Intermetrics, Inc.
end;

procedure PD_MemberSelect (Req_MemberSSN : SSN) is
select MemberName, SUM(MemberAge), COUNT(*)
-- May not be NO_DOMAIN ^
from RecDB.Members
  where RecDB.Members.MemberSSN = Req_MemberSSN;

cursor MemberSelect (Req_MemberSSN : SSN) for
select MemberName, SUM(MemberAge), COUNT(*)
-- May not be NO_DOMAIN ^
from RecDB.Members
  where RecDB.Members.MemberSSN = Req_MemberSSN;

is
  procedure FetchIt is
    fetch into Row : MemberRec;
  end MemberSelect;

cursor D_MemberSelect (ReqMemberSSN : SSN) for
select MemberName, SUM(MemberAge), COUNT(*)
-- May not be NO_DOMAIN ^
from RecDB.Members
  where RecDB.Members.MemberSSN = ReqMemberSSN;

is
  procedure FetchIt is
    fetch;
  end D_MemberSelect;

end A.et4;

A.3.5 t1/et5.sme

-- Check for various type/domain inconsistencies

DEFINITION MODULE d.et5 IS
--
-- enumeration declarations
--
ENUMERATION Branches IS
  (Bethesda,
   Silver_Spring,
   Gaithersburg,
   Potomac);

ENUMERATION Loan_types IS
  (mortgage,
   auto,
   personal);

--
-- domain character declarations
--
DOMAIN Customer_name_domain IS
  NEW SQL_CHAR(length => 15);
DOMAIN Addr_domain IS
  NEW SQL_CHAR(length => 15);
DOMAIN City_domain IS
  NEW SQL_CHAR(length => 15);
DOMAIN State_domain IS
  NEW SQL_CHAR(length => 15);
NEW SQL_CHAR(length => 2);
--
-- domain integer declarations
--

DOMAIN SSN_domain IS
  NEW SQL_INT NOT NULL ( FIRST => 0, LAST => 999999999);
DOMAIn acct_num_domain IS
  NEW SQL_SMALLINT NOT NULL ( FIRST => 0, LAST => 999);
--
-- domain real declarations
--

DOMAIN Balance_domain IS
  NEW SQL_REAL;
DOMAIN Interest_rate_domain IS
  NEW SQL_REAL( FIRST => 0.0, LAST => 1.0);
DOMAIN Loan_payment_domain IS
  NEW SQL_REAL;
DOMAIN Branch_assets_domain IS
  NEW SQL_REAL;
--
-- domain enumeration declarations
--

DOMAIN Loan_type_domain IS
  NEW SQL_ENUMERATION_AS_int
    (ENUMERATION => Loan_types, MAP => POS);
DOMAIN branch_num_domain IS
  NEW SQL_ENUMERATION_AS_INT
    (ENUMERATION => Branches, MAP => POS);
--
-- record definitions
--

RECORD Customer_record IS
  Cust_Name : Customer_name_domain;
  SSN       : SSN_domain;
  Street    : Addr_domain;
  City      : City_domain;
  State     : State_domain;
END customer_record;

RECORD Savings_entry IS
  branch_num : branch_num_domain;
  acct_num   : acct_num_domain;
  Balance    : Balance_domain;
  cust_ssn   : SSN_domain;
END Savings_entry;

RECORD Chequeing_entry IS
  branch_num : branch_num_domain;
  acct_num   : acct_num_domain;
  Balance    : Balance_domain;
  cust_ssn   : SSN_domain;
END Chequeing_entry;

RECORD Loan_entry IS
  branch_num : branch_num_domain;
  acct_num   : acct_num_domain;
  Balance    : Balance_domain;
SAMeDL Development Environment - Test Plan

---

Loan_type : Loan_type_domain;
cust_ssn : SSN_domain;
END loan_entry;

RECORD Branch_entry IS
  branch_num : branch_num_domain;
  Assets : Branch_assets_domain;
END Branch_entry;

END d_et5;

WITH d_et5;
USE d_et5;
SCHEMA MODULE s_et5 IS
  --
  -- Basic customer information
  --
  TABLE Cust IS
    Cust_Name : Customer_name_domain,
    SSN not null : SSN_domain,
    Street_addr : Addr_domain,
    City_addr : City_domain,
    State_addr : State_domain
  END cust;
  --
  -- Checking account
  --
  TABLE cheque IS
    branch_num : branch_num_domain,
    acct_num not null : acct_num_domain,
    Balance : Balance_domain,
    cust_ssn not null : SSN_domain
  END cheque;
  --
  -- Savings account
  --
  TABLE Save IS
    branch_num : branch_num_domain,
    acct_num not null : acct_num_domain,
    Balance : Balance_domain,
    cust_ssn not null : SSN_domain
  END Save;
  --
  -- loan account
  --
  TABLE loan IS
    branch_num : branch_num_domain,
    acct_num not null : acct_num_domain,
    Balance : Balance_domain,
    Loan_type : loan_type_domain,
    cust_ssn not null : SSN_domain
  END loan;
  --
  -- Branch information
  --
  TABLE Branch IS
    num : branch_num_domain,
    Assets : Branch_assets_domain
END d_etS;
END Branch;

END s_et5;

WITH d_et5;
USE d_et5;
ABSTRACT MODULE a_et5 IS
  AUTHORIZATION s_et5

  CURSOR large_deposits1
    (upper_bound :balance_domain) FOR
    SELECT *
    FROM
      s_et5.save
    WHERE
      s_et5.save.balance = (SELECT Cust_SSN FROM s_et5.save
                               where s_et5.save.balance = 0.0)
  -- ERR

