INTERNATIONAL STANDARD

ISO/IEC 12227

First edition
1995-02-15

Information technology — Programming languages — SQL/Ada Module Description Language (SAMeDL)

Technologies de l'information — Langages de programmation — Langage de description de modules SQL/Ada (SAMeDL)

Reference number
ISO/IEC 12227:1995(E)

Contents

	Page
1 Scope	**1**
2 Normative references	**1**
3 Notations and Structures	**2**
3.1 Syntax Notation	2
3.2 Semantic Notation	2
3.3 Structure	3
3.4 Examples, Notes and Index Entries	3
4 Design Goals and Language Summary	**3**
4.1 Design Goals	3
4.2 Language Summary	4
4.2.1 Overview	4
4.2.2 Compilation Units	4
4.2.3 Modules	5
4.2.4 Procedures and Cursors	5
4.2.5 Domain and Base Domain Declarations	5
4.2.6 Other Declarations	6
4.2.7 Value Expressions and Typing	6
4.2.8 Standard Post Processing	6
4.2.9 Extensions	6
4.2.10 Default Values in Grammar	6
4.3 Entry Level SQL	7
5 Lexical Elements	**7**
5.1 Character Set	7
5.2 Lexical Elements, Separators, and Delimiters	7
5.3 Identifiers	8
5.4 Literals and Data Classes	9
5.5 Comments	10
5.6 Reserved Words	10

6 Common Elements 10
6.1 Compilation Units 11
6.2 Context Clause 11
6.3 Table Names and the From Clause 12
6.4 References 12
6.5 Assignment Contexts and Conformance of an Expression to a Domain 17
6.6 Standard Post Processing 18
6.7 Extensions 18

7 Data Description Language and Data Semantics 19
7.1 Definitional Modules 19
7.1.1 Base Domain Declarations 20
7.1.1.1 Base Domain Parameters 20
7.1.1.2 Base Domain Patterns 21
7.1.1.3 Base Domain Options 22
7.1.2 The SAME Standard Base Domains 23
7.1.3 Domain and Subdomain Declarations 24
7.1.4 Constant Declarations 28
7.1.5 Record Declarations 29
7.1.6 Enumeration Declarations 31
7.1.7 Exception Declarations 32
7.1.8 Status Map Declarations 32
7.2 Schema Modules 34
7.2.1 Table Definitions 34
7.2.2 View Definitions 36
7.3 Data Conversions 37

8 Abstract Module Description Language 38
8.1 Abstract Modules 38
8.2 Procedures 39
8.3 Statements 43
8.4 Cursor Declarations 45
8.5 Cursor Procedures 49
8.6 Input Parameter Lists 52
8.7 Select Parameter Lists 54
8.8 Value Lists and Column Lists 56
8.9 Into_Clause and Insert_From_Clause 58
8.10 Value Expressions 60
8.11 Search Conditions 66
8.12 Subqueries 68
8.13 Status Clauses 68

9 Conformance 69
9.1 Introduction 69
9.2 Claims of Conformance 69
9.2.1 Introduction 69
9.2.2 Conformance via mapping. 69
9.2.2.1 Conformance via SQL module. 69
9.2.2.2 Conformance via embedded SQL Syntax. 69

9.2.3 Conformance via effects. 70
9.2.4 Multiple claims of conformance. 70
9.3 Extensions 70

Annex A. SAMeDL_Standard 71

Annex B. SAMeDL_System 84

Annex C. Standard Support Operations and Specifications 85
C.1 Standard Base Domain Operations 85
C.1.1 All Domains 85
C.1.2 Numeric Domains 86
C.1.3 Int and Smallint Domains 86
C.1.4 Character Domains 87
C.1.5 Enumeration Domains 87
C.1.6 Boolean Functions 88
C.1.7 Operations Available to the Application 88
C.2 Standard Support Package Specifications 90
C.2.1 SQL_Standard 90
C.2.2 SQL_Boolean_Pkg 91
C.2.3 SQL_Int_Pkg 92
C.2.4 SQL_Smallint_Pkg 95
C.2.5 SQL_Real_Pkg 98
C.2.6 SQL_Double_Precision_Pkg 101
C.2.7 SQL_Char_Pkg 104
C.2.8 SQL_Enumeration_Pkg 107

Annex D. Transform Chart 109

Annex E. Glossary 114

Annex F. References 117

Index 118

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work.

