9 Additional common rules

9.1 Retrieval assignment

Function

Specify rules for assignments to targets that do not support null values or that support null values with indicator parameters (*e.g.*, assigning SQL-data to host parameters or host variables).

Syntax Rules

1) Let *T* and *V* be the *TARGET* and *VALUE* specified in an application of this Subclause. Let the declared types of *T* and *V* be *TD* and *SD*, respectively.

2) If *TD* is binary string, numeric, boolean, datetime, interval, or a user-defined type, then either *SD* shall be assignable to *TD* or there shall exist an appropriate user-defined cast function *UDCF* from *SD* to *TD*.

   NOTE 202 — "Appropriate user-defined cast function" is defined in Subclause 4.11, "Data conversions".

3) If *TD* is character string, then

   Case:

   a) If *T* is either a locator parameter of an external routine, a locator variable, or a host parameter that is a character large object locator parameter, then *SD* shall be CHARACTER LARGE OBJECT and *SD* shall be assignable to *TD*.

   b) Otherwise, either *SD* shall be assignable to *TD* or there shall exist an appropriate user-defined cast function *UDCF* from *SD* to *TD*.

4) If the declared type of *T* is a reference type, then the declared type of *V* shall be a reference type whose referenced type is a subtype of the referenced type of *T*.

5) If the declared type of *T* is a row type, then:

   a) The declared type of *V* shall be a row type.

   b) The degree of *V* shall be the same as the degree of *T*. Let *n* be that degree.

   c) Let TT_i, 1 (one) $\leq i \leq n$, be the declared type of the *i*-th field of *T*, let VT_i be the declared type of the *i*-th field of *V*, let $T1_i$ be an arbitrary target whose declared type is TT_i, and let $V1_i$ be an arbitrary expression whose declared type is VT_i. For each *i*, 1 (one) $\leq i \leq n$, the Syntax Rules of this Subclause apply to T_i V_i, as *TARGET* and *VALUE*, respectively.

6) If the declared type of *T* is a collection type, then:

   a) If the declared type of *T* is an array type, then the declared type of *V* shall be an array type.

b) If the declared type of *T* is a multiset type, then the declared type of *V* shall be a multiset type.

c) Let *TT* be the element type of the declared type of *T*, let *VT* be the element type of the declared type of *V*, let *T1* be an arbitrary target whose declared type is *TT*, and let *V1* be an arbitrary expression whose declared type is *VT*. The Syntax Rules of this Subclause apply to *T1* and *V1*, as *TARGET* and *VALUE*, respectively.

Access Rules

None.

General Rules

1) If the declared type of *V* is not assignable to the declared type of *T*, then for the remaining General Rules of this Subclause *V* is effectively replaced by the result of evaluating the expression *UDCF*(*V*).

2) If *V* is the null value and *T* is a host parameter, then

   Case:

   a) If an indicator parameter is specified for *T*, then that indicator parameter is set to –1.

   b) If no indicator parameter is specified for *T*, then an exception condition is raised: *data exception — null value, no indicator parameter.*

3) If *V* is the null value and *T* is a host variable, then

   Case:

   a) If an indicator variable is specified for *T*, then that indicator variable is set to –1.

   b) If no indicator variable is specified for *T*, then an exception condition is raised: *data exception — null value, no indicator parameter.*

4) If *V* is not the null value, *T* is a host parameter, and *T* has an indicator parameter, then

   Case:

   a) If the declared type of *T* is character string or binary string and the length *M* in characters or octets, respectively, of *V* is greater than the length in characters or octets, respectively, of *T*, then the indicator parameter is set to *M*. If *M* exceeds the maximum value that the indicator parameter can contain, then an exception condition is raised: *data exception — indicator overflow.*

   b) Otherwise, the indicator parameter is set to 0 (zero).

5) If *V* is not the null value, *T* is a host variable, and *T* has an indicator variable, then

   Case:

   a) If the declared type of *T* is character string or binary string and the length in characters or octets, respectively, *M* of *V* is greater than the length in characters or octets, respectively, of *T*, then the indicator parameter is set to *M*. If *M* exceeds the maximum value that the indicator parameter can contain, then an exception condition is raised: *data exception — indicator overflow.*

b) Otherwise, the indicator variable is set to 0 (zero).

6) If *V* is not the null value, then

Case:

a) If the declared type of *T* is fixed-length character string with length in characters *L* and the length in characters of *V* is equal to *L*, then the value of *T* is set to *V*.

b) If the declared type of *T* is fixed-length character string with length in characters *L*, and the length in characters of *V* is greater than *L*, then the value of *T* is set to the first *L* characters of *V* and a completion condition is raised: *warning — string data, right truncation*.

c) If the declared type of *T* is fixed-length character string with length in characters *L*, and the length in characters *M* of *V* is smaller than *L*, then the first *M* characters of *T* are set to *V*, and the last *L–M* characters of *T* are set to <space>s.

d) If the declared type of *T* is variable-length character string and the length in characters *M* of *V* is not greater than the maximum length in characters of *T*, then the value of *T* is set to *V* and the length in characters of *T* is set to *M*.

e) If the declared type of *T* is variable-length character string and the length in characters of *V* is greater than the maximum length in characters *L* of *T*, then the value of *T* is set to the first *L* characters of *V*, then the length in characters of *T* becomes *L*, and a completion condition is raised: *warning — string data, right truncation*.

f) If the declared type of *T* is a character large object type and the length in characters *M* of *V* is not greater than the maximum length in characters of *T*, then the value of *T* is set to *V* and the length in characters of *T* is set to *M*.

g) If the declared type of *T* is a character large object type and the length in characters of *V* is greater than the maximum length in characters *L* of *T*, then the value of *T* is set to the first *L* characters of *V*, the length in characters of *T* becomes *L*, and a completion condition is raised: *warning — string data, right truncation*.

