ISO Working Draft

Persistent SQL Modules

«Addendum 2 to SQL-92»

January 1993

Working Draft

(ISO) Persistent Module Addendum Working Draft—January 6, 1993

Working Draft

Digital Equipment Corporation
Maynard, Massachusetts

Contents

Page

Foreword v

Introduction vii

1 Scope 1

2 Normative references 1

3 Definitions, notations, and conventions 1
3.1 Definitions 1
3.2 Notations 1
3.3 Conventions 1

4 Concepts 3
4.1 Modules 3
4.2 Routines 3
4.3 Persistence 4
4.4 Statements 4
4.5 Exception handling 4

5 Persistent SQL modules 5
5.1 Introduction 5
5.2 <module> 6
5.3 <routine> 9
5.4 Data type correspondences 19

6 Persistent SQL module maintenance 27
6.1 Introduction 27
6.2 <module definition> 28
6.3 <drop module statement> 30

7 Persistent SQL module use 31
7.1 Introduction 31
7.2 <call statement> 32
7.3 <return statement> 33

Annex A Place Holder 35

Index

Working Draft

TABLES

Table		Page
1	Data type correspondences for Ada	19
2	Data type correspondences for C	20
3	Data type correspondences for COBOL	21
4	Data type correspondences for Fortran	22
5	Data type correspondences for MUMPS	23
6	Data type correspondences for Pascal	24
7	Data type correspondences for PL/I	25

Working Draft

Foreword

To be supplied.

Working Draft

Introduction

The organization of this International Standard is as follows:

1) To be supplied.

Working Draft

Information Technology — Database Languages — SQL Addendum 2 — Persistent SQL Modules

1 Scope

To be supplied

2 Normative references

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

— ISO/IEC 9075:1992, *Information Technology — Database Languages — SQL.*

More to be supplied as required

3 Definitions, notations, and conventions

3.1 Definitions

To be supplied

3.2 Notations

To be supplied

3.3 Conventions

To be supplied

Working Draft

Working Draft

4 Concepts

4.1 Modules

To be reworked

A <module> is an object specified in the module language. A <module> is either a persistent <module> or an SQL-session <module>. <module>s are created and destroyed by the <create module statement> and <drop module statement>, respectively. They may also be created and destroyed by implementation-defined mechanisms (which can include the granting and revoking of module privileges). A <module> consists of a <module name>, a <language clause>, a <module authorization clause> with either or both of a <module authorization identifier> and a <schema name>, an optional <module character set specification> that identifies the character repertoire used for expressing the names of schema objects used in the <module>, zero or more <temporary table declaration>s, zero or more cursors specified by <declare cursor>s, and one or more <routine>s. All <identifier>s contained in the <module> are expressed in either <SQL language character> or the character repertoire indicated by <module character set specification> unless they are specified with ''<introducer>''.

A compilation unit is a segment of executable code, possibly consisting of one or more subprograms. A <module> is associated with a compilation unit during its execution. A single <module> may be associated with multiple compilation units and multiple <module>s may be associated with a single compilation unit. The manner in which this association is specified, including the possible requirement for execution of some implementation-defined statement, is implementation-defined. Whether a compilation unit may invoke or transfer control to other compilation units, written in the same or a different programming language, is implementation-defined.

A <module> is described by a module descriptor. A module descriptor contains:

- the name of the <module>,
- the descriptor of the character set used for representing the <module>,
- the schema name used for implicit qualification of unqualified names in the <module>
- the <module authorization identifier>,
- the <language clause> of the <module>,
- the text of the <module definition>

4.2 Routines

To be reworked

A <routine> consists of a <routine name>, a sequence of <parameter declaration>, and a <routine body>.

Working Draft

A <routine body> is either an <SQL procedure statement> (which could be a <compound statement>), or an <external body reference> that identifies a routine body that is provided externally as a program written in a standard programming language.

A <routine> is either a component of a <module> or an object in a <schema>.

The <caller language clause> of a <routine> specifies whether it will be invoked from within SQL, by <routine invocation>s or <call statement>s, or by a compilation unit of a specified standard programming language, by means of "call" statements or function invocations.

An invocation of a <routine> specifies the <routine name> of the <routine> and supplies a sequence of argument values corresponding in number and in <data type> to the <parameter declaration>s of the <routine>. Such invocation causes the <SQL procedure statement> or external routine body to be executed.

A <routine> is either a function, which returns a value, or a procedure, which does not return a value. Function invocations can be specified in <value expression>s and <search condition>s in both schemas and modules. Procedure invocations are specified in <call statement>s or in compilation units.

Different <routine>s can have the same <routine name>. This is referred to an *overloading* the <routine name>. Overloading is allowed only for <routine>s that are called from SQL. The parameter declaration lists of such routines must be sufficiently different to distinguish which of the routines is to be invoked for a given <routine invocation>.

A routine is described by a routine descriptor. A routine descriptor contains:

— The <routine name>;

— The name, data type, and position of each of the routine's parameters;

— If the routine is a function, then the data type returned by the function; and

— If the routine is defined within an abstract data type definition or a table definition, then an indicator of its encapsulation level.

4.3 Persistence

To be supplied

4.4 Statements

To be supplied

4.5 Exception handling

To be supplied

Working Draft

5 Persistent SQL modules

5.1 Introduction

To be supplied.

Working Draft

5.2 <module>

Function

Define a module.

Format

```
<module> ::=
       <module name clause>
         <module remainder>
       [ END MODULE ]

<module name clause> ::=
       MODULE [ <module name> ]

<module remainder> ::=
       [ <module character set specification> ]
       <language clause>
       <module authorization clause>
       [ <temporary table declaration>... ]
       <module contents>...

<module character set specification> ::=
       NAMES ARE <character set specification>

<module authorization clause> ::=
         SCHEMA <schema name>
       | AUTHORIZATION <module authorization identifier>
       | SCHEMA <schema name> AUTHORIZATION <module authorization identifier>

<module authorization identifier> ::=
       <authorization identifier>

<module contents> ::=
         <declare cursor>
       | <dynamic declare cursor>
       | <routine>
       | <temporary table declaration>
       | <routine>
       | <temporary view definition>
       | <exception declaration>
```

Syntax Rules

1) If a <module name clause> does not specify a <module name>, then an implementation-dependent <module name> is implicit.