In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1. Draft International Standards adopted by the joint technical committee are circulated to national bodies for voting. Publication as an International Standard requires approval by at least 75 % of the national bodies casting a vote.

International Standard ISO/IEC 12227 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 22, *Programming languages, their environments and system software interfaces*.

Annexes A to C form an integral part of this International Standard. Annexes D to F are for information only.

This page intentionally left blank

Information technology - Programming languages - SQL/Ada Module Description Language (SAMeDL)

1 Scope

This International Standard specifies the syntax and semantics of a database programming language, the SQL Ada Module Description Language, SAMeDL. Texts written in the SAMeDL describe database interactions which are to be performed by database management systems (DBMS) implementing Database Language SQL. The interactions so described and so implemented are to be performed on behalf of application programs written in Programming Language Ada.

The SAMeDL is not a Programming Language; it may be used solely to specify application program database interactions and solely when those interactions are to occur between an Ada application program and an SQL DBMS.

The SAMeDL is defined with respect to Entry Level SQL. Therefore, all inclusions by reference of text from ISO/IEC 9075:1992 include all applicable Leveling Rules for Entry Level SQL.

This International Standard does not define the Programming Language Ada nor the Database Language SQL. Therefore, ISO 8652:1987 takes precedence in all matters dealing with the syntax and semantics of any Ada construct contained, referred or described within this International Standard; similarly, ISO/IEC 9075:1992 takes precedence in all matters dealing with the syntax and semantics of any SQL construct contained, referred or described within this International Standard.

Note: The SAMeDL is an example of an Abstract Modular Interface. A reference model for programming language interfaces to database management systems, which includes a description of Abstract Modular interfaces, can be found in reference [2].

2 Normative references

The following International Standards contain provisions which, through reference in this text, constitute provisions of this International Standard. At the time of publication, the editions indicated were valid. All International Standards are subject to revision, and parties to agreements based on this International Standard are encouraged to investigate the possibility of applying the most recent editions of the International Standards indicated below. Members of IEC and ISO maintain registers of currently valid International Standards.

ISO/IEC 9075:1992, *Information technology -- Database languages -- SQL.*

ISO/IEC 8652:1995, *Information technology -- Programming languages -- Ada.*

3 Notations and Structures

3.1 Syntax Notation

The context-free syntax of the language is described using a simple variant of the variant of BNF used in the description of Ada, (1.5 in ISO 8652:1987). The variation is:

-- Underscores are preserved when using the name of a syntactic category outside of a syntax rule (1.5(6) of ISO 8652:1987).

-- The italicized prefixes *Ada* and *SQL*, when appearing in the names of syntactic categories, indicate that an Ada or SQL syntactic category has been incorporated into this document. For example, the category *Ada*_identifier is identical to the category identifier as described in 2.3 of ISO 8652:1987; whereas the category *SQL*_identifier is identical to the category identifier as described in 5.4 of ISO/IEC 9075:1992.

-- Numerical suffixes attached to the names of syntactic categories are used to distinguish appearances of the category within a rule or set of rules. An example of this usage is given below.

3.2 Semantic Notation

The meaning of a SAMeDL compilation unit (except where specified as implementation-defined) is given by:

-- An Ada compilation unit, conforming to ISO 8652:1987.

-- A module conforming to clause 12 of ISO/IEC 9075:1992.

-- Interface rules, concerning the relationship between the SQL and Ada texts.

The semantics of SAMeDL constructs are given in part by collections of string transformers that produce Ada and SQL texts from SAMeDL input.

Note: A quick reference to these transformers appears in Annex D.