h) If the declared type of *T* is binary string and the length in octets *M* of *V* is not greater than the maximum length in octets of *T*, then the value of *T* is set to *V* and the length in octets of *T* is set to *M*.

i) If the declared type of *T* is binary string and the length in octets of *V* is greater than the maximum length in octets *L* of *T*, then the value of *T* is set to the first *L* octets of *V*, the length in octets of *T* becomes *L*, and a completion condition is raised: *warning — string data, right truncation*.

j) If the declared type of *T* is numeric, then

Case:

i) If *V* is a member of the declared type of *T*, then *T* is set to *V*.

ii) If a member of the declared type of *T* can be obtained from *V* by rounding or truncation, then *T* is set to that value. If the declared type of *T* is exact numeric, then it is implementation-defined whether the approximation is obtained by rounding or by truncation.

iii) Otherwise, an exception condition is raised: *data exception — numeric value out of range*.

k) If the declared type of *T* is boolean, then the value of *T* is set to *V*.

l) If the declared type *DT* of *T* is datetime, then:

   i) If only one of *DT* and the declared type of *V* is datetime with time zone, then *V* is effectively replaced by

   ```
   CAST ( V AS DT )
   ```

   ii) Case:

      1) If *V* is a member of the declared type of *T*, then *T* is set to *V*.

      2) If a member of the declared type of *T* can be obtained from *V* by rounding or truncation, then *T* is set to that value. It is implementation-defined whether the approximation is obtained by rounding or truncation.

      3) Otherwise, an exception condition is raised: *data exception — datetime field overflow.*

m) If the declared type of *T* is interval, then

   Case:

   i) If *V* is a member of the declared type of *T*, then *T* is set to *V*.

   ii) If a member of the declared type of *T* can be obtained from *V* by rounding or truncation, then *T* is set to that value. It is implementation-defined whether the approximation is obtained by rounding or by truncation.

   iii) Otherwise, an exception condition is raised: *data exception — interval field overflow.*

n) If the declared type of *T* is a row type, then:

   i) Let *n* be the degree of *T*.

   ii) For *i* ranging from 1 (one) to *n*, the General Rules of this Subclause are applied to the *i*-th element of *T* and the *i*-th element of *V* as *TARGET* and *VALUE*, respectively.

o) If the declared type of *T* is a user-defined type, then the value of *T* is set to *V*.

p) If the declared type of *T* is a reference type, then the value of *T* is set to *V*.

q) If the declared type of *T* is an array type, then

   Case:

   i) If the maximum cardinality *L* of *T* is equal to the cardinality *M* of *V*, then the elements of *T* are set to the values of the corresponding elements of *V* by applying the General Rules of this Subclause to each pair of elements with the element of *T* as *TARGET* and the element of *V* as *VALUE*.

   ii) If the maximum cardinality *L* of *T* is smaller than the cardinality *M* of *V*, then the elements of *T* are set to the values of the first *L* corresponding elements of *V* by applying the General Rules of this Subclause to each pair of elements with the element of *T* as *TARGET* and the element of *V* as *VALUE*; a completion condition is raised: *warning — array data, right truncation.*

   iii) If the maximum cardinality *L* of *T* is greater than the cardinality *M* of *V*, then the *M* first elements of *T* are set to the values of the corresponding elements of *V* by applying the General Rules of this Subclause to each pair of elements with the element of *T* as *TARGET* and the element of *V* as *VALUE*. The cardinality of the value of *T* is *M*.

NOTE 203 — The maximum cardinality *L* of *T* is unchanged.

r) If the declared type of *T* is a multiset type, then the value of *T* is set to *V*.

Conformance Rules

None.

9.2 Store assignment

Function

Specify rules for assignments where the target permits null without the use of indicator parameters or indicator variables, such as storing SQL-data or setting the value of SQL parameters.

Syntax Rules

1) Let *T* and *V* be the *TARGET* and *VALUE* specified in an application of this Subclause. Let the declared types of *T* and *V* be *TD* and *SD*, respectively.

2) If *TD* is character string, binary string, numeric, boolean, datetime, interval, or a user-defined type, then either *SD* shall be assignable to *TD* or there shall exist an appropriate user-defined cast function *UDCF* from *SD* to *TD*.

   NOTE 204 — "Appropriate user-defined cast function" is defined in Subclause 4.11, "Data conversions".

3) If the declared type of *T* is a reference type, then the declared type of *V* shall be a reference type whose referenced type is a subtype of the referenced type of *T*.

4) If the declared type of *T* is a row type, then:

   a) The declared type of *V* shall be a row type.

   b) The degree of *V* shall be the same as the degree of *T*. Let *n* be that degree.

   c) Let TT_i, 1 (one) $\leq i \leq n$, be the declared type of the *i*-th field of *T*, let VT_i be the declared type of the *i*-th field of *V*, let $T1_i$ be an arbitrary target whose declared type is TT_i, and let $V1_i$ be an arbitrary expression whose declared type is VT_i. For each *i*, 1 (one) $\leq i \leq n$, the Syntax Rules of this Subclause apply to T_i V_i, as *TARGET* and *VALUE*, respectively.

5) If the declared type of *T* is a collection type, then:

   a) If the declared type of *T* is an array type, then the declared type of *V* shall be an array type.

   b) If the declared type of *T* is a multiset type, then the declared type of *V* shall be a multiset type.

   c) Let *TT* be the element type of the declared type of *T*, let *VT* be the element type of the declared type of *V*, let *T1* be an arbitrary target whose declared type is *TT*, and let *V1* be an arbitrary expression whose declared type is *VT*. The Syntax Rules of this Subclause apply to *T1* and *V1*, as *TARGET* and *VALUE*, respectively.