2) The <module name> shall be different from the <module name> of any other <module> in the same environment.

   Note: An SQL-environment may have multiple <module>s that are unnamed.

3) If the <language clause> of the containing <module> specifies ADA, then a <module name> shall be specified, and that <module name> shall be a valid Ada library unit name.

4) If SCHEMA <schema name> is not specified, then a <schema name> equal to <module authorization identifier> is implicit.

Working Draft

5) If a <module character set specification> is not specified, then a <module character set specification> that specifies an implementation-defined character set that contains at least every character that is in <SQL language character> is implicit.

6) If the explicit or implicit <schema name> does not specify a <catalog name>, then an implementation-defined <catalog name> is implicit.

7) The implicit or explicit <catalog name> is the implicit <catalog name> for all unqualified <schema name>s in the <module>.

8) A <declare cursor> or <dynamic declare cursor> shall precede in the text of the <module> any <routine> that references the <cursor name> of the <declare cursor> or <dynamic declare cursor>.

9) An <exception declaration> shall precede in the text of the module any <exception handler condition> that references the <exception name> of the <exception declaration>.

10) For every <declare cursor> in a <module>, there shall be exactly one <routine> in that <module> that contains an <open statement> that specifies the <cursor name> declared in the <declare cursor>.

Note: See the Syntax Rules of Subclause 13.1, "<declare cursor>".

Access Rules

None.

General Rules

1) In the SQL-environment, the containing <module> has the name given by the explicit or implicit <module name>.

2) If the SQL-agent performs calls of <routine>s from more than one Ada task, then the results are implementation-dependent.

3) Case:

   a) If a <module authorization identifier> is specified, then it is the current <authorization identifier> for privilege determination for the execution of each <routine> in the <module>.

   b) Otherwise, the current <authorization identifier> for privilege determination for the execution of each <routine> in the <module> is the SQL-session <authorization identifier>.

4) After the last time that an SQL-agent performs a call of a <routine>:

   a) A <rollback statement> or a <commit statement> is effectively executed. If an unrecoverable error has occurred, or if the SQL-agent terminated unexpectedly, or if any constraint is not satisfied, then a <rollback statement> is performed. Otherwise, the choice of which of these SQL-statements to perform is implementation-dependent. The determination of whether an SQL-agent has terminated unexpectedly is implementation-dependent.

   b) Let *D* be the <descriptor name> of any system descriptor area that is currently allocated within an SQL-session associated with the SQL-agent. A <deallocate descriptor statement> that specifies

      DEALLOCATE DESCRIPTOR *D*

Working Draft

is effectively executed.

c) All SQL-sessions associated with the SQL-agent are terminated.

Leveling Rules

1) The following restrictions apply for Full SQL:

None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

a) A <module> shall not contain a <temporary table declaration>.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

a) A <module character set specification> shall not be specified.

b) A <module> shall be associated with an SQL-agent during its execution. An SQL-agent shall be associated with at most one <module>.

c) A <module contents> shall not be a <dynamic declare cursor>.

d) A <module authorization clause> shall specify AUTHORIZATION and shall not specify SCHEMA.

Working Draft

5.3 <routine>

Function

Define a routine.

Format

```
<routine> ::=
       [CREATE] <routine header> <routine name> <parameter list>
       [<returns clause>]
       [<caller language clause> ] <semicolon>
       <routine body> <semicolon>

<routine header> ::=
         PROCEDURE
       | [ <function type> ] FUNCTION

<function type> ::=
         CONSTRUCTOR
       | ACTOR
       | DESTRUCTOR

<parameter list> ::=
         <parenthesized style list>
       | <unparenthesized style list>

<parenthesized style list> ::=
       <left paren>
         [ <parameter declaration> [{ <comma> <parameter declaration> }... ] ]
       <right paren>

<unparenthesized style list> ::= [ <parameter declaration>... ]

<parameter declaration> ::=
         <SQL parameter>
       | <external parameter>

<SQL parameter> ::=
         <parameter name> <data type>
       | <status parameter>

<status parameter> ::=
         SQLSTATE
       | SQLCODE

<external parameter> ::= [ <parameter mode> ] <data type>

<parameter mode> ::=
         IN
       | OUT
       | INOUT

<returns clause> ::= RETURNS <returns data type> [ <result cast> ]

<result cast> ::= CAST FROM <data type>

<returns data type> ::= <data type>
```

Working Draft

```
<caller language clause> ::= <language clause>

<routine body> ::=
        <SQL routine body>
      | <external body reference>

<SQL routine body> ::= <SQL procedure statement>

<external body reference> ::=
      EXTERNAL [ NAME <external routine name> ]
      <external routine language clause>
      [ <variant attribute> ]

<variant attribute> ::=
        VARIANT
      | NOT VARIANT

<external routine language clause> ::= <language clause>
```

Syntax Rules

1) A <routine> shall contain a <returns clause> if and only if the <routine header> specifies FUNCTION.

2) A <routine> shall specify CREATE if and only if the <routine> is contained in a <module>.

3) A <routine> that specifies PROCEDURE is called a *procedure*. A <routine> that specifies FUNCTION is called a *function*. A <routine> (procedure or function) whose whose <caller language clause> specifies SQL is called an *SQL-invoked <routine>* (procedure or function). A <routine> (procedure or function) whose <caller language clause> does not specify SQL is called an *externally-invoked <routine>* (procedure or function). A <routine> (procedure or function) that specifies an <SQL body> is called an *SQL <routine>* (procedure or function). A <routine> (procedure or function) that specifies an <external body reference> is called an *external <routine>* (procedure or function).

4) Case:

   a) If a <routine> is contained in a <module>, then it shall not specify a <caller language clause>. A <caller language clause> that is the same as the <language clause> of the <module> is implicit.

   b) If a <routine> is not contained in a <module>, then if a <caller language clause> is not specified, then LANGUAGE SQL is implicit.

5) If the <routine header> of a <routine> specifies a <function type>, then:

   a) The implicit or explicit <caller language clause> shall specify LANGUAGE SQL.

   b) The <routine> shall be an SQL routine.

   c) The <routine> shall be contained in an <abstract data type body> *ADTB*.

      Case:

      i) If *ADTB* is immediately contained in an <explicit abstract data type definition> *ADT*, then let *DT* be the <abstract data type name> immediately contained in *ADT*.