  CURSOR large_deposits2
    (upper_bound :balance_domain) FOR
    SELECT *
    FROM
      s_et5.save
    WHERE
      s_et5.save.balance = (SELECT balance FROM s_et5.save
                               where s_et5.save.balance = 0.0)
  -- OK

  CURSOR large_deposits3
    (upper_bound :balance_domain) FOR
    SELECT *
    FROM
      s_et5.save
    WHERE
      s_et5.save.balance = (SELECT * FROM s_et5.save
                               where s_et5.save.balance = 0.0)
  -- ERR

  CURSOR large_deposits4
    (upper_bound :balance_domain) FOR
    SELECT *
    FROM
      s_et5.save
    WHERE
      s_et5.save.balance
        BETWEEN 0 -- ERR
        AND upper_bound

  CURSOR large_deposits5
    (upper_bound :balance_domain) FOR
    SELECT *
    FROM
      s_et5.save

WHERE
  s_et5.save.balance
    BETWEEN interest_rate_domain(0.0) -- ERR
    AND upper_bound;

PROCEDURE Delete_customer_loan
  (loan_number_in : acct_num_domain) IS
  DELETE FROM
    s_et5.Loan
  WHERE
    SSN_domain(s_et5.Loan.acct_num) = loan_number_in; -- ERR

CURSOR loans_over(loan_balance_in : balance_domain) FOR
SELECT
  s_et5.Loan.acct_num,
  s_et5.Loan.branch_num,
  s_et5.Loan.cust_ssn,
  s_et5.Loan.balance
FROM
  s_et5.Loan
WHERE
  s_et5.Loan.balance >= Bethesda -- ERR
;

CURSOR loans_under(loan_balance_in : balance_domain) FOR
SELECT
  s_et5.Loan.acct_num,
  s_et5.Loan.branch_num,
  s_et5.Loan.cust_ssn,
  s_et5.Loan.balance
FROM
  s_et5.Loan
WHERE
  branch_num <= Bethesda -- OK
;

CURSOR large_loans
  (lower_bound : balance_domain; upper_bound :balance_domain) FOR
SELECT
  s_et5.Loan.acct_num,
  s_et5.Loan.balance,
  s_et5.Loan.cust_ssn
FROM
  s_et5.Loan
WHERE
  Bethesda NOT BETWEEN Bethesda AND Bethesda -- ERR
;

-- like predicate
--
CURSOR find_customer (name_in : customer_name_domain) FOR
SELECT
  s_et5.cust.cust_name
FROM
  s_et5.cust
WHERE
   s_et5.cust.cust_name LIKE 2.0 -- ERR
-- in predicate
--
CURSOR Loan_count ( Branch_in: branch_num_domain ) FOR
   SELECT
      *
   FROM
      s_et5.Loan
WHERE
   s_et5.Loan.Branch_num IN (Branch_in, Bethesda, Potomac) -- OK
;
--
CURSOR Loan_count2 ( Branch_in: branch_num_domain ) FOR
   SELECT
      *
   FROM
      s_et5.Loan
WHERE
   s_et5.Loan.Branch_num IN (Branch_in, Bethesda, auto) -- ERR
;
END a_et5;

A.3.6 t1/et6.sme

-- Tests error checking on constant decls

definition module d_et6 is
   -- Member Information
   domain MemName is new SQL_CHAR Not Null (Length => 30);
   domain SSN is new SQL_CHAR Not Null (Length => 9);
   domain Age is new SQL_SMALLINT (FIRST => 1, LAST => 199);

   enumeration SexEnum is (F, M);
   enumeration SexEN is (Female, Male);
   domain Sex is new SQL_ENUMERATION_AS_INT (Enumeration => SexEnum, Map => Pos);

   domain Phone is new SQL_CHAR (Length => 8);
   domain Street is new SQL_CHAR (Length => 30);
   domain City is new SQL_CHAR (Length => 15);

   domain County is new SQL_CHAR Not Null (Length => 2);

   domain Club_Number is new SQL_SMALLINT Not Null;
   domain Money is new SQL_REAL;

   constant C_Name : MemName is '1234567890123456789012345678901'; -- ERR
   constant C_SSN : SSN is 123456789; -- ERR
   constant C_Club_Number : Club_Number is 10.0;
   constant C_Club_Number1: Club_Number is 10.0E+0; -- ERR
   constant C_M1 : Money is 39;
   constant C_M2 : Money is 39.0;
   constant C_M3 : Money is 39.E+0;
constant C_Sex is F; -- ERR
constant C_Sex1: Sex is Female; -- ERR
end d_et6;

A.3.7 t1/et7.sme

-- Check assignment conformance on selects

DEFINITION MODULE d_et7 IS
  --
  -- enumeration declarations

  ENUMERATION Branches IS
     ( Bethesda,
      Silver_Spring,
      Gaithersburg,
      Potomac);

  ENUMERATION Loan_types IS
     ( mortgage,
      auto,
      personal);

  -- domain character declarations

  DOMAIN Customer_name_domain IS
     NEW SQL_CHAR(length => 15);
  DOMAIN Addr_domain IS
     NEW SQL_CHAR(length => 15);
  DOMAIN City_domain IS
     NEW SQL_CHAR(length => 15);
  DOMAIN State_domain IS
     NEW SQL_CHAR(length => 2);