Access Rules

None.

General Rules

1) If the declared type of *V* is not assignable to the declared type of *T*, then for the remaining General Rules of this Subclause *V* is effectively replaced by the result of evaluating the expression *UDCF*(*V*).

2) Case:

   a) If *V* is the null value, then

      Case:

      i) If *V* is specified using NULL, then *T* is set to the null value.

      ii) If *V* is a host parameter and contains an indicator parameter, then

         Case:

         1) If the value of the indicator parameter is equal to –1, then *T* is set to the null value.

         2) If the value of the indicator parameter is less than –1, then an exception condition is raised: *data exception — invalid indicator parameter value*.

      iii) If *V* is a host variable and contains an indicator variable, then

         Case:

         1) If the value of the indicator variable is equal to –1, then *T* is set to the null value.

         2) If the value of the indicator variable is less than –1, then an exception condition is raised: *data exception — invalid indicator parameter value*.

      iv) Otherwise, *T* is set to the null value.

   b) Otherwise,

      Case:

      i) If the declared type of *T* is fixed-length character string with length in characters *L* and the length in characters of *V* is equal to *L*, then the value of *T* is set to *V*.

      ii) If the declared type of *T* is fixed-length character string with length in characters *L* and the length in characters *M* of *V* is larger than *L*, then

         Case:

         1) If the rightmost *M–L* characters of *V* are all <space>s, then the value of *T* is set to the first *L* characters of *V*.

         2) If one or more of the rightmost *M–L* characters of *V* are not <space>s, then an exception condition is raised: *data exception — string data, right truncation*.

      iii) If the declared type of *T* is fixed-length character string with length in characters *L* and the length in characters *M* of *V* is less than *L*, then the first *M* characters of *T* are set to *V* and the last *L–M* characters of *T* are set to <space>s.

iv) If the declared type of *T* is variable-length character string and the length in characters *M* of *V* is not greater than the maximum length in characters of *T*, then the value of *T* is set to *V* and the length in characters of *T* is set to *M*.

v) If the declared type of *T* is variable-length character string and the length in characters *M* of *V* is greater than the maximum length in characters *L* of *T*, then

Case:

1) If the rightmost *M–L* characters of *V* are all <space>s, then the value of *T* is set to the first *L* characters of *V* and the length in characters of *T* is set to *L*.

2) If one or more of the rightmost *M–L* characters of *V* are not <space>s, then an exception condition is raised: *data exception — string data, right truncation*.

vi) If the declared type of *T* is a character large object type and the length in characters *M* of *V* is not greater than the maximum length in characters of *T*, then the value of *T* is set to *V* and the length in characters of *T* is set to *M*.

vii) If the declared type of *T* is a character large object type and the length in characters *M* of *V* is greater than the maximum length in characters *L* of *T*, then

Case:

1) If the rightmost *M–L* characters of *V* are all <space>s, then the value of *T* is set to the first *L* characters of *V* and the length in characters of *T* is set to *L*.

2) If one or more of the rightmost *M–L* characters of *V* are not <space>s, then an exception condition is raised: *data exception — string data, right truncation*.

viii) If the declared type of *T* is binary string and the length in octets *M* of *V* is not greater than the maximum length in octets of *T*, then the value of *T* is set to *V* and the length in octets of *T* is set to *M*.

ix) If the declared type of *T* is binary string and the length in octets *M* of *V* is greater than the maximum length in octets *L* of *T*, then

Case:

1) If the rightmost *M–L* octets of *V* are all equal to X'00', then the value of *T* is set to the first *L* octets of *V* and the length in octets of *T* is set to *L*.

2) If one or more of the rightmost *M–L* octets of *V* are not equal to X'00', then an exception condition is raised: *data exception — string data, right truncation*.

x) If the declared type of *T* is numeric, then

Case:

1) If *V* is a member of the declared type of *T*, then *T* is set to *V*.

2) If a member of the declared type of *T* can be obtained from *V* by rounding or truncation, then *T* is set to that value. If the declared type of *T* is exact numeric, then it is implementation-defined whether the approximation is obtained by rounding or by truncation.

3) Otherwise, an exception condition is raised: *data exception — numeric value out of range*.

xi) If the declared type *DT* of *T* is datetime, then

1) If only one of *DT* and the declared type of *V* is datetime with time zone, then *V* is effectively replaced by

```
CAST ( V AS DT )
```

2) Case:

A) If *V* is a member of the declared type of *T*, then *T* is set to *V*.

B) If a member of the declared type of *T* can be obtained from *V* by rounding or truncation, then *T* is set to that value. It is implementation-defined whether the approximation is obtained by rounding or truncation.

C) Otherwise, an exception condition is raised: *data exception — datetime field overflow*.

xii) If the declared type of *T* is interval, then

Case:

1) If *V* is a member of the declared type of *T*, then *T* is set to *V*.

2) If a member of the declared type of *T* can be obtained from *V* by rounding or truncation, then *T* is set to that value. It is implementation-defined whether the approximation is obtained by rounding or by truncation.

3) Otherwise, an exception condition is raised: *data exception — interval field overflow*.

xiii) If the declared type of *T* is boolean, then the value of *T* is set to *V*.

xiv) If the declared type of *T* is a row type, then:

1) Let *n* be the degree of *T*.