Working Draft

ii) If *ADTB* is immediately contained in a <type template definition> *TTD*, then let *DT* be ''<GEN_TYPE''.

d) If the <function type> is CONSTRUCTOR, then:

i) The <returns data type> shall be *DT*.

ii) At least one <parameter declaration> shall specify *DT*.

iii) The <SQL routine body> shall contain exactly one <new statement>, and the operand of that statement shall be a parameter whose data type is *DT*.

e) If the <function type> is DESTRUCTOR, then:

i) Exactly one <parameter declaration> shall specify *DT*.

ii) The <SQL routine body> shall contain exactly one <destroy statement>, and the operand of that statement shall be the parameter whose data type is *DT*.

6) If a <routine> is an SQL-invoked routine, then:

a) The <routine name> shall be a <qualified name>.

b) Case:

i) If the <routine> is contained in a <schema definition>, then let the *subject <schema name>* be the <schema name> of that <schema definition>.

ii) If the <routine> is contained in a <module> that is not contained in a <schema>, then let the *subject <schema name>* be the implicit or explicit <schema name> of the <module name> of that <module>.

c) If the <routine name> contains a <schema name>, then that <schema name> shall be the same as the subject <schema name>.

d) If the <routine name> does not contain a <schema name>, then the subject <schema name> is implicit.

e) The <parameter list> shall not contain a <status parameter>.

f) The *routine family* for a given <qualified identifer> consists of all <routine>s whose <routine name> is equal to that <qualified identifier>. The *routine family* for a <routine> whose <parameter list> contains *N* <parameter declarations> is all members of the routine family of the <routine name> of the <routine> whose <parameter list>s contain *N* <parameter declarations>.

g) All <routine>s in the routine family for a <routine> that is a procedure shall be procedures. All <routine>s in the routine family of a <routine> that is a function shall be functions. If the *i*-th <parameter declaration> of a <routine> specifies an input (output) parameter, then the *i*-th <parameter declaration> of all <routine>s in the routine family of that <routine> shall specify input (output) parameters.

h) For each <data type> specified in a <parameter declaration> or <returns clause> that is an abstract data type, let *ADTD* be the <abstract data type definition> that defined the abstract data type. Either the <routine> shall be contained in *ADTD* or the <routine name> shall be contained in the <operator name list> of *ADTD*.

Working Draft

7) If a <routine> is an externally-invoked routine, then:

   a) The <routine name> shall be an <identifier>.

   b) If the <routine> is contained in a <module>, then the <identifier> of the <routine name> of the <routine> shall be different from the <identifier> of the <routine name> of any other <routine> in the containing <module>.

   c) If the <routine> is contained in a <schema>, then the <identifier> of the <routine name> of the routine shall be different from the <identifier> of the <routine name> of any other <routine> in that <schema>.

   d) The <parameter list> shall contain at least one <status parameter>, at most one <status parameter> that specifies SQLCODE, and at most one <status parameter> that specifies SQLSTATE. A parameter that corresponds with SQLCODE is referred to as an SQLCODE parameter. A parameter than corresponds with SQLSTATE is referred to as an SQLSTATE parameter. The SQLCODE and SQLSTATE parameters are referred to as status parameters.

      Note: SQLSTATE is the preferred status parameter. The SQLCODE status parameter is a deprecated future that is supported for compatibility with earlier versions of this International Standard. See Appendix D, "Deprecated features".

   e) No <parameter declaration> shall specify a <data type> that is an object identifier type containing an <abstract data type definition> that implicitly or explicitly specifies WITH OID NOT VISIBLE.

   f) When the <external routine language clause> specifies ADA, C, COBOL, FORTRAN, MUMPS, PASCAL, or PLI, then let the *operative data type correspondences table* be Table 1, "Data type correspondences for Ada", Table 2, "Data type correspondences for C", Table 3, "Data type correspondences for COBOL", Table 4, "Data type correspondences for Fortran", Table 5, "Data type correspondences for MUMPS", Table 6, "Data type correspondences for Pascal", or Table 7, "Data type correspondences for PL/I", respectively. Refer to the two columns of the operative data type correspondences table as the ''SQL data type'' column and the ''host data type column''.

   g) If the <routine> is a function, then let the externally returned data type be the <data type> specified by the <returns data type>.

   h) Any <data type> in a <parameter declaration> or externally returned data type (if any) shall specify a data type listed in the SQL data type column for which the corresponding row in the host data type column is not ''none''. If the <data type> in a <parameter declaration> or externally returned data type (if any) specifies the data type listed in the *i*-th row of the SQL data type column, then the type of the *i*-th parameter or the externally returned data type shall be the type listed in the *i*-th row of the host data type column.

   i) If the <calle language clause> specifies ADA, then:

      i) The <routine>s of the <module> are identified by the <procedure name> as if they were declared immediately within an Ada library unit package specification that has a name identical to the <module name> of the containing <module>.

      ii) The Ada base type of any parameter shall be an Ada data type declared in an Ada package named SQL_STANDARD of the following form:

```
package SQL_STANDARD is
        package CHARACTER_SET renames csp;
```