  -- domain integer declarations

  DOMAIN SSN_domain IS
     NEW SQL_Int NOT NULL (FIRST => 0, LAST => 9999999999);
  DOMAIN acct_num_domain IS
     NEW SQL_SMALLINT NOT NULL (FIRST => 0, LAST => 9999);

  -- domain real declarations

  DOMAIN Balance_domain IS
     NEW SQL_REAL;
  DOMAIN Interest_rate_domain IS
     NEW SQL_REAL(FIRST => 0.0, LAST => 1.0);
  DOMAIN Loan_payment_domain IS
     NEW SQL_REAL;
  DOMAIN Branch_assets_domain IS
     NEW SQL_REAL;
domain enumeration declarations

DOMAIN Loan_type_domain IS
    NEW SQL_ENUMERATION AS_int
    (ENUMERATION => Loan_types, MAP => POS);

DOMAIN branch_num_domain IS
    NEW SQL_ENUMERATION AS_INT
    (ENUMERATION => Branches, MAP => POS);

RECORD Customer_record IS
    Cust_Name : Customer_name_domain;
    SSN : SSN_domain;
    Street : Addr_domain;
    City : City_domain;
    State : State_domain;
END customer_record;

END d_et7;
WITH d_et7;
USE d_et7;
SCHEMA MODULE s_et7 IS

--  Basic customer information
--

  TABLE Cust IS
    Cust_Name : Customer_name_domain,
    SSN not null : SSN_domain,
    Street_addr : Addr_domain,
    City_addr : City_domain,
    State_addr : State_domain
  END cust;

--

-- Savings account
--

  TABLE Save IS
    branch_num : branch_num_domain,
    acct_num not null : acct_num_domain,
    Balance : Balance_domain,
    cust_ssn not null : SSN_domain
  END Save;

END s_et7;
WITH d_et7;
USE d_et7;
ABSTRACT MODULE a_et7 IS
   AUTHORIZATION s_et7

   RECORD Savings_entry IS
      branch_num : branch_num_domain;
      acct_num : acct_num_domain;
      Balance : Balance_domain;
      cust_ssn : SSN_domain;
   END Savings_entry;

   RECORD Savings_entry2 IS
      branch_num : branch_num_domain;
      acct_num : acct_num_domain;
      Balance : Balance_domain;
      cust_ssn : SSN_domain;
      cust_ssn2 : SSN_domain;
   END Savings_entry2;

   RECORD Savings_entry3 IS
      branch_num : branch_num_domain;
      acct_num : acct_num_domain;
      Balance : Balance_domain;
   END Savings_entry3;

   RECORD Customer_record_plus IS
      branch_num : branch_num_domain;
      acct_num : acct_num_domain;
      Balance : Balance_domain;
      cust_ssn : SSN_domain;
      Cust_Name : Customer_name_domain;
      SSN : SSN_domain;
      Street : Addr_domain;
      City : City_domain;
      State : State_domain;
   END customer_record_plus;

   RECORD Customer_record_plus1 IS
      branch_num : branch_num_domain;
      acct_num : acct_num_domain;
      Balance : Balance_domain;
      cust_ssn : SSN_domain;
      Cust_Name : Customer_name_domain;
      SSN : SSN_domain;
      Street : Addr_domain;
      City : City_domain;
      State : State_domain;
      State1 : State_domain;
   END customer_record_plus1;

   RECORD Customer_record_plus2 IS
      branch_num : branch_num_domain;
      acct_num : acct_num_domain;
      Balance : Balance_domain;

   END customer_record_plus2;
cust_ssn : SSN_domain;

Cust_Name : Customer_name_domain;

SSN : SSN_domain;

Street : Addr_domain;

City : City_domain;

END customer_record_plus2;

RECORD customer_record_minus is

Cust_Name : Customer_name_domain;

Addr : Addr_Domain;

City : City_domain;

State : State_domain;

END customer_record_minus;

PROCEDURE Get_save_record
(acct_num_in : acct_num_domain) IS

SELECT * INTO savings_record : savings_entry
FROM s_et7.save
WHERE s_et7.save.acct_num = acct_num_in;

PROCEDURE Get_save_record1
(acct_num_in : acct_num_domain) IS

SELECT * -- ERR INTO savings_record : savings_entry2
FROM s_et7.save
WHERE s_et7.save.acct_num = acct_num_in;

PROCEDURE Get_save_record2
(acct_num_in : acct_num_domain) IS

SELECT * -- ERR INTO savings_record : savings_entry3
FROM s_et7.save
WHERE s_et7.save.acct_num = acct_num_in;

PROCEDURE Get_save_record3
(acct_num_in : acct_num_domain) IS

SELECT * INTO savings_record : customer_record_plus
FROM s_et7.save, cust
WHERE s_et7.save.acct_num = acct_num_in;
PROCEDURE Get_save_record4
  (acct_num_in : acct_num_domain) IS
  SELECT * -- ERR
  INTO
    savings_record : customer_record_plus1
  FROM
    s_et7.save, cust
  WHERE
    s_et7.save.acct_num =
    acct_num_in;

PROCEDURE Get_save_record5
  (acct_num_in : acct_num_domain) IS
  SELECT * -- ERR
  INTO
    savings_record : customer_record_plus2
  FROM
    s_et7.save, cust
  WHERE
    s_et7.save.acct_num =
    acct_num_in;

PROCEDURE Get_save_record6
  (acct_num_in : acct_num_domain) IS
  SELECT branch_num, acct_num, Balance, cust_ssn
  INTO
    savings_record : savings_entry
  FROM
    s_et7.save
  WHERE
    s_et7.save.acct_num =
    acct_num_in;