2) For *i* ranging from 1 (one) to *n*, the General Rules of this Subclause are applied to the *i*-th element of *T* and the *i*-th element of *V* as *TARGET* and *VALUE*, respectively.

xv) If the declared type of *T* is a user-defined type, then the value of *T* is set to *V*.

xvi) If the declared type of *T* is a reference type, then the value of *T* is set to *V*.

xvii) If the declared type of *T* is an array type, then

Case:

1) If the maximum cardinality *L* of *T* is equal to the cardinality *M* of *V*, then the elements of *T* are set to the values of the corresponding elements of *V* by applying the General Rules of this Subclause to each pair of elements with the element of *T* as *TARGET* and the element of *V* as *VALUE*.

2) If the maximum cardinality *L* of *T* is smaller than the cardinality *M* of *V*, then

Case:

A) If the rightmost *M–L* elements of *V* are all null, then the elements of *T* are set to the values of the first *L* corresponding elements of *V* by applying the General Rules of this

Subclause to each pair of elements with the element of *T* as *TARGET* and the element of *V* as *VALUE*.

B) If one or more of the rightmost *M*–*L* elements of *V* are not null, then an exception condition is raised: *data exception — array data, right truncation*.

3) If the maximum cardinality *L* of *T* is greater than the cardinality *M* of *V*, then the *M* first elements of *T* are set to the values of the corresponding elements of *V* by applying the General Rules of this Subclause to each pair of elements with the element of *T* as *TARGET* and the element of *V* as *VALUE*. The cardinality of the value of *T* is set to *M*.

NOTE 205 — The maximum cardinality *L* of *T* is unchanged.

xviii) If the declared type of *T* is a multiset type, then the value of *T* is set to *V*.

Conformance Rules

None.

9.3 Data types of results of aggregations

Function

Specify the result data type of the result of an aggregation over values of compatible data types, such as <case expression>s, <collection value expression>s, or a column in the result of a <query expression>.

Syntax Rules

1) Let *IDTS* be the set of data types specified in an application of this Subclause. Let *DTS* be the set of data types in *IDTS* excluding any data types that are undefined. If the cardinality of *DTS* is 0 (zero), then the result data type is undefined and no further Rules of this Subclause are evaluated.

   NOTE 206 — The notion of "undefined data type" is defined in Subclause 19.6, "<prepare statement>".

2) All of the data types in *DTS* shall be comparable.

3) Case:

   a) If any of the data types in *DTS* is character string, then:

      i) All data types in *DTS* shall be character string, and all of them shall have the same character repertoire. The character set of the result is the character set of the data type in *DTS* that has the character encoding form with the higest precedence.

      ii) The collation derivation and declared type collation of the result are determined as follows.

         Case:

         1) If some data type in *DTS* has an *explicit* collation derivation and declared type collation *EC1*, then every data type in *DTS* that has an *explicit* collation derivation shall have a declared type collation that is *EC1*. The collation derivation is *explicit* and the collation is *EC1*.

         2) If every data type in *DTS* has an *implicit* collation derivation, then

            Case:

            A) If every data type in *DTS* has the same declared type collation *IC1*, then the collation derivation is *implicit* and the declared type collation is *IC1*.

            B) Otherwise, the collation derivation is *none*.

         3) Otherwise, the collation derivation is *none*.

      iii) Case:

         1) If any of the data types in *DTS* is a character large object type, then the result data type is a character large object type with maximum length in characters equal to the maximum of the lengths in characters and maximum lengths in characters of the data types in *DTS*.

         2) If any of the data types in *DTS* is variable-length character string, then the result data type is variable-length character string with maximum length in characters equal to the maximum of the lengths in characters and maximum lengths in characters of the data types in *DTS*.

3) Otherwise, the result data type is fixed-length character string with length in characters equal to the maximum of the lengths in characters of the data types in *DTS*.

b) If any of the data types in *DTS* is binary string, then the result data type is binary string with maximum length in octets equal to the maximum of the lengths in octets and maximum lengths in octets of the data types in *DTS*.

c) If all of the data types in *DTS* are exact numeric, then the result data type is exact numeric with implementation-defined precision and with scale equal to the maximum of the scales of the data types in *DTS*.

d) If any data type in *DTS* is approximate numeric, then each data type in *DTS* shall be numeric and the result data type is approximate numeric with implementation-defined precision.

e) If some data type in *DTS* is a datetime data type, then every data type in *DTS* shall be a datetime data type having the same datetime fields. The result data type is a datetime data type having the same datetime fields, whose fractional seconds precision is the largest of the fractional seconds precisions in *DTS*. If some data type in *DTS* has a time zone displacement value, then the result has a time zone displacement value; otherwise, the result does not have a time zone displacement value.

f) If any data type in *DTS* is interval, then each data type in *DTS* shall be interval. If the precision of any data type in *DTS* specifies YEAR or MONTH, then the precision of each data type shall specify only YEAR or MONTH. If the precision of any data type in *DTS* specifies DAY, HOUR, MINUTE, or SECOND(*N*), then the precision of no data type of *DTS* shall specify the <primary datetime field>s YEAR and MONTH. The result data type is interval with precision "*S* TO *E*", where *S* and *E* are the most significant of the <start field>s and the least significant of the <end field>s of the data types in *DTS*, respectively.

g) If any data type in *DTS* is boolean, then each data type in *DTS* shall be boolean. The result data type is boolean.

h) If any data type in *DTS* is a row type, then each data type in *DTS* shall be a row type with the same degree and the data type of each field in the same ordinal position of every row type shall be comparable. The result data type is a row type defined by an ordered sequence of (<field name>, data type) pairs FD_i, where data type is the data type resulting from the application of this Subclause to the set of data types of fields in the same ordinal position as FD_i in every row type in *DTS* and <field name> is determined as follows:

   Case:

   i) If the names of fields in the same ordinal position as FD_i in every row type in *DTS* is *F*, then the <field name> in FD_i is *F*.

   ii) Otherwise, the <field name> in FD_i is implementation-dependent.

i) If any data type in *DTS* is an array type then every data type in *DTS* shall be an array type. The data type of the result is array type with element data type *ETR*, where *ETR* is the data type resulting from the application of this Subclause to the set of element types of the array types of *DTS*, and maximum cardinality equal to the maximum of the maximum cardinalities of the data types in *DTS*.

j) If any data type in *DTS* is a multiset type then every data type in *DTS* shall be a multiset type. The data type of the result is multiset type with element data type *ETR*, where *ETR* is the data type resulting from the application of this Subclause to the set of element types of the multiset types of *DTS*.

k) If any data type in *DTS* is a reference type, then there shall exist a subtype family *STF* such that each data type in *DTS* is a member of *STF*. Let *RT* be the minimal common supertype of each data type in *DTS*.