Working Draft

```
subtype CHARACTER_TYPE is CHARACTER_SET.cst;
type CHAR is array (POSITIVE range <>) of CHARACTER_TYPE;
type BIT is array (NATURAL range <>) of BOOLEAN;
type SMALLINT is range bs .. ts;
type INT is range bi .. ti;
type REAL is digits dr;
type DOUBLE_PRECISION is digits dd;
subtype INDICATOR_TYPE is t;
type SQLCODE_TYPE is range bsc .. tsc;
subtype SQL_ERROR is SQLCODE_TYPE range SQL_TYPE'FIRST .. -1;
subtype NOT_FOUND is SQLCODE_TYPE range 100 .. 100;
type SQLSTATE_TYPE is new CHAR (1 .. 5);
package SQLSTATE_CODES is
    AMBIGUOUS_CURSOR_NAME_NO_SUBCLASS:
            constant SQLSTATE_TYPE :="3C000";
    CASE_NOT_FOUND_FOR_CASE_STATEMENT:
            constant SQLSTATE_TYPE :="20000";
    CARDINALITY_VIOLATION_NO_SUBCLASS:
            constant SQLSTATE_TYPE :="21000";
    CONNECTION_EXCEPTION_NO_SUBCLASS:
            constant SQLSTATE_TYPE :="08000";
    CONNECTION_EXCEPTION_CONNECTION_DOES_NOT_EXIST:
            constant SQLSTATE_TYPE :="08003";
    CONNECTION_EXCEPTION_CONNECTION_FAILURE:
            constant SQLSTATE_TYPE :="08006";
    CONNECTION_EXCEPTION_CONNECTION_NAME_IN_USE:
            constant SQLSTATE_TYPE :="08002";
    CONNECTION_EXCEPTION_SQLCLIENT_UNABLE_TO_ESTABLISH_SQLCONNECTION:
            constant SQLSTATE_TYPE :="08001";
    CONNECTION_EXCEPTION_SQLSERVER_REJECTED_ESTABLISHMENT_OF_SQLCONNECTION:
            constant SQLSTATE_TYPE :="08004";
    CONNECTION_EXCEPTION_UNKNOWN_STATEMENT_COMPLETION:
            constant SQLSTATE_TYPE :="08005";
    CURSOR_OPERATION_CONFLICT_NO_SUBCLASS:
            constant SQLSTATE_TYPE :="09000";
    DATA_EXCEPTION_NO_SUBCLASS:
            constant SQLSTATE_TYPE :="22000";
    DATA_EXCEPTION_CHARACTER_NOT_IN_REPERTOIRE:
            constant SQLSTATE_TYPE :="22021";
    DATA_EXCEPTION_DATETIME_FIELD_OVERFLOW:
            constant SQLSTATE_TYPE :="22008";
    DATA_EXCEPTION_DIVISION_BY_ZERO:
            constant SQLSTATE_TYPE :="22012";
    DATA_EXCEPTION_ERROR_IN_ASSIGNMENT:
            constant SQLSTATE_TYPE :="22005";
    DATA_EXCEPTION_INDICATOR_OVERFLOW:
            constant SQLSTATE_TYPE :="22022";
    DATA_EXCEPTION_INTERVAL_FIELD_OVERFLOW:
            constant SQLSTATE_TYPE :="22015";
    DATA_EXCEPTION_INVALID_CHARACTER_VALUE_FOR_CAST:
            constant SQLSTATE_TYPE :="22018";
    DATA_EXCEPTION_INVALID_DATETIME_FORMAT:
            constant SQLSTATE_TYPE :="22007";
    DATA_EXCEPTION_INVALID_ESCAPE_CHARACTER:
            constant SQLSTATE_TYPE :="22019";
    DATA_EXCEPTION_INVALID_ESCAPE_SEQUENCE:
            constant SQLSTATE_TYPE :="22025";
    DATA_EXCEPTION_INVALID_FETCH_SEQUENCE:
            constant SQLSTATE_TYPE :="22006";
    DATA_EXCEPTION_INVALID_PARAMETER_VALUE:
            constant SQLSTATE_TYPE :="22023";
    DATA_EXCEPTION_INVALID_TIME_ZONE_DISPLACEMENT_VALUE:
            constant SQLSTATE_TYPE :="22009";
    DATA_EXCEPTION_NULL_VALUE_NO_INDICATOR_PARAMETER:
```

Working Draft

```
        constant SQLSTATE_TYPE :="22002";
DATA_EXCEPTION_NUMERIC_VALUE_OUT_OF_RANGE:
        constant SQLSTATE_TYPE :="22003";
DATA_EXCEPTION_STRING_DATA_LENGTH_MISMATCH:
        constant SQLSTATE_TYPE :="22026";
DATA_EXCEPTION_STRING_DATA_RIGHT_TRUNCATION:
        constant SQLSTATE_TYPE :="22001";
DATA_EXCEPTION_SUBSTRING_ERROR:
        constant SQLSTATE_TYPE :="22011";
DATA_EXCEPTION_TRIM_ERROR:
        constant SQLSTATE_TYPE :="22027";
DATA_EXCEPTION_UNTERMINATED_C_STRING:
        constant SQLSTATE_TYPE :="22024";
DEPENDENT_PRIVILEGE_DESCRIPTORS_STILL_EXIST_NO_SUBCLASS:
        constant SQLSTATE_TYPE :="2B000";
DYNAMIC_SQL_ERROR_NO_SUBCLASS:
        constant SQLSTATE_TYPE :="07000";
DYNAMIC_SQL_ERROR_CURSOR_SPECIFICATION_CANNOT_BE_EXECUTED:
        constant SQLSTATE_TYPE :="07003";
DYNAMIC_SQL_ERROR_INVALID_DESCRIPTOR_COUNT:
        constant SQLSTATE_TYPE :="07008";
DYNAMIC_SQL_ERROR_INVALID_DESCRIPTOR_INDEX:
        constant SQLSTATE_TYPE :="07009";
DYNAMIC_SQL_ERROR_PREPARED_STATEMENT_NOT_A_CURSOR_SPECIFICATION:
        constant SQLSTATE_TYPE :="07005";
DYNAMIC_SQL_ERROR_RESTRICTED_DATA_TYPE_ATTRIBUTE_VIOLATION:
        constant SQLSTATE_TYPE :="07006";
DYNAMIC_SQL_ERROR_USING_CLAUSE_DOES_NOT_MATCH_DYNAMIC_PARAMETER_SPEC:
        constant SQLSTATE_TYPE :="07001";
DYNAMIC_SQL_ERROR_USING_CLAUSE_DOES_NOT_MATCH_TARGET_SPEC:
        constant SQLSTATE_TYPE :="07002";
DYNAMIC_SQL_ERROR_USING_CLAUSE_REQUIRED_FOR_DYNAMIC_PARAMETERS:
        constant SQLSTATE_TYPE :="07004";
DYNAMIC_SQL_ERROR_USING_CLAUSE_REQUIRED_FOR_RESULT_FIELDS:
        constant SQLSTATE_TYPE :="07007";
FEATURE_NOT_SUPPORTED_NO_SUBCLASS:
        constant SQLSTATE_TYPE :="0A000";
FEATURE_NOT_SUPPORTED_MULTIPLE_ENVIRONMENT_TRANSACTIONS:
        constant SQLSTATE_TYPE :="0A001";
INTEGRITY_CONSTRAINT_VIOLATION_NO_SUBCLASS:
        constant SQLSTATE_TYPE :="23000";
INVALID_AUTHORIZATION_SPECIFICATION_NO_SUBCLASS:
        constant SQLSTATE_TYPE :="28000";
INVALID_CATALOG_NAME_NO_SUBCLASS:
        constant SQLSTATE_TYPE :="3D000";
INVALID_CHARACTER_SET_NAME_NO_SUBCLASS:
        constant SQLSTATE_TYPE :="2C000";
INVALID_CONDITION_NUMBER_NO_SUBCLASS:
        constant SQLSTATE_TYPE :="35000";
INVALID_CONNECTION_NAME_NO_SUBCLASS:
        constant SQLSTATE_TYPE :="2E000";
INVALID_CURSOR_NAME_NO_SUBCLASS:
        constant SQLSTATE_TYPE :="34000";
INVALID_CURSOR_STATE_NO_SUBCLASS:
        constant SQLSTATE_TYPE :="24000";
INVALID_SCHEMA_NAME_NO_SUBCLASS:
        constant SQLSTATE_TYPE :="3F000";
INVALID_SQL_DESCRIPTOR_NAME_NO_SUBCLASS:
        constant SQLSTATE_TYPE :="33000";
INVALID_SQL_STATEMENT_NAME_NO_SUBCLASS:
        constant SQLSTATE_TYPE :="26000";
INVALID_TRANSACTION_STATE_NO_SUBCLASS:
        constant SQLSTATE_TYPE :="25000";
INVALID_TRANSACTION_TERMINATION_NO_SUBCLASS:
```