PROCEDURE Get_save_record7
  (acct_num_in : acct_num_domain) IS
  SELECT branch_num, acct_num, Balance, cust_ssn -- ERR
  INTO
    savings_record : savings_entry2
  FROM
    s_et7.save
  WHERE
    s_et7.save.acct_num =
    acct_num_in;

PROCEDURE Get_save_record8
  (acct_num_in : acct_num_domain) IS
  SELECT branch_num, acct_num, Balance, cust_ssn -- ERR
  INTO
    savings_record : savings_entry3
  FROM
    s_et7.save
  WHERE
    s_et7.save.acct_num =
    acct_num_in;

Cursor CGet_save_record
  (acct_num_in : acct_num_domain) FOR
SELECT * 
FROM s_et7.save 
WHERE s_et7.save.acct_num = acct_num_in;

IS
procedure CFETCH IS FETCH INTO savings_record : savings_entry;
END;

Cursor CGet_save_record1 
(acct_num_in : acct_num_domain) FOR
SELECT * -- ERR 
FROM s_et7.save 
WHERE s_et7.save.acct_num = acct_num_in;

IS
procedure CFETCH IS FETCH INTO savings_record : savings_entry2;
END;

Cursor CGet_save_record2 
(acct_num_in : acct_num_domain) FOR
SELECT * -- ERR 
FROM s_et7.save 
WHERE s_et7.save.acct_num = acct_num_in;

IS
procedure CFETCH IS FETCH INTO savings_record : savings_entry3;
END;

Cursor CGet_save_record3 
(acct_num_in : acct_num_domain) FOR
SELECT * 
FROM s_et7.save, cust 
WHERE s_et7.save.acct_num = acct_num_in;

IS
procedure CFETCH IS FETCH INTO savings_record : customer_record_plus;
END;

Cursor CGet_save_record4 
(acct_num_in : acct_num_domain) FOR
SELECT * -- ERR 
FROM s_et7.save, cust 
WHERE s_et7.save.acct_num = acct_num_in;
Appendix A - Compiler Test Suite Source Code

IS
procedure CFETCH IS FETCH INTO savings_record:
customer_record_plus1;
END;

Cursor CGet_save_record5
(acct_num_in : acct_num_domain) FOR
SELECT * -- ERR
FROM
  s_et7.save, cust
WHERE
  s_et7.save.acct_num = acct_num_in;
IS
procedure CFETCH IS FETCH INTO savings_record:
customer_record_plus2;
END;

Cursor CGet_save_record6
(acct_num_in : acct_num_domain) FOR
SELECT branch_num, acct_num, Balance, cust_ssn
FROM
  s_et7.save
WHERE
  s_et7.save.acct_num = acct_num_in;
IS
procedure CFETCH IS FETCH INTO savings_record:
savings_entry;
END;

Cursor CGet_save_record7
(acct_num_in : acct_num_domain) FOR
SELECT branch_num, acct_num, Balance, cust_ssn -- ERR
FROM
  s_et7.save
WHERE
  s_et7.save.acct_num = acct_num_in;
IS
procedure CFETCH IS FETCH INTO savings_record:
savings_entry2;
END;

Cursor CGet_save_record8
(acct_num_in : acct_num_domain) FOR
SELECT branch_num, acct_num, Balance, cust_ssn -- ERR
FROM
  s_et7.save
WHERE
  s_et7.save.acct_num = acct_num_in;
IS
procedure CFETCH IS FETCH INTO savings_record:
savings_entry3;
END;
PROCEDURE New_customer IS
    INSERT INTO
        s_et7.cust -- ERR
    FROM
        New_customer_info : customer_record_minus
    VALUES;

PROCEDURE New_customer1 IS
    INSERT INTO
        s_et7.cust -- ERR
    FROM
        New_customer_info : customer_record
    VALUES (Cust_Name, SSN, City_addr, State_addr);

PROCEDURE New_customer2 IS
    INSERT INTO -- ERR
        s_et7.cust (Cust_Name, SSN, Street_addr, City_addr,
                    State_addr)
    FROM
        New_customer_info : customer_record_minus
    VALUES (Cust_Name, NULL, Street_addr, City_addr, State_addr);

PROCEDURE New_customer3 IS
    INSERT INTO -- ERR
        s_et7.cust (Cust_Name, SSN, Street_addr, City_addr,
                    State_addr)
    FROM
        New_customer_info : customer_record_minus
    VALUES (Cust_Name, SSN, '11261 Col Pike', City_addr,
            State_addr);

PROCEDURE New_customer4 IS
    INSERT INTO -- ERR
        s_et7.cust (Cust_Name, Street_addr, City_addr, State_addr)
    VALUES (Cust_Name, City_addr, Street_addr, State_addr);

PROCEDURE New_customer5 IS
    INSERT INTO -- ERR
        s_et7.cust (Cust_Name, SSN, Street_addr, City_addr,
                    State_addr)
    VALUES (Cust_Name, SSN, Street_addr, City_addr, 'The State of
            MD');

PROCEDURE New_customer6 IS
    INSERT INTO -- ERR
        s_et7.cust (Cust_Name, SSN, Street_addr, City_addr,
                    State_addr)
    VALUES (Cust_Name, SSN, NULL, City_addr);

PROCEDURE New_customer7 IS
    INSERT INTO -- ERR
        s_et7.cust (Cust_Name, SSN, Street_addr, City_addr,
                    State_addr)
    FROM
        New_customer_info : customer_record_minus
    VALUES;
END a_et7;