   Case:

   i) If the data type descriptor of every data type in *DTS* includes the name of a referenceable table identifying the scope of the reference type, and every such name is equivalent to some name *STN*, then result data type is:

   ```
   RT SCOPE ( STN )
   ```

   ii) Otherwise, the result data type is *RT*.

l) Otherwise, there shall exist a subtype family *STF* such that each data type in *DTS* is a member of *STF*. The result data type is the minimal common supertype of each data type in *DTS*.

   NOTE 207 — *Minimal common supertype* is defined in Subclause 4.7.5, "Subtypes and supertypes".

Access Rules

None.

General Rules

None.

Conformance Rules

None.

9.4 Subject routine determination

Function

Determine the subject routine of a given routine invocation.

Syntax Rules

1) Let *SR* and *AL* be respectively the set of SQL-invoked routines, arbitrarily ordered, and the <SQL argument list> specified in an application of this Subclause.

2) Let n be the number of SQL-invoked routines in *SR*. Let R_i, 1 (one) $\leq i \leq n$, be the i-th SQL-invoked routine in *SR* in the ordering of *SR*.

3) Let m be the number of SQL arguments in *AL*. Let A_j, 1 (one) $\leq j \leq m$, be the j-th SQL argument in *AL*.

4) For A_j, 1 (one) $\leq j \leq m$, then let $SDTA_j$ be the declared type of A_j.

5) Let $SDTP_{i,j}$ be the type designator of the declared type of the j-th SQL parameter of R_i.

6) For r varying from 1 (one) to m, if A_r is not a <dynamic parameter specification> and if there is more than one SQL-invoked routine in *SR*, then for each pair of SQL-invoked routines $\{ R_p, R_q \}$ in *SR*, if $SDTP_{p,r} \prec SDTP_{q,r}$ in the type precedence list of $SDTA_r$, then eliminate R_q from *SR*.

NOTE 208 — The "type precedence list" of a given type is determined by Subclause 9.5, "Type precedence list determination".

7) The set of subject routines is the set of SQL-invoked routines remaining in *SR*.

Access Rules

None.

General Rules

None.

Conformance Rules

None.

9.5 Type precedence list determination

Function

Determine the type precedence list of a given type.

Syntax Rules

1) Let *DT* be the data type specified in an application of this Subclause.

2) The *type precedence list TPL* of *DT* is a list of *type designators* as specified in the Syntax Rules of this Subclause.

3) Let "$A \prec B$" represent "*A* has precedence over *B*" and let "$A \simeq B$" represent "*A* has the same precedence as *B*".

4) If *DT* is a user-defined type, then:

   a) Let *ST* be the set of supertypes of *DT*. Let *n* be the number of data types in *ST*.

   b) For any two data types *TA* and *TB* in *ST*, $TA \prec TB$ if and only if *TA* is a proper subtype of *TB*.

   c) Let T_1 be *DT* and let T_{i+1}, 1 (one) $\leq i \leq n-1$, be the direct supertype of T_i.

   d) Let DTN_i, 1 (one) $\leq i \leq n$, be the data type designator of T_i.

   NOTE 209 — The type designator of a user-defined type is the type name included in its user-defined type descriptor.

   e) *TPL* is DTN_1, DTN_2, ..., DTN_n.

5) If *DT* is fixed-length character string, then *TPL* is

```
CHARACTER, CHARACTER VARYING, CHARACTER LARGE OBJECT
```

6) If *DT* is variable-length character string, then *TPL* is

```
CHARACTER VARYING, CHARACTER LARGE OBJECT
```

7) If *DT* is binary string, then *TPL* is

```
BINARY LARGE OBJECT
```

8) If *DT* is numeric, then:

   a) Let *NDT* be the following set of numeric types: NUMERIC, DECIMAL, SMALLINT, INTEGER, BIGINT, REAL, FLOAT, and DOUBLE PRECISION. For each type *T* in *NDT*, the *effective binary precision* is defined as follows.

   Case:

   i) If *T* is DECIMAL or NUMERIC, then the effective binary precision is the product of $\log_2(10)$ and the implementation-defined maximum precision of *T*.

ii) If *T* is FLOAT, then the effective binary precision is the implementation-defined maximum precision of *T*.

iii) If the radix of *T* is decimal, then the effective binary precision is the product of $\log_2(10)$ and the implementation-defined precision of *T*.

iv) Otherwise, the effective binary precision is the implementation-defined precision of *T*.

b) Let *PTC* be the set of all precedence relationships determined as follows: For any two types *T1* and *T2*, not necessarily distinct, in *NDT*,

Case:

i) If *T1* is exact numeric and *T2* is approximate numeric, then $T1 \prec T2$.

ii) If *T1* is approximate numeric and *T2* is exact numeric, then $T1 \succ T2$.

iii) If the effective binary precision of *T1* is greater than the effective binary precision of *T2*, then $T2 \prec T1$.

iv) If the effective binary precision of *T1* equals the effective binary precision of *T2*, then $T2 \simeq T1$.

v) Otherwise, $T1 \prec T2$.

c) *TPL* is determined as follows:

i) *TPL* is initially empty.

ii) Let *ST* be the set of types containing *DT* and every type *T* in *NDT* for which the precedence relationship $DT \prec T$ or $DT \simeq T$ is in *PTC*.

iii) Let *n* be the number of types in *ST*.

iv) For *i* ranging from 1 (one) to *n*:

1) Let *NT* be the set of types T_k in *ST* such that there is no other type T_j in *ST* for which $T_j \prec T_k$ according to *PTC*.