The effects of these transformers are described through the use of sample input strings. Those strings are written in a variant of the syntax notation. For example, the syntax of an input_parameter (see 8.6) is given by:

identifier_1 `[named_phrase] : domain_reference [`**not null**`]`

A representative input parameter declaration is given by

Id_1 `[`**named** *Id_2*`] : Id_3 [`**not null**`]`

It is then possible to discuss the four variants of input parameters (the variants described by the presence or absence of optional phrases) in a single piece of text.

3.3 Structure

The remainder of this International Standard is structured in the following way. Clause 4 contains a descriptive overview of the goals and concepts of the SAMeDL. Clause 5 defines the lexical structure of the SAMeDL including the rules for identifier and literal formation and the list of reserved words. Clauses 6, 7 and 8 define the syntax and semantics of the SAMeDL. Each subclause of those clauses adheres to the following general format.

-- The purpose of the item being defined by the subclause is introduced.

-- The syntax of the item being defined is given in the notation described earlier.

-- Abstract (non-context free) syntactical rules governing formation of instances of the item being defined are given, if applicable.

-- The semantics of the item being defined are given. These semantics are given under the headings **Ada Semantics, SQL Semantics** and **Interface Semantics**, as appropriate.

3.4 Examples, Notes and Index Entries

Many of the subclauses of this International Standard are illustrated with examples. These examples are introduced by the words: "*Note:* **Examples**" on a line by themselves and are terminated by the words "**End Examples**," likewise appearing on a line by themselves.

Note: **Examples**

This is an example of an example.

End Examples

Other notes also appear in this International Standard, introduced by the word *Note*. Both kinds of note are informative only and do not form part of the definition of the SAMeDL.

Items in the grammar that contain underscores are represented in the Index by a corresponding entry without underscores. For example, the "Ada identifier" entry in the index contains the page numbers of occurrences of both "Ada_identifier" and "Ada identifier".

4 Design Goals and Language Summary

4.1 Design Goals

The SQL Ada Module Description Language (SAMeDL) is a Database Programming Language designed to automate the construction of software conformant to the SQL Ada Module Extensions (SAME) application architecture. This architecture is described in the document, *Guidelines for the Use of the SAME* [1].

The SAME is a *modular* architecture. It uses the concept of a Module as defined in 4.16 and 12 of ISO/IEC 9075:1992. As a consequence, a SAME-conformant Ada application does not contain embedded SQL statements and is not an embedded SQL Ada program as defined in 19.3 of ISO/IEC 9075:1992. Such a SAME-conformant application treats SQL in the manner in which Ada treats all other languages: it imports complete functional modules, not language fragments.

Modular architectures treat the interaction of the application program and the database as a design object. This results in a further isolation of the application program from details of the database design and implementation and improves the potential for increased specialization of software development staff.

Ada and SQL are vastly different languages: Ada is a Programming Language designed to express algorithms, while SQL is a Database Language designed to describe desired results. Text containing both Ada and SQL is therefore confusing and difficult to maintain. SAMeDL is a Database Programming Language designed to support the goals and exploit the capabilities of Ada with a language whose syntax and semantics is based firmly in SQL. Beyond modularity, the SAMeDL provides the application programmer the following services:

-- An abstract treatment of null values. Using Ada typing facilities, a safe treatment of missing information based on SQL is introduced into Ada database programming. The treatment is safe in that it prevents an application from mistaking missing information (null values) for present information (non-null values).

-- Robust status code processing. SAMeDL's Standard Post Processing provides a structured mechanism for the processing of SQL status parameters.

-- Strong typing. SAMeDL's typing rules are based on the strong typing of Ada, not the permissive typing of SQL.

-- Extensibility. The SAMeDL supports a class of user extensions. Further, it controls, but does not restrict, implementation defined extensions.