Working Draft

```
        constant SQLSTATE_TYPE :="2D000";
NO_DATA_NO_SUBCLASS:
        constant SQLSTATE_TYPE :="02000";
REMOTE_DATABASE_ACCESS_NO_SUBCLASS:
        constant SQLSTATE_TYPE :="HZ000";
SUCCESSFUL_COMPLETION_NO_SUBCLASS:
        constant SQLSTATE_TYPE :="00000";
SYNTAX_ERROR_OR_ACCESS_RULE_VIOLATION_NO_SUBCLASS:
        constant SQLSTATE_TYPE :="42000";
SYNTAX_ERROR_OR_ACCESS_RULE_VIOLATION_IN_DIRECT_STATEMENT_NO_SUBCLASS:
        constant SQLSTATE_TYPE :="2A000";
SYNTAX_ERROR_OR_ACCESS_RULE_VIOLATION_IN_DYNAMIC_STATEMENT_NO_SUBCLASS:
        constant SQLSTATE_TYPE :="37000";
TRANSACTION_ROLLBACK_NO_SUBCLASS:
        constant SQLSTATE_TYPE :="40000";
TRANSACTION_ROLLBACK_INTEGRITY_CONSTRAINT_VIOLATION:
        constant SQLSTATE_TYPE :="40002";
TRANSACTION_ROLLBACK_SERIALIZATION_FAILURE:
        constant SQLSTATE_TYPE :="40001";
TRIGGERED_DATA_CHANGE_VIOLATION_NO_SUBCLASS:
        constant SQLSTATE_TYPE :="27000";
WARNING_NO_SUBCLASS:
        constant SQLSTATE_TYPE :="01000";
WARNING_CONSTRAINT_CURSOR_OPERATION_CONFLICT:
        constant SQLSTATE_TYPE :="01001";
WARNING_DISCONNECT_ERROR:
        constant SQLSTATE_TYPE :="01002";
WARNING_IMPLICIT_ZERO_BIT_PADDING:
        constant SQLSTATE_TYPE :="01008";
WARNING_INSUFFICIENT_ITEM_DESCRIPTOR_AREAS:
        constant SQLSTATE_TYPE :="01005";
WARNING_NULL_VALUE_ELIMINATED_IN_SET_FUNCTION:
        constant SQLSTATE_TYPE :="01003";
WARNING_PRIVILEGE_NOT_GRANTED:
        constant SQLSTATE_TYPE :="01007";
WARNING_PRIVILEGE_NOT_REVOKED:
        constant SQLSTATE_TYPE :="01006";
WARNING_SEARCH_CONDITION_TOO_LONG_FOR_INFORMATION_SCHEMA:
        constant SQLSTATE_TYPE :="01009";
WARNING_STRING_DATA_RIGHT_TRUNCATION_WARNING:
        constant SQLSTATE_TYPE :="01004";
WARNING_QUERY_EXPRESSION_TOO_LONG_FOR_INFORMATION_SCHEMA:
        constant SQLSTATE_TYPE :="0100A";
WITH_CHECK_OPTION_VIOLATION_NO_SUBCLASS:
        constant SQLSTATE_TYPE :="44000";
    end SQLSTATE_CODES;
end SQL_STANDARD;
```

where *csp* is an implementation-defined package and *cst* is an implementation-defined character type such that within the scope of an Ada **use** clause for SQL_STANDARD.CHARACTER_SET, string literals can be of type SQL_STANDARD.CHAR. *bs*, *ts*, *ti*, *dr*, *dd*, *bsc*, and *tsc* are implementation-defined integer values. *t* is INT or SMALLINT, corresponding with an implementation-defined <exact numeric type> of indicator parameters.

SQL_STANDARD shall contain no other declarations.

8) If a <routine> is an SQL routine, then:

   a) Each <parameter declaration> in the <parameter list> shall be an <SQL parameter>.

Working Draft

b) The <parameter name> of each <SQL parameter> in a <parameter list> shall be different from the <parameter name> of any other <SQL parameter> in that <parameter list>.

c) Whether a <parameter declaration> is for an *input parameter*, an *output parameter*, or both is determined as follows:

Case:

i) A <status parameter> is an *output parameter*.

ii) For every <parameter declaration> that is not a <status parameter>,

Case:

1) If the <parameter name> of a parameter is contained in a <value specification> or a <simple value specification> that is contained in <SQL procedure statement>, but it is not contained in a <target specification> or a <simple target specification> that is contained in <SQL procedure statement>, then the parameter is an *input parameter*.

2) If the <parameter name> of a parameter is contained in a <target specification> or a <simple target specification> that is contained in <SQL procedure statement>, but it is not contained in a <value specification> or a <simple value specification> that is contained in <SQL procedure statement>, then the parameter is an *output parameter*.

3) If the <parameter name> of a parameter is contained in a <value specification> or a <simple value specification> that is contained in <SQL procedure statement> and it is contained in a <target specification> or a <simple target specification> that is contained in <SQL procedure statement>, then the parameter is both an *input parameter* and an *output parameter*.