2) Case:

A) If there is exactly one type T_k in *NT*, then T_k is placed next in *TPL* and all relationships of the form $T_k \prec T_r$ are removed from *PTC*, where T_r is any type in *ST*.

B) If there is more than one type T_k in *NT*, then every type T_s in *NT* is assigned the same position in *TPL* as T_k and all relationships of the forms $T_k \prec T_r$, $T_k \simeq T_r$, $T_s \prec T_r$, and $T_s \simeq T_r$ are removed from *PTC*, where T_r is any type in *ST*.

9) If *DT* specifies a year-month interval type, then *TPL* is

```
INTERVAL YEAR
```

10) If *DT* specifies a day-time interval type, then *TPL* is

```
INTERVAL DAY
```

11) If *DT* specifies DATE, then *TPL* is

```
DATE
```

12) If *DT* specifies TIME, then *TPL* is

```
TIME
```

13) If *DT* specifies TIMESTAMP, then *TPL* is

```
TIMESTAMP
```

14) If *DT* specifies BOOLEAN, then *TPL* is

```
BOOLEAN
```

15) If *DT* is a collection type, then let *CTC* be the kind of collection (either ARRAY or MULTISET) specified in *DT*.

    Let *n* be the number of elements in the type precedence list for the element type of *DT*. For *i* ranging from 1 (one) to *n*, let RIO_i be the *i*-th such element. *TPL* is

    RIO_1 *CTC*, RIO_2 *CTC*, . . . , RIO_n *CTC*

16) If *DT* is a reference type, then let *n* be the number of elements in the type precedence list for the referenced type of *DT*. For *i* ranging from 1 (one) to *n*, let KAW_i be the *i*-th such element. *TPL* is

    REF(KAW_1), REF(KAW_2), . . . , REF(KAW_n)

17) If *DT* is a row type, then *TPL* is

```
ROW
```

NOTE 210 — This rule is placed only to avoid the confusion that might arise if row types were not mentioned in this Subclause. As a row type cannot be used as a <parameter type>, the type precedence list of a row type is never referenced.

Conformance Rules

None.

9.6 Host parameter mode determination

Function

Determine the parameter mode for a given host parameter.

Syntax Rules

1) Let *PD* and *SPS* be a <host parameter declaration> and an <SQL procedure statement> specified in an application of this Subclause.

2) Let *P* be the host parameter specified by *PD* and let *PN* be the <host parameter name> immediately contained in *PD*.

3) Whether *P* is an input host parameter, an output host parameter, or both an input host parameter and an output host parameter is determined as follows:

   Case:

   a) If *PD* is a <status parameter>, then *P* is an output host parameter.

   b) Otherwise,

      Case:

      i) If *PN* is contained in an <SQL argument> A_i of the <SQL argument list> of a <routine invocation> immediately contained in a <call statement> that is contained in *SPS*, then:

         1) Let *R* be the subject routine of the <routine invocation>.

         2) Let PR_i be the *i*-th SQL parameter of *R*.

         3) Case:

            A) If *PN* is contained in a <host parameter specification> that is the <target specification> that is simply contained in A_i and PR_i is an output SQL parameter, then *P* is an output host parameter.

            B) If *PN* is contained in a <host parameter specification> that is the <target specification> that is simply contained in A_i and PR_i is both an input SQL parameter and an output SQL parameter, then *P* is both an input host parameter and an output host parameter.

            C) Otherwise, *P* is an input host parameter.

      ii) If *PN* is contained in a <value specification> or a <simple value specification> that is contained in *SPS*, and *PN* is not contained in a <target specification> or a <simple target specification> that is contained in *SPS*, then *P* is an input host parameter.

      iii) If *PN* is contained in a <target specification> or a <simple target specification> that is contained in *SPS*, and *PN* is not contained in a <value specification> or a <simple value specification> that is contained in *SPS*, then *P* is an output host parameter.

iv) If *PN* is contained in a <value specification> or a <simple value specification> that is contained in *SPS*, and in a <target specification> or a <simple target specification> that is contained in *SPS*, then *P* is both an input host parameter and an output host parameter.

v) Otherwise, *P* is neither an input host parameter nor an output host parameter.

Access Rules

None.

General Rules

None.

Conformance Rules

None.

9.7 Type name determination

Function

Determine an <identifier> given the name of a predefined data type.

Syntax Rules

1) Let *DT* be the <predefined type> specified in an application of this Subclause.
2) Let *FNSDT* be the <identifier> resulting from an application of this Subclause, defined as follows.