4.2 Language Summary

4.2.1 Overview

The SAMeDL is designed to facilitate the construction of Ada database applications that conform to the SAME architecture as described in [1]. The SAME method involves the use of an abstract interface, an abstract module, a concrete interface, and a concrete module. The abstract interface is a set of Ada package specifications containing the type and procedure declarations to be used by the Ada application program. The abstract module is a set of bodies for the abstract interface. These bodies are responsible for invoking the routines of the concrete interface, and converting between the Ada and the SQL data and error representations. The concrete interface is a set of Ada specifications that define the SQL procedures needed by the abstract module. The concrete module is a set of SQL procedures that implement the concrete interface.

Within this International Standard, the concrete module of [1] is called an SQL module and its contents are given under the headings **SQL Semantics** within the clauses of this specification. The abstract modules of [1] are given under the heading **Ada Semantics** within the clauses of this specification.

4.2.2 Compilation Units

A *compilation unit* consists of one or more modules. A module may be either a definitional module containing shared definitions, a schema module containing table, view, and privilege definitions, or an abstract module containing local definitions and procedure and cursor declarations.

4.2.3 Modules

A *definitional module* contains the definitions of base domains, domains, constants, records, enumerations, exceptions, and status maps. Definitions in definitional modules may be seen by other modules.

A *schema module* contains the definitions of tables, views, and privileges.

An *abstract module* defines (a portion of) an application's interface to the database: it defines SQL services needed by an Ada application program. An abstract module may contain procedure declarations, cursor declarations, and definitions such as those that may appear in a definitional module. Definitions in an abstract module, however, may not be seen by other modules.

4.2.4 Procedures and Cursors

A *procedure* declaration defines a basic database operation. The declaration defines an Ada procedure declaration and a corresponding SQL procedure. A SAMeDL procedure consists of a single statement along with an optional input parameter list and an optional status clause. The input parameter list provides the mechanism for passing information to the database at runtime. A statement in a SAMeDL procedure may be a commit statement, rollback statement, insert statement query, insert statement values, update statement, select statement or an implementation-defined extended statement. The semantics of a SAMeDL statement directly parallel that of its corresponding SQL statement.

SAMeDL *cursor* declarations directly parallel SQL cursor declarations. In contrast to the language of ISO/IEC 9075:1992, the procedures that operate on cursors, procedures containing either an open, fetch, close, update positioned or delete positioned statement, are packaged with the declaration of the cursor upon which they operate, thereby improving readability. Further, if no procedure containing an open, fetch or close statement is explicitly given in a cursor declaration, the language provides such procedures implicitly, thereby improving writeability (ease of use).

4.2.5 Domain and Base Domain Declarations

Objects in the language have an associated *domain*, which characterizes the set of values and applicable operations for that object. In this sense, a domain is similar to an Ada type.

A *base domain* is a template for defining domains. A base domain declaration consists of a set of parameters, a set of patterns and a set of options. The parameters are used to supply information needed to declare a domain or subdomain derived from the base domain. Patterns contain templates for the generation of Ada code to support the domain in Ada applications. This code generally contains type declarations and package instantiations. Options contain information needed by the compiler. Parameters may be used in the patterns and options and their values may be referenced in other statements.

Base domains are classified according to their associated data class. A data class is either integer, fixed, float, enumeration, or character. A numeric base domain has a data class of either integer, fixed, or float. An enumeration base domain has a data class of enumeration, and defines both an ordered set of distinct enumeration literals and a bijection between the enumeration literals and their associated database values. A character base domain has a data class of character.

4.2.6 Other Declarations

Certain SAMeDL declarations are provided as a convenience for the user. For example, *constant* declarations name and associate a domain with a static expression. *Record* declarations allow distinct procedures to share types. An *exception* declaration defines an Ada exception declaration with the same name.

4.2.7 Value Expressions and Typing

Value expressions are formed and evaluated according to the rules of SQL, with the exception that the strong typing rules are based on those of Ada. In the typing rules of the SAMeDL, the domain acts as an Ada type in a system without user defined operations. Strong typing necessitates the introduction of domain conversions. These conversions are modeled after Ada type conversions; the operational semantics of the SAMeDL domain conversion is the null operation or identity mapping. The language rules specify that an informational message be displayed under circumstances in which this departure from the Ada model has visible effect.