SAMeDL Development Environment - Test Plan

Intermetrics, Inc.
A.3.8 tl/et8.sme

-- Check Assign conformance for set clauses

DEFINITION MODULE d_et8 IS

-- enumeration declarations

ENUMERATION Branches IS
  ( Bethesda,
    Silver_Spring,
    Gaithersburg,
    Potomac);

ENUMERATION Loan_types IS
  ( mortgage,
    auto,
    personal);

-- domain enumeration declarations

DOMAIN Loan_type_domain IS
  NEW SQL_ENUMERATION_AS_int
    (ENUMERATION => Loan_types, MAP => POS);

DOMAIN branch_num_domain IS
  NEW SQL_ENUMERATION_AS_Char
    (ENUMERATION => Branches, MAP => IMAGE);

  constant C1 : loan_type_domain is mortgage;
  constant C2 : loan_type_domain is loan_type_domain (loan_type_domain
      (auto));
  constant C3 : branch_num_domain is Bethesda;
  constant C4 : branch_num_domain is branch_num_domain
      (branch_num_domain (Silver_Spring));

END d_et8;
WITH d_et8;
USE d_et8;
SCHEMA MODULE s_et8 IS

TABLE Cust IS
    Col11 : loan_type_domain,
    Col12 : loan_type_domain,
    Col13 : branch_num_domain,
    Col14 : branch_num_domain,
    Col15 : loan_type_domain,
    Col16 : branch_num_domain
END cust;

END s_et8;
WITH d_et8;
USE d_et8;
ABSTRACT MODULE a_et8 IS
  AUTHORIZATION s_et8

  PROCEDURE Upd_Cust IS
    UPDATE s_et8.cust
    SET
      Col11 = C1,  
      Col12 = C3,  -- ERR
      Col13 = C3,
      Col14 = C4,
      Col15 = Gaithersburg, -- ERR
      Col16 = Gaithersburg;

  CURSOR Curs
    FOR SELECT * FROM s_et8.cust;
  IS
    PROCEDURE Upd (val : branch_num_domain) IS
      update s_et8.cust
      SET
      Col11 = val,  -- ERR
      Col12 = C3,  -- ERR
      Col13 = val,
      Col14 = C4,
      Col15 = Gaithersburg, -- ERR
      Col16 = Gaithersburg;
  END;

END a_et8;

A.3.9 t1/et9.sme

-- Check misc

DEFINITION MODULE d_et9 IS
  -- enumeration declarations
  --
  ENUMERATION Branches IS
    ( Bethesda,
      Silver_Spring,
      Gaithersburg,
      Potomac);

  ENUMERATION Loan_types IS
    ( mortgage,
      auto,
      personal);
  --
  -- domain enumeration declarations
  --
  DOMAIN Loan_type_domain IS
    NEW SQL_ENUMERATION_AS_int
    (ENUMERATION => Loan_types, MAP => POS);

  DOMAIN branch_num_domain IS
NEW SQL_ENUMERATION_AS_Char
(ENUMERATION => Branches, MAP => IMAGE);

record Rec is
  cl : branch_num_domain;
end Rec_Name; -- ERR: name must match

status Stat uses branch_num_domain -- ERR
  is ( 1 => mortgage, 2 => auto );
status Stat2 uses branches
  is ( 1 => mortgage, 2.0 => auto ); -- ERR
status Stat3 uses boolean
  is ( 1 => true, 2 => false );

END def_et9; -- ERR: name must match

A.3.10  t1/et10.sme

definition module d_et10 is
  -- Member Information
  domain MemName is new SQL_CHAR Not Null (Length => 30);
  domain SSN is new SQL_CHAR Not Null (Length => 9);
  domain Age is new SQL_SMALLINT (FIRST => 1, LAST => 199);

enumeration SexEnum is (F, M);
  domain Sex is new SQL_ENUMERATION_AS_INT
      (Enumeration => SexEnum, map => pos);

domain Phone is new SQL_CHAR (Length => 8);
  domain Street is new SQL_CHAR (Length => 30);
  domain City is new SQL_CHAR (Length => 15);

domain County is new SQL_CHAR Not Null (Length => 2);

domain Club_Number is new SQL_SMALLINT Not Null;
  domain Sum_Domain is new SQL_SMALLINT Not Null;
  domain Count_Domain is new SQL_INT;

end d_et10;

with d_et10; use d_et10;
schema module s_recdb is
  table Members is
    MemberName not null : MemName,
    MemberSSN not null : SSN,
    ClubNumber not null : Club_Number,
    MemberAge : Age,
    MemberSex : Sex,
    MemberPhone : Phone,
    MemberStreet : Street,
    MemberCity : City,
    MemberCnty not null : County
end Members;

end s_recdb;

with d_et10; use d_et10;
abstract module a_et10 is
authorization s_recdb

record MemberRec is
  R_MemberName : MemName;
  R_Sum : Sum_Domain;
  R_Count : Count_Domain;
end;

procedure P_Del (ReqjMemberSSN : SSN) is
  delete from MemberRec; -- ERR

procedure P_Upd (ReqMemberSSN : SSN) is
  update MemberRec -- ERR
    set MemberName = ReqMemberRec;

procedure P_Ins is
  insert into MemberRec -- ERR
    VALUES;
end a_et10;

A.3.11  t1/et11.sme
-- Check conformance on insert subquery statements

DEFINITION MODULE d_et11 IS
  -- enumeration declarations
  --
  ENUMERATION Branches IS
    ( Bethesda,
      Silver_Spring,
      Gaithersburg,
      Potomac);