   Case:

   a) If *DT* specifies CHARACTER, then let *FNSDT* be "CHAR".
   b) If *DT* specifies CHARACTER VARYING, then let *FNSDT* be "VARCHAR".
   c) If *DT* specifies CHARACTER LARGE OBJECT, then let *FNSDT* be "CLOB".
   d) If *DT* specifies BINARY LARGE OBJECT, then let *FNSDT* be "BLOB".
   e) If *DT* specifies SMALLINT, then let *FNSDT* be "SMALLINT".
   f) If *DT* specifies INTEGER, then let *FNSDT* be "INTEGER".
   g) If *DT* specifies BIGINT, then let *FNSDT* be "BIGINT".
   h) If *DT* specifies DECIMAL, then let *FNSDT* be "DECIMAL".
   i) If *DT* specifies NUMERIC, then let *FNSDT* be "NUMERIC".
   j) If *DT* specifies REAL, then let *FNSDT* be "REAL".
   k) If *DT* specifies FLOAT, then let *FNSDT* be "FLOAT".
   l) If *DT* specifies DOUBLE PRECISION, then let *FNSDT* be "DOUBLE".
   m) If *DT* specifies DATE, then let *FNSDT* be "DATE".
   n) If *DT* specifies TIME, then let *FNSDT* be "TIME".
   o) If *DT* specifies TIMESTAMP, then let *FNSDT* be "TIMESTAMP".
   p) If *DT* specifies INTERVAL, then let *FNSDT* be "INTERVAL".

Access Rules

None.

General Rules

None.

Conformance Rules

None.

9.8 Determination of identical values

Function

Determine whether two instances of values are identical, that is to say, are occurrences of the same value.

Syntax Rules

None.

Access Rules

None.

General Rules

1) Let *V1* and *V2* be two values specified in an application of this Subclause.

   NOTE 211 — This Subclause is invoked implicitly wherever the word *identical* is used of two values.

2) Case:

   a) If *V1* and *V2* are both null, then *V1* is identical to *V2*.

   b) If *V1* is null and *V2* is not null, or if *V1* is not null and *V2* is null, then *V1* is not identical to *V2*.

   c) If *V1* and *V2* are of comparable predefined types, then

      Case:

      i) If *V1* and *V2* are character strings, then let *L* be CHARACTER_LENGTH(*V1*).

         Case:

         1) If CHARACTER_LENGTH(*V2*) equals *L*, and if for all *i*, 1 (one) ≤ *i* ≤ *L*, the *i*-th character of *V1* corresponds to the same character position of ISO/IEC 10646 as the *i*-th character of *V2*, then *V1* is identical to *V2*.

         2) Otherwise, *V1* is not identical to *V2*.

      ii) If *V1* and *V2* are TIME WITH TIME ZONE or TIMESTAMP WITH TIME ZONE and are not distinct, and their time zone displacement fields are not distinct, then *V1* is identical to *V2*.

      iii) Otherwise, *V1* is identical to *V2* if and only if *V1* is not distinct from *V2*.

   d) If *V1* and *V2* are of constructed types, then

      Case:

      i) If *V1* and *V2* are rows and their respective fields are identical, then *V1* is identical to *V2*.

ii) If *V1* and *V2* are arrays and have the same cardinality and elements in the same ordinal position in the two arrays are identical, then *V1* is identical to *V2*.

iii) If *V1* and *V2* are multisets and have the same cardinality *N* and there exist enumerations $VE1_i$, 1 (one) $\leq i \leq N$, of *V1* and $VE2_i$, 1 (one) $\leq i \leq N$, of *V2* such that for all *i*, $VE1_i$ is identical to $VE2_i$, then *V1* is identical to *V2*.

iv) If *V1* and *V2* are references and *V1* is not distinct from *V2*, then *V1* is identical to *V2*.

v) Otherwise, *V1* is not identical to *V2*.

e) If *V1* and *V2* are of the same most specific type *MST* and *MST* is a user-defined type, then

Case:

i) If *MST* is a distinct type whose source type is *SDT* and the results of `SDT(V1)` and `SDT(V2)` are identical, then *V1* is identical to *V2*.

ii) If *MST* is a structured type and, for every observer function *O* defined for *MST*, the results of the invocations *O*(*V1*) and *O*(*V2*) are identical, then *V1* is identical to *V2*.

iii) Otherwise, *V1* is not identical to *V2*.

f) Otherwise, *V1* is not identical to *V2*.

Conformance Rules

None.

9.9 Equality operations

Function

Specify the prohibitions and restrictions by data type on operations that involve testing for equality.

Syntax Rules

1) An *equality operation* is any of the following:
   a) A <comparison predicate> that specifies <equals operator> or <not equals operator>.
   b) A <quantified comparison predicate> that specifies <equals operator> or <not equals operator>.
   c) An <in predicate>.
   d) A <like predicate>.
   e) A <similar predicate>.
   f) A <distinct predicate>.
   g) A <match predicate>.
   h) A <member predicate>.
   i) A <joined table> that specifies NATURAL or USING.
   j) A <user-defined ordering definition> that specifies MAP.
   k) A <position expression>.
2) An *operand of an equality operation* is any of the following:
   a) A field of the declared row type of a <row value predicand> that is simply contained in a <comparison predicate> that specifies <equals operator> or <not equals operator>.
   b) A field of the declared row type of a <row value predicand> that is simply contained in a <quantified comparison predicate> that specifies <equals operator> or <not equals operator>.
   c) A column of a <table subquery> that is simply contained in a <quantified comparison predicate> that specifies <equals operator> or <not equals operator>.
   d) A field of the declared row type of a <row value predicand> or <row value expression> that is simply contained in an <in predicate>.
   e) A column of a <table subquery> that is simply contained in an <in predicate>.
   f) A field of the declared row type of a <row value predicand> that is simply contained in a <like predicate>.
   g) A <character pattern>, <escape character>, <octet pattern> or <escape octet> that is simply contained in a <like predicate>.

h) A field of the declared row type of a <row value predicand> that is simply contained in a <similar predicate>.

i) A <similar pattern> or <escape character> that is simply contained in a <similar predicate>.

j) A field of the declared row type of a <row value predicand> that is simply contained in a <distinct predicate>.

k) A field of the declared row type of a <row value predicand> that is simply contained in a <match predicate>.

l) A column of a <table subquery> that is simply contained in a <match predicate>.

m) A field of the declared row type of a <row value predicand> that is simply contained in a <member predicate>.

n) A corresponding join column of a <joined table> that specifies NATURAL or USING.

o) The <returns data type> of the SQL-invoked function identified by a <map function specification> simply contained in a <user-defined ordering definition> that specifies MAP.

p) A <string value expression> that is simply contained in a <position expression>.

q) A <blob value expression> that is simply contained in a <position expression>.