4.2.8 Standard Post Processing

Standard post processing is performed after the execution of an SQL procedure but before control is returned to the calling application procedure. The *status clause* from a SAMeDL procedure declaration attaches a *status mapping* to the application procedure. That status mapping is used to process SQL status data in a uniform way for all procedures and to present SQL status codes to the application in an application-defined manner, either as a value of an enumerated type, or as a user defined exception. SQL status codes not specified by the status map result in a call to a standard database error processing procedure and the raising of the predefined SAMeDL exception, SQL_Database_Error. This prevents a database error from being ignored by the application.

4.2.9 Extensions

The data semantics of the SAMeDL may be extended without modification to the language by the addition of user-defined base domains. For example, a user-defined base domain of DATE may be included without modification to the SAMeDL.

DBMS specific (i.e., non-standard) operations and features that require compiler modification (e.g., dynamic SQL) may also be included into the SAMeDL. Such additions to the SAMeDL are referred to as extensions. Schema elements, table elements, statements, query expressions, query specifications, and cursor statements may be extended. The modules, tables, views, cursors, and procedures that contain these extensions are marked (with the keyword **extended**) to indicate that they go outside the standard.

4.2.10 Default Values in Grammar

Obvious but overridable defaults are provided in the grammar. For example, open, close, and fetch statements are essential for a cursor, but their form may be deduced from the cursor declaration. The SAMeDL will therefore supply the needed open, close, and fetch procedure declarations if they are not supplied by the user.

4.3 Entry Level SQL

Within the text of this specification, the name SQL references the language known as Entry Level SQL in ISO/IEC 9075:1992. Features and capabilities of ISO/IEC 9075:1992 that do not lie within Entry Level SQL may be implemented via the extension facility defined in this International Standard. See 6.7 of this specification.

5 Lexical Elements

The text of a compilation is a sequence of lexical elements, each composed of characters from the basic character set. The rules of composition are given in this chapter.

5.1 Character Set

The only characters allowed in the text of a compilation are the basic characters and the characters that make up character literals (described in 5.4 of this specification). Each character in the basic character set is represented by a graphical symbol.

```
basic_character ::=
    upper_case_letter | lower_case_letter | digit |
    special_character | space_character
```


The characters included in each of the above categories of the basic characters are defined as follows:

1. upper_case_letter
   A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

2. lower_case_letter
   a b c d e f g h i j k l m n o p q r s t u v w x y z

3. digit
   0 1 2 3 4 5 6 7 8 9

4. special_character
   ' ( ) * + , - . / : ; < = > _ |

5. space_character

5.2 Lexical Elements, Separators, and Delimiters

The text of each compilation is a sequence of separate lexical elements. Each lexical element is either an identifier (which may be a reserved word), a literal, a comment, or a delimiter. The effect of a compilation depends only on the particular sequences of lexical elements excluding the comments, if any, as described in this chapter. Identifiers, literals, and comments are discussed in the following clauses. The remainder of this clause discusses delimiters and separators.

An explicit separator is required to separate adjacent lexical elements when, without separation, interpretation as a single lexical element is possible. A separator is any of a space character, a format effector, or the end of a

line. A space character is a separator except within a comment or a character literal. Format effectors other than horizontal tabulation are always separators. Horizontal tabulation is a separator except within a comment.

The end of a line is always a separator. The language does not define what causes the end of a line.

One or more separators are allowed between any two adjacent lexical elements, before the first lexical element of each compilation, or after the last lexical element of each compilation. At least one separator is required between an identifier or a numeric literal and an adjacent identifier or numeric literal.

A delimiter is one of the following special characters

```
( ) * + , - . / : ; < = > |
```

or one of the following compound delimiters each composed of two adjacent special characters

```
=> .. := <> >= <=
```

Each of the special characters listed for single character delimiters is a single delimiter except if that character is used as a character of a compound delimiter, a comment, or a literal.