4) Otherwise, the parameter is neither an *input parameter* nor an *output parameter*.

d) Any <parameter name> contained in the <SQL routine body> of a <routine> shall be specified in an <SQL parameter> of the <routine>.

e) If the <SQL routine body> contains an <enumeration literal> that is identical to a <parameter name> in the <parameter list>, then the <enumeration literal> shall contain a <schema name>.

f) The <returns clause> shall not specify a <result cast>.

g) An <SQL routine body> shall contain a <return statement> if and only if the <routine> is a function.

9) If a <routine> is an external routine, then:

a) Each <parameter declaration> in the <parameter list> shall be an <external parameter>.

b) The implicit or explicit <caller language clause> shall specify SQL.

c) If an <external routine name> is not specified, then the <qualified identifier> of the <routine name> is implicit.

d) If a <variant attribute> is not specified, then VARIANT is implicit.

e) The <external routine language clause> shall not specify SQL.

Working Draft

f) If a <result cast> is specified, then let *R* be some value of the <data type> specified in the <result cast> and let *RT* be the <returns data type>. The following shall be valid according to the Syntax Rules of Subclause 6.10, "<cast specification>":

CAST (*R* AS *RT*)

g) If <parameter mode> is not specified, then IN is implicit.

h) If <parameter mode> is IN or INOUT, then the parameter is an *input parameter*. If the <parameter mode> is OUT or INOUT, then the parameter is an *output parameter*.

i) Let *N* be the number of <external parameter>s. Let *PN* be the number of <external parameter>s that are predefined types. Let *UN* be the number of actual base types of the operand data types that are abstract data types. If the <returns data type> is an abstract data type, then let *RN* be the number of actual base types of that data type; otherwise, let *RN* be 1. Let the *effective external parameter type list* be a list of $PN + UN + RN + N + 2$ data types, as follows:

i) Effective parameter type list entries 1 to $PN+UN$ are *N* groups of entries. For *i* ranging from 1 to *N*,

Case:

1) If the *i*-th <external parameter> is a predefined type, then the *i*-th group of effective parameter type list entries consists of the *i*-th <external parameter>.

2) If the *i*-th <external parameter> is some abstract data type ADT_i, then the *i*-th group of effective parameter type list entries consists of the <parameter mode> of the <external parameter> and the actual base types of the <abstract data type definition> of ADT_i.

ii) Effective parameter type list entries $(PN + UN) + 1$ to $(PN + UN) + RN$ are as follows:

Case:

1) If the <returns data type> is a predefined type, then effective external parameter list entry $(PN + UN) + 1$ is the <parameter mode> OUT and the <returns data type>.

2) If the <returns data type> is some abstract data type ADT_r, then effective external parameter list entries (PN+UN)+1 to $(PN + UN) + RN$ consist of the <parameter mode> OUT and the actual base types of the <abstract data type definition> of ADT_r.

iii) Effective external parameter list entries $(PN+UN+RN)+1$ to $(PN+UN+RN)+N+1$ are $N + 1$ occurrences of an implementor-defined <data type> that is an exact numeric type with scale 0. The <parameter mode> for the *i*-th such effective external parameter is the same as that of the $i - RN - PN - UN$-th effective external parameter.

iv) Effective parameter type list entry $(PN + UN + RN + N + 1) + 1$ is a <data type> that is character string of length 3 and character set SQL_TEXT with <parameter mode> OUT.

j) Depending on whether the <caller language clause> specifies ADA, C, COBOL, FORTRAN, MUMPS, PASCAL, or PLI, let the *operative data type correspondences table* be Table 1, "Data type correspondences for Ada", Table 2, "Data type correspondences for C", Table 3, "Data type correspondences for COBOL", Table 4, "Data type correspondences for Fortran", Table 5, "Data type correspondences for MUMPS", Table 6, "Data type correspondences for Pascal", or Table 7, "Data type correspondences for PL/I", respectively. Refer to the two

Working Draft

columns of the operative data type correspondences table as the ''SQL data type'' column and the ''host data type column''.

k) If the <routine> is a function, then if the <routine> contains a <result cast>, let the externally returned data type be the <data type> contained in that <result cast>; otherwise, let the externally returned data type be the <data type> contained in the <returns clause>.

l) Any <data type> in a <parameter declaration> or externally returned data type (if any) shall specify a data type listed in the SQL data type column for which the corresponding row in the host data type column is not ''none''. If the <data type> in a <parameter declaration> or externally returned data type (if any) specifies the data type listed in the *i*-th row of the SQL data type column, then the type of the *i*-th parameter or the externally returned data type shall be the type listed in the *i*-th row of the host data type column.

Access Rules

1) If an external routine is contained in a <module>, then the <user authorization identifier> of the current SQL-session shall be equal to the <user authorization identifier> that owns the schema identified by the implicit or explicit <schema name> of the <routine>.

General Rules

1) If the <routine> is an external routine, then a privilege descriptor is created that defines the EXTERNAL PRIVILEGES privilege on the <routine> to the <authorization identifier> of the schema or <module> in which the <routine> appears. This privilege is grantable. The grantor for the privilege descriptor is set to the special grantor value ''_SYSTEM''.

2) Case:

a) If a <routine> is an SQL-invoked routine, then the rules for invocation of the <routine> are specified in <REFERENCE>(routine_invocation\FULL).

b) If a <routine> is an externally-invoked routine, then the rules for invocation of the <routine> are specified in <REFERENCE>(external_invocation\FULL).

Leveling Rules

1) The following restrictions apply for Full SQL:

a) A <routine> shall be a routine shall be contained in a <module>, and shall not contain a <caller language clause>.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

None.

Working Draft

5.4 Data type correspondences

Function

Specify the data type correspondences for SQL data types and host language types.

Note: These tables are referenced in <routine\FULL), for the definitions of externally-invoked routines and external routines.