  ENUMERATION Loan_types IS
    ( mortgage,
      auto,
      personal);

  -- domain character declarations
  --
  DOMAIN Customer_name_domain IS
    NEW SQL_CHAR(length => 15);
  DOMAIN Addr_domain IS
    NEW SQL_CHAR(length => 15);
  DOMAIN City_domain IS
    NEW SQL_CHAR(length => 15);
  DOMAIN State_domain IS
    NEW SQL_CHAR(length => 2);
-- domain integer declarations

-- DOMAIN SSN_domain IS
   NEW SQL_INT NOT NULL ( FIRST => 0, LAST => 999999999);
DOMAIN acct_num_domain IS
   NEW SQL_SMALLINT NOT NULL ( FIRST => 0, LAST => 9999);

-- domain real declarations

-- DOMAIN Balance_domain IS
   NEW SQL_REAL;
DOMAIN Interest_rate_domain IS
   NEW SQL_REAL( FIRST => 0.0, LAST => 1.0);
DOMAIN Loan_payment_domain IS
   NEW SQL_REAL;
DOMAIN Branch_assets_domain IS
   NEW SQL_REAL;
Appendix A - Compiler Test Suite Source Code

-- domain enumeration declarations

DOMAIN Loan_type_domain IS  
  NEW SQL_ENUMERATION_AS_int  
  (ENUMERATION => Loan_types, MAP => POS);  

DOMAIN branch_num_domain IS  
  NEW SQL_ENUMERATION_AS_INT  
  (ENUMERATION => Branches, MAP => POS);  

-- record definitions

RECORD Customer_record IS  
  Cust_Name : Customer_name_domain;  
  SSN : SSN_domain;  
  Street : Addr_domain;  
  City : City_domain;  
  State : State_domain;  
END customer_record;

RECORD Savings_entry IS  
  branch_num : branch_num_domain;  
  acct_num : acct_num_domain;  
  Balance : Balance_domain;  
  cust_ssn : SSN_domain;  
END Savings_entry;

RECORD Chequeing_entry IS  
  branch_num : branch_num_domain;  
  acct_num : acct_num_domain;  
  Balance : Balance_domain;  
  cust_ssn : SSN_domain;  
END Chequeing_entry;

RECORD loan_entry IS  
  branch_num : branch_num_domain;  
  acct_num : acct_num_domain;  
  Balance : Balance_domain;  
  Loan_type : Loan_type_domain;  
  cust_ssn : SSN_domain;  
END loan_entry;

RECORD Branch_entry IS  
  branch_num : branch_num_domain;  
  Assets : Branch_assets_domain;  
END Branch_entry;

END d_et11;
WITH d_etll;
USE d_etll;
SCHEMA MODULE s_etll IS

-- Basic customer information

-- TABLE Cust IS
   Cust_Name : Customer_name_domain,
   SSN not null : SSN_domain,
   Street_addr : Addr_domain,
   City_addr : City_domain,
   State_addr : State_domain
END Cust;

-- Checking account

-- TABLE cheque IS
   branch_num : branch_num_domain,
   acct_num not null : acct_num_domain,
   Balance : Balance_domain,
   cust_ssn not null : SSN_domain
END cheque;

-- Savings account

-- TABLE Save IS
   branch_num : branch_num_domain,
   acctnum not null : acctnum_domain,
   Balance : Balance_domain,
   cust_ssn not null : SSN_domain
END Save;

-- loan account

-- TABLE loan IS
   branch_num : branch_num_domain,
   acct_num not null : acct_num_domain,
   Balance : Balance_domain,
   Loan_type : loan_type_domain,
   cust_ssn not null : SSN_domain
END loan;

-- Branch information

-- TABLE Branch IS
   num : branch_num_domain,
   Assets : Branch_assets_domain
END Branch;

END s_etll;
WITH d_etll;
USE d_etll;
ABSTRACT MODULE a_etll IS
  -- insert statement (query)
  --
  PROCEDURE move_loan_to_save
  (account_num_in : acct_num_domain)
  IS
  INSERT INTO
    s_etll.save
  SELECT *
  FROM
    s_etll.loan -- ERR
  WHERE
    s_etll.loan.acct_num >= account_num_in;
  PROCEDURE move_loan_to_save2
  (account_num_in : acct_num_domain)
  IS
  INSERT INTO
    s_etll.save
  SELECT branch_num, acct_num, Balance, cust_ssn
  FROM
    s_etll.loan
  WHERE
    s_etll.loan.acct_num >= account_num_in;
  PROCEDURE move_loan_to_save3
  (account_num_in : acct_num_domain)
  IS
  INSERT INTO
    s_etll.save
  SELECT branch_num, acct_num, cust_ssn, cust_ssn -- ERR
  FROM
    s_etll.loan
  WHERE
    s_etll.loan.acct_num >= account_num_in;
END a_etll;