Each lexical element shall fit on one line, since the end of a line is a separator. The single quote and underscore characters, as well as two adjacent hyphens, are not delimiters, but may form part of other lexical elements.

5.3 Identifiers


```
identifier ::= SQL_actual_identifier
```

The length restrictions that apply to *SQL*_identifiers (see 5.2, syntax rules 8, 9 and 5.5 syntax rule 3 of ISO/IEC 9075:1992) *do not apply* to SAMeDL identifiers. Whenever two identifiers are deemed equivalent, it is in the sense of the rules of SQL (see 5.2, syntax rules 10 through 14 of ISO/IEC 9075:1992).

Note. An *SQL*_actual_identifier is either a delimited_identifier or a regular_identifier. The form of an SQL regular_identifier is essentially the same as the Ada identifier, except that a regular_identifier may end in an underscore. A delimited_identifier is *any* character string within a pair of doublequote characters ("). Delimited identifiers provide a means of using tokens that would otherwise be reserved words (see 5.6 of this specification) as identifiers. Thus **fetch** is a reserved word, but the construct

```
procedure "fetch" is fetch;
```

defines a procedure named "fetch" that contains a fetch statement.

Identifier equivalence in SQL is similar to Ada identifier equivalence for regular identifiers. Equivalence for delimited identifiers is case sensitive. Equivalence of regular identifiers with delimited identifiers proceeds by considering the regular identifier to be all upper case and then doing a case sensitive comparison to the delimited identifier. So a column named `Status` is not identified by the delimited identifier "Status" but is identified by the delimited identifier "STATUS". *end Note*

Ada Semantics

Let `ident` be an identifier. Define AdaID(`ident`) by

```
AdaID(ident) = ident if ident is a regular identifier
               id    if ident is the delimited identifier "id"
```

Note: If `ident` is an identifier, AdaID(`ident`) is not necessarily an *Ada*_identifier.

SQL Semantics

Let SQL_{NAME} be a function on identifiers into the set of *SQL*_identifiers with the property that SQL_{NAME}(`Id_1`) and SQL_{NAME}(`Id_2`) shall be equivalent if and only if Id_1 and Id_2 are equivalent (see above).

Note: Entry level SQL identifiers are limited to eighteen characters in length (see 5.2, leveling rule 2.a of ISO/IEC 9075:1992) but SAMeDL identifiers are not. SQL_{NAME} deals with this discrepancy. SQL_{NAME} is not applied to externally visible names, i.e., schema, table, view and column names, but only to names within SQL modules, e.g., cursor, procedure and parameter names.

5.4 Literals and Data Classes

Literals follow the SQL literal syntax (5.3 of ISO/IEC 9075:1992). There are five classes into which literals are placed based on their lexical properties: **character, integer, fixed, float**, and **enumeration**.

```
literal ::=
    database_literal | enumeration_literal

database_literal ::=
    character_literal | [+ | -] numeric_literal

numeric_literal ::=
    integer_literal | fixed_literal | float_literal

character_literal ::=
    ' {character} '

character ::=
    implementation defined

integer_literal ::=
    digit {digit}

fixed_literal ::=
    integer_literal . integer_literal |
    . integer_literal |
    integer_literal .

float_literal ::=
    fixed_literal exp [+ | -] integer_literal

exp ::=
    e | E

enumeration_literal ::= identifier
```


1. Each literal has an associated data class, denoted DATACLASS(*L*). In particular:

   if *L* is a character_literal then DATACLASS(*L*) is **character**

integer_literal	**integer**
fixed_literal	**fixed**
float_literal	**float**
enumeration_literal	**enumeration**

   (see data_class in 7.1.1.1 of this specification).

2. If `Id` is an enumeration literal, than AdaID(`id`) shall be an *Ada*_identifier. In particular, an enumeration literal shall not be an Ada character literal.