Tables

Table 1—Data type correspondences for Ada

SQL Data Type	Ada Data Type
SQLSTATE	SQL_STANDARD.SQLSTATE_TYPE
SQLCODE	SQL_STANDARD.SQLCODE_TYPE
CHARACTER(*L*)	SQL_STANDARD.CHAR, with P'LENGTH of *L*
CHARACTER VARYING(*L*)	*None*
BIT(*L*)	SQL_STANDARD.BIT, with P'LENGTH of *L*
BIT VARYING(*L*)	*None*
BOOLEAN	SQL_STANDARD.BOOLEAN
SMALLINT	SQL_STANDARD.SMALLINT
INTEGER	SQL_STANDARD.INT
DECIMAL(*P*,*S*)	*None*
NUMERIC(*P*,*S*)	*None*
REAL	SQL_STANDARD.REAL
DOUBLE PRECISION	SQL_STANDARD.DOUBLE_PRECISION
FLOAT(*P*)	*None*
object ADT	SQL_STANDARD.CHAR, with P'LENGTH of *X*[1]

[1]The length *X* of the character data type corresponding with an SQL object ADT is the implementor-defined length of an object identifier.

Working Draft

Table 2—Data type correspondences for C

SQL Data Type	C Data Type
SQLCODE	pointer to long
SQLSTATE	char, with length 6
CHARACTER(*L*)	char, with length *L*+1
CHARACTER VARYING (*L*)	char, with length *L*+1
BIT(*L*)	char, with length *X*[1]
BIT VARYING(*L*)	*None*
BOOLEAN	pointer to long
SMALLINT	pointer to short
INTEGER	pointer to long
DECIMAL(*P*,*S*)	None
NUMERIC(*P*,*S*)	None
REAL	pointer to float
DOUBLE PRECISION	pointer to double
FLOAT(*P*)	*None*
object ADT	char, with length *Y*[2]

[1]The length *X* of the character data type corresponding with SQL data type BIT(*L*) is the smallest integer not less than the quotient of the division *L*/*B*, where *B* is the implementation-defined number of bits contained in character of the host language.

[2]The length *Y* of the character data type corresponding with an SQL object ADT is the implementor-defined length of an object identifier.

Working Draft

Table 3—Data type correspondences for COBOL

SQL Data Type	COBOL Data Type
SQLSTATE	PICTURE X(5)
SQLCODE	PICTURE S9(*PC*) USAGE COMPUTATIONAL, where *PC* is implementation-defined between 4 and 18, inclusive
CHARACTER(*L*)	alphanumeric, with length *L*
CHARACTER VARYING(L)	*None*
BIT(*L*)	alphanumeric, with length *X*[1]
BIT VARYING(*L*)	*None*
BOOLEAN	PICTURE X
SMALLINT	PICTURE S9(*SPI*) USAGE BINARY, where *SPI* is implementation-defined
INTEGER	PICTURE S9(*PI*) USAGE BINARY, where *PI* is implementation-defined
DECIMAL(*P*,*S*)	*None*
NUMERIC(*P*,*S*)	USAGE DISPLAY SIGN LEADING SEPARATE, with PICTURE as specified[2]
REAL	*None*
DOUBLE PRECISION	*None*
FLOAT(*P*)	*None*
object ADT	alphanumeric, with length *Y*[3]

[1]The length of a character type corresponding with SQL quotient of the division *L*/*B*, where *B* is the implementation-defined number of bits contained in one character of the host language.

[2]Case:

a) If *S*=*P*, then a PICTURE with an 'S' followed by a 'V' followed by *P* '9's.
b) If *P*>*S*>0, then a PICTURE with an 'S' followed by *S* '9's followed by a 'V' followed by *S* '9's.
c) If *S*=0, then a PICTURE with an 'S' followed by *P* '9's optionally followed by a 'V'.

[3]The length *Y* of the character data type corresponding with an SQL object ADT is the implementor-defined length of an object identifier.

Working Draft

Table 4—Data type correspondences for Fortran

SQL Data Type	Fortran Data Type
SQLSTATE	CHARACTER, with length 5
SQLCODE	INTEGER
CHARACTER(*L*)	CHARACTER, with length *L*
CHARACTER VARYING(*L*)	*None*
BIT(*L*)	CHARACTER, with length *X*[1]
BIT VARYING(*L*)	*None*
BOOLEAN	LOGICAL
SMALLINT	*None*
INTEGER	INTEGER
DECIMAL(*P*,*S*)	*None*
NUMERIC(*P*,*S*)	*None*
REAL	REAL
DOUBLE PRECISION	DOUBLE PRECISION
FLOAT(*P*)	*None*
object ADT	CHARACTER, with length *Y*[2]

[1]The length *X* of the character data type corresponding with SQL data type BIT(*L*) is the smallest integer not less than the quotient of the division *L*/*B*, where *B* is the implementation-defined number of bits contained in character of the host language.

[2]The length *Y* of the character data type corresponding with an SQL object ADT is the implementor-defined length of an object identifier.

Working Draft

Table 5—Data type correspondences for MUMPS

SQL Data Type	MUMPS Data Type
SQLSTATE	character, with maximum length at least 5
SQLCODE	*None*
CHARACTER(*L*)	*None*
CHARACTER VARYING (*L*)	character with maximum length *L*
BIT(*L*)	*None*
BIT VARYING(*L*)	*None*
BOOLEAN	*None*
SMALLINT	*None*
INTEGER	character
DECIMAL(*P*,*S*)	character
NUMERIC(*P*,*S*)	character
REAL	character
DOUBLE PRECISION	*None*
FLOAT(*P*)	*None*
object ADT	CHARACTER, with length *X*[1]

[1]The length *X* of the character data type corresponding with an SQL object ADT is the implementor-defined length of an object identifier.

Working Draft

Table 6—Data type correspondences for Pascal

SQL Data Type	Pascal Data Type
SQLSTATE	PACKED ARRAY[1..5] OF CHAR
SQLCODE	INTEGER
CHARACTER(1)	CHAR
CHARACTER(*L*), *L*>1	PACKED ARRAY[1..*L*] OF CHAR
CHARACTER VARYING(L)	*None*
BIT(*L*), 1 $\leq L \leq B^1$	CHAR
BIT(*L*), $B^1 < L$	PACKED ARRAY[LB[1]] OF CHAR
BIT VARYING(*L*)	*None*
BOOLEAN	BOOLEAN
SMALLINT	*None*
INTEGER	INTEGER
DECIMAL(*P*,*S*)	*None*
NUMERIC(*P*,*S*)	*None*
REAL	REAL
DOUBLE PRECISION	*None*
FLOAT(*P*)	*None*
object ADT	CHAR, with length X^2

[1]The length *LB* of the character data type corresponding with SQL data type BIT(*L*) is the smallest integer not less than the quotient of the division *L*/*B*, where *B* is the implementation-defined number of bits contained in a character of the host language.