A.3.12  t1/t2.sme

definition module d_et12 is
  -- Member Information
  domain MemName is new SQL_CHAR Not Null (Length => 30);
  domain SSN is new SQL_CHAR Not Null (Length => 9);
  domain Age is new SQL_SMALLINT (FIRST => 1, LAST => 199);
  enumeration SexEnum is (F, M);
  domain Sex is new SQL_ENUMERATION_AS_INT (Enumeration => SexEnum, MAP => POS);
  domain Phone is new SQL_CHAR (Length => 8);
  domain Street is new SQL_CHAR (Length => 30);
  domain City is new SQL_CHAR (Length => 15);
domain County is new SQL_CHAR Not Null (Length => 2);

domain Club_Number is new SQL_SMALLINT Not Null;

constant C_Name : MemName is '123456789012345678901234567890';
constant C_SSN : SSN is '123456789';
constant C_Club_Number : Club_Number is 10;
constant C_Age : Age is 39;
constant C_Sex : Sex is F;
constant C_Phone : Phone is '12345678';
constant C_Street : Street is '123456789012345678901234567890';
constant C_City : City is '123456789012345';
constant C_County : County is 'MO';

end d_etl2;

with d_etl2; use d_etl2;
schema module RecDB is
  table Members is
    MemberName not null : MemName,
    MemberSSN not null : SSN,
    ClubNumber not null : Club_Number,
    MemberAge : Age,
    MemberSex : Sex,
    MemberPhone : Phone,
    MemberStreet : Street,
    MemberCity : City,
    MemberCnty not null : County
  end Members;
end RecDB;

with d_etl2; use d_etl2;
abstract module a_etl2 is
  authorization RecDB

  record MemberRec named Named(MemberRec is
    -- record MemberRec is
    R_MemberName : MemName;
    R_MemberSSN : SSN;
    R_ClubNumber : Club_Number;
    R_MemberAge : Age;
    R_MemberSex : Sex;
    R_MemberPhone : Phone;
    R_MemberStreet : Street;
    R_MemberCity : City not null;
    R_MemberCnty : County;
  end;

  cursor MemberSelect2 (ReqMemberSSN named ReqMemberSSN : SSN) for
    (select
      MemberName named NS_MemberName,
      MemberSSN,
      ClubNumber,
      MemberAge,
      MemberSex
    )
end;
MemberPhone Not Null,
MemberStreet named NS_MemberStreet Not Null,
MemberCity,
MemberCnty
from RecDB.Members
where RecDB.Members.MemberSSN = Req_MemberSSN
UNION
select
  MemberName named NS_MemberName,
  MemberSSN,
  ClubNumber,
  MemberAge,
  MemberSex,
  MemberPhone Not Null,
MemberStreet named NS_MemberStreet Not Null,
MemberCity,
MemberCnty
from RecDB.Members
where RecDB.Members.MemberSSN = Req_MemberSSN
UNION
(select
MemberSSN,
ClubNumber,
MemberAge,
MemberSex,
MemberPhone Not Null,
MemberStreet named NS_MemberStreet Not Null,
MemberCity,
MemberCnty
from RecDB.Members
where RecDB.Members.MemberSSN = Req_MemberSSN)
UNION
(select
MemberSSN,
ClubNumber,
MemberAge,
MemberSex,
MemberPhone Not Null,
MemberStreet named NS_MemberStreet Not Null,
MemberCity,
MemberCnty
from RecDB.Members
where RecDB.Members.MemberSSN = Req_MemberSSN)
UNION
select
  MemberName named NS_MemberName,
  MemberSSN,
  ClubNumber,
  MemberAge,
  MemberSex,
  MemberPhone Not Null,
MemberStreet named NS_MemberStreet Not Null,
MemberCity,
MemberCnty
from RecDB.Members
where RecDB.Members.MemberSSN = Req_MemberSSN);
cursor MemberSelect3 (Req_MemberSSN named Req_MemberSSN : SSN) for
(select
  MemberName named NS_MemberName,
  MemberSSN,
  ...
ClubNumber,  
MemberAge,  
MemberSex,  
MemberPhone,  
MemberStreet named NS_MemberStreet Not Null,  
MemberCity,  
MemberCnty  
from RecDB.Members  
where RecDB.Members.MemberSSN = Req_MemberSSN  
UNION  
select  
MemberName named NS_MemberName,  
MemberSSN,  
ClubNumber,  
MemberAge,  
MemberSex,  
MemberPhone Not Null,  
MemberStreet named NS_MemberStreet Not Null,  
MemberCity,  
MemberCnty  
from RecDB.Members  
where RecDB.Members.MemberSSN = Req_MemberSSN)  
UNION  
(select  
MemberName named NS_MemberName,  
MemberSSN,  
ClubNumber,  
MemberAge,  
MemberSex,  
MemberPhone Not Null,  
MemberStreet named NS_MemberStreet Not Null,  
MemberCity,  
MemberCnty  
from RecDB.Members  
where RecDB.Members.MemberSSN = Req_MemberSSN  
UNION  
select  
MemberName named NS_MemberName,  
MemberSSN,  
ClubNumber,  
MemberAge,  
MemberSex,  
MemberPhone Not Null,  
MemberStreet named NS_MemberStreet Not Null,  
MemberCity,  
MemberCnty  
from RecDB.Members  
where RecDB.Members.MemberSSN = Req_MemberSSN)
where RecDB.Members.MemberSSN = Req_.MemberSSN);
end aetl2;