[2]The length *X* of the character data type corresponding with an SQL object ADT is the implementor-defined length of an object identifier.

Working Draft

Table 7—Data type correspondences for PL/I

SQL Data Type	PL/I Data Type
SQLSTATE	CHARACTER(5)
SQLCODE	FIXED BINARY(*PP*), where *PP* is an implementation-defined precision at least 15.
CHARACTER(*L*)	CHARACTER(*L*)
CHARACTER VARYING(*L*)	CHARACTER VARYING(*L*)
BIT(*L*)	BIT(*L*)
BIT VARYING(*L*)	BIT VARYING (*L*)
BOOLEAN	BIT(1)
SMALLINT	FIXED BINARY(*SPI*), where *SPI* is implementation-defined
INTEGER	FIXED BINARY(*PI*), where *PI* is implementation-defined
DECIMAL(*P*,*S*)	FIXED REAL DECIMAL(*P*,*S*)
NUMERIC(*P*,*S*)	*None*
REAL	*None*
DOUBLE PRECISION	*None*
FLOAT(*P*)	FLOAT(*P*)
object ADT	CHARACTER with length *X*[1]

[1]The length *X* of the character data type corresponding with an SQL object ADT is the implementor-defined length of an object identifier.

Working Draft

Working Draft

6 Persistent SQL module maintenance

6.1 Introduction

To be supplied.

Working Draft

6.2 <module definition>

Function

Define a module.

Format

```
<module definition> ::=
       CREATE MODULE <module name>
         <module remainder>
       END MODULE
```

Syntax Rules

1) Case:

   a) If a <module definition> is contained in a <schema definition>, and if the <module definition> contains a <schema name>, then that <schema name> shall be the same as the specified or implicit <schema name> of the containing <schema definition>.

   b) If a <module definition> is contained in a <module> other than in a <schema definition>, and if the <module definition> contains a <schema name>, then that <schema name> shall be the same as the specified or implicit <schema name> of the containing <module>.

2) The <qualified identifier> of <module name> shall be different from the <qualified identifier> of any other <module name> of a module definition in the same schema.

Access Rules

1) If a <module definition> is contained in a <module>, then the <authorization identifier> of the current SQL-session shall be equal to the <authorization identifier> that owns the schema identified by the implicit or explicit <schema name> of the <module name>.

General Rules

1) A <module definition> defines a module.

2) A privilege descriptor is created that defines the EXECUTE privilege on this module to the <authorization identifier> of the <schema> or <module> in which the <module definition> appears. This privilege is grantable. The grantor of the privilege descriptor is set to the special grantor value ''_SYSTEM''.

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) Conforming Full SQL language shall contain no <module definition>.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

Working Draft

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

None.

Working Draft

6.3 <drop module statement>

Function

Destroy a module.

Format

```
<drop module statement> ::=
       DROP MODULE <module name> <drop behavior>
```

Syntax Rules

1) Let *M* be the module identified by <module name> and let *MN* be the <module name>.

Access Rules

1) The <authorization identifier> of the current SQL-session shall be equal to the <authorization identifier> that owns the schema identified by the <schema name> of the module identified by *MN*.

General Rules

1) Let *A* be the current <authorization identifier>. The following <revoke statement> is effectively executed with a current <authorization identifier> of "_SYSTEM" and without further Access Rule checking:

   REVOKE EXECUTE ON MODULE *M* FROM *A*

2) The descriptor of *M* is destroyed.

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) Conforming Full SQL language shall contain no <drop module statement>.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

Working Draft

7 Persistent SQL module use

7.1 Introduction

To be supplied.

Working Draft

7.2 <call statement>

Function

Invoke a procedure.

Format

```
<call statement> ::=
       CALL <routine invocation>
```

Syntax Rules

1) A <call statement> shall contain a <routine invocation> whose candidate <routine>s are procedures.

   Note: The *candidate <routine>s* of a <routine invocation> are defined in <REFERENCE>(routine_invocation\FULL).

Access Rules

1) The applicable privileges shall include the EXECUTE privilege on the <routine> identified by <routine name>.

General Rules

1) The <routine invocation> is executed.

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) Conforming Full SQL language shall contain no <call statement>.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

Working Draft

7.3 <return statement>

Function

Return a value from an SQL function.

Format

```
<return statement> ::=
       RETURN <return value>

<return value> ::=
         <tuple constructor>
       | <value expression>

<tuple constructor> ::=
       TUPLE <left paren> <value expression list> <right paren>

<value expression list> ::=
       <value expression> [ { <comma> <value expression> } ... ]
```

Syntax Rules

1) A <return statement> shall be contained in a <routine> that is an SQL function. Let *F* be that <routine>.

2) Case:

   a) If the <returns type> of *F* specifies a <value expression>, then the data type of the <value expression> shall be the same as the <data type> specified with RETURNS in *F*.

   b) If the <returns type> of *F* specifies a <tuple constructor>, then:

      i) *F* shall be contained in an <abstract data type definition> *ADT*. Let *TD* be the sequence of stored columns in *ADT*.

      ii) The <data type> specified in the <returns clause> of *F* shall identify the <abstract data type name> of *ADT*.

      iii) The number of <value expression>s simply contained in the <tuple constructor> shall be the same as the number of <component definition>s contained in *TD*.

      iv) Let TU_i be the i-th <value expression> in the <tuple constructor> and let TD_i be the data type of the i-th column of *TD*. CAST (TU_i AS TD_i) shall conform to the Syntax Rules of Subclause 6.10, "<cast specification>".

Access Rules

None.

Working Draft

General Rules

1) Case:

   a) If the <returns type> is a <value expression>, then the return value is set to the value of that <value expression>.

   b) If the <returns type> is a <tuple constructor>, then:

      i) For every <value expression> in the <tuple constructor>, the General Rules for CAST (TU_i AS TD_i) are performed.

      ii) The return value is set to a value whose type is that of *TN* and whose *i*th component value is CAST (TU_i AS TD_i).

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) Conforming Full SQL language shall contain no <return statement>.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

Working Draft

Annex A
(Informative)

Place Holder

To be supplied.

Working Draft

Working Draft