A.3.13 t3/el.sme

definition module t_1 is
  -- Member Information
  domain MemberName is new SQL_CHAR Not Null (Length => 30);
domain SSN is new SQL_CHAR Not Null (Length => 9);
domain Age is new SQL_SMALLINT (FIRST => 1, LAST => 199);

  enumeration SexEnum is (F, M);
domain Sex is new SQL_ENUMERATION_AS_INT (Enumeration => SexEnum, MAP => Pos);

  domain Phone is new SQL_CHAR (Length => 8);
domain Street is new SQL_CHAR (Length => 30);
domain City is new SQL_CHAR (Length => 15);

  domain County is new SQL_CHAR Not Null (Length => 2);
domain Club_Number is new SQL_SMALLINT Not Null;

  exception Record_Not_Found;

  enumeration FailType is (Not_Logged_In, SQL_OK, SQL_Fail);

  status fetch_map named is_found uses FailType is
  (-999 .. -300 => SQL_Fail,
   -299, -298 => Not_Logged_In,
   0 => SQL_OK,
   100 => raise samplemod.record_not_found);
end t_1;

with SampleMod; use SampleMod;
schema module RecDB is
  table Members is
    MemberName not null : MemberName,
    MemberSSN not null : SSN,
    ClubNumber not null : Club_Number,
    MemberAge : Age,
    MemberSex : Sex,
    MemberPhone : Phone,
    MemberStreet : Street,
    MemberCity : City,
    MemberCnty not null : County
  end Members;

end RecDB;

with SampleMod; use SampleMod;
abstract module RecDML is

Intermetrics, Inc.
authorization RecDB

record MemberRec is
    MemberName : MemberName;
    MemberSSN : SSN;
    ClubNumber : Club_Number;
    MemberAge : Age;
    MemberSex : Sex;
    MemberPhone : Phone;
    MemberStreet : Street;
    MemberCity : City;
    MemberCnty : County;
end;

cursor MemberSelect (Req_MemberSSN : SSN) for
    select tl.MemberSSN, t2.Dummy
    from RecDB.Members as tl, RecDB.Members as t2
    where tl.MemberSSN = t2.MemberSSN;
end RecDML;

A.3.14 t3/e2.sme

definition module t_2 is
    -- Member Information
    domain MemberName is new SQL_CHAR Not Null (Length => 30);
    domain SSN is new SQL_CHAR Not Null (Length => 9);
    domain Age is new SQL_SMALLINT (FIRST => 1, LAST => 199);
    enumeration SexEnum is (F, M);
    domain Sex is new SQL_ENUMERATION_AS_INT
        (Enumeration => SexEnum, Map => POS);
    domain Phone is new SQL_CHAR (Length => 8);
    domain Street is new SQL_CHAR (Length => 30);
    domain City is new SQL_CHAR (Length => 15);
    domain County is new SQL_CHAR Not Null (Length => 2);
    domain Club_Number is new SQL_SMALLINT Not Null;
    exception Record_Not_Found;
    enumeration FailType is (Not_Logged_In, SQL_OK, SQL_Fail);
    status fetch_map named is_found uses Failtype is
        ( -999 .. -300 => SQL_Fail,
          -299, -298 => Not_Logged_In,
          0 => SQL_OK,
          100 => raise samplemod.record_not_found);
end t_2;

with t_2; use t_2;
schema module RecDB is
    table Members is
        MemberName not null : MemberName,
Appendix A - Compiler Test Suite Source Code

MemberSSN not null : SSN,
ClubNumber not null : Club_Number,
MemberAge : Age,
MemberSex : Sex,
MemberPhone : Phone,
MemberStreet : Street,
MemberCity : City,
MemberCnty not null : County
end Members;

table Members2 is
MemberName not null : MemberName,
MemberSSN not null : SSN,
ClubNumber not null : Club_Number
end Members2;

definition module t_3 is
--- Member Information
domain MemberName is new SQL_CHAR Not Null (Length => 30);
domain SSN is new SQL_CHAR Not Null (Length => 9);
domain Age is new SQL_SMALLINT ( FIRST => 1, LAST => 199);

enumeration SexEnum is (F, M);
domain Sex is new SQL_ENUMERATION_AS_INT (Enumeration => SexEnum, Map => POS);
domain Phone is new SQL_CHAR (Length => 8);
domain Street is new SQL_CHAR (Length => 30);

Intermetrics, Inc.
domain City is new SQL_CHAR (Length => 15);
domain County is new SQL_CHAR Not Null (Length => 2);
domain Club_Number is new SQL_SMALLINT Not Null;

exception Record_Not_Found;

enumeration FailType is (NotLogged_In, SQLOk, SQLFail);

status fetch_map named is_found uses Failtype is
  (-999 .. -300 => SQLFail,
   -299, -298 => NotLogged_In,
   0 => SQLOk,
   100 => raise samplemod.record_not_found);
end t_3;

with t_3; use t_3;

Jschema module RecDB is
  table Members is
    MemberName not null : MemberName,
    MemberSSN not null : SSN,
    ClubNumber not null : Club_Number,
    MemberAge : Age,
    MemberSex : Sex,
    MemberPhone : Phone,
    MemberStreet : Street,
    MemberCity : City,
    MemberCnty not null : County
  end Members;

  table Members2 is
    MemberName not null : MemberName,
    MemberSSN not null : SSN,
    ClubNumber not null : Club_Number
  end Members2;
end RecDB;

with t_3; use t_3;
abstract module RecDML is
  authorization RecDB

  record MemberRec is
    MemberName : MemberName;
    MemberSSN : SSN;
    ClubNumber : Club_Number;
    MemberAge : Age;
    MemberSex : Sex;
    MemberPhone : Phone;
    MemberStreet : Street;
    MemberCity : City;
    MemberCnty : County;
  end;
cursor MemberSelect (Req_MemberSSN : SSN) for
select MemberSSN, Recdb.Members.Membername
from RecDB.Members2 as t1, Recdb.members as t2
where t1.MemberSSN = (select MemberSSN
from t2
where Recdb.members.membername = 'John');

...........

dend RecDML;