3) The declared type of an operand of an equality operation shall not be UDT-NC-ordered.

4) If the declared type of the operands of an equality operation is a character string type, then the Syntax Rules of Subclause 9.13, “Collation determination”, apply.

5) If the declared type of an operand *OP* of an equality operation is a multiset type, then *OP* is a multiset operand of a multiset element grouping operation. The Syntax Rules of Subclause 9.11, “Multiset element grouping operations”, apply.

Access Rules

None.

General Rules

None.

Conformance Rules

1) Without Feature S024, “Enhanced structured types”, in conforming SQL language, the declared type of an operand of an equality operation shall not be ST-ordered.

2) Without Feature T042, “Extended LOB data type support”, in conforming SQL language, the declared type of an operand of an equality operation shall not be LOB-ordered.

3) Without Feature S275, “Advanced multiset support”, in conforming SQL language, the declared type of an operand of an equality operation shall not be multiset-ordered.

NOTE 212 — If the declared type of an operand *OP* of an equality operation is a multiset type, then *OP* is a multiset operand of a multiset element grouping operation. The Conformance Rules of Subclause 9.11, "Multiset element grouping operations", apply.

9.10 Grouping operations

Function

Specify the prohibitions and restrictions by data type on operations that involve grouping of data.

Syntax Rules

1) A *grouping operation* is any of the following:
   a) A <group by clause>.
   b) A <window partition clause>.
   c) An <aggregate function> that specifies DISTINCT.
   d) A <query specification> that immediately contains DISTINCT.
   e) A <query expression body> that simply contains or implies UNION DISTINCT.
   f) A <query expression body> that simply contains EXCEPT.
   g) A <query term> that simply contains INTERSECT.
   h) A <unique predicate>.
   i) A <unique constraint definition>.
   j) A <referential constraint definition>.
2) An *operand of a grouping operation* is any of the following:
   a) A grouping column of a <group by clause>.
   b) A partitioning column of a <window partition clause>.
   c) A <value expression> simply contained in an <aggregate function> that specifies DISTINCT.
   d) A column of the result of a <query specification> that immediately contains DISTINCT.
   e) A column of the result of a <query expression body> that simply contains or implies UNION DISTINCT.
   f) A column of the result of a <query expression body> that simply contains EXCEPT.
   g) A column of the result of a <query term> that simply contains INTERSECT.
   h) A column of the <table subquery> simply contained in a <unique predicate>.
   i) A column identified by the <unique column list> of a <unique constraint definition>.
   j) A referencing column of a <referential constraint definition>.
3) The declared type of an operand of a grouping operation shall not be LOB-ordered, array-ordered, multiset-ordered, UDT-EC-ordered, or UDT-NC-ordered.

4) If the declared type of an operand of a grouping operation is a character string type, then the Syntax Rules of Subclause 9.13, "Collation determination", apply.

Access Rules

None.

General Rules

None.

Conformance Rules

1) Without Feature S024, "Enhanced structured types", in conforming SQL language, the declared type of an operand of a grouping operation shall not be ST-ordered.

9.11 Multiset element grouping operations

Function

Specify the prohibitions and restrictions by data type on the declared element type of a multiset for operations that involve grouping the elements of a multiset.

Syntax Rules

1) A *multiset element grouping operation* is any of the following:
   a) An equality operation such that the declared type of an operand of the equality operation is a multiset type.
   b) A <multiset set function>.
   c) A <multiset value expression> that specifies MULTISET UNION DISTINCT.
   d) A <multiset value expression> that specifies MULTISET EXCEPT.
   e) A <multiset term> that specifies MULTISET INTERSECT.
   f) A <submultiset predicate>.
   g) A <set predicate>.
   h) A <general set function> that specifies INTERSECTION.
2) A *multiset operand* of a multiset element grouping operation is any of the following:
   a) A <multiset value expression> simply contained in a <multiset set function>.
   b) A <multiset value expression> or a <multiset term> that is simply contained in a <multiset value expression> that simply contains MULTISET UNION DISTINCT.
   c) A <multiset value expression> or a <multiset term> that is simply contained in a <multiset value expression> that simply contains MULTISET EXCEPT.
   d) A <multiset term> or a <multiset primary> that is simply contained in a <multiset term> that simply contains MULTISET INTERSECT.
   e) A field of a comparand of a <comparison predicate> such that the <comparison predicate> specifies <equals operator> or <not equals operator> and such that the declared type of the field is a multiset type.
   f) A <multiset value expression> simply contained in a <member predicate>.
   g) A field of the <row value expression> simply contained in a <submultiset predicate>.
   h) The <multiset value expression> simply contained in a <submultiset predicate>.
   i) A field of the <row value expression> simply contained in a <set predicate>.
   j) A value expression> simply contained in a <general set function> that specifies INTERSECTION.

3) The declared element type of a multiset operand of a multiset element grouping operation shall not be LOB-ordered, array-ordered, multiset-ordered, UDT-EC-ordered, or UDT-NC-ordered.

4) If the declared element type of a multiset operand of a multiset element grouping operation is a character string type, then the Syntax Rules of Subclause 9.13, “Collation determination”, apply.

Access Rules

None.

General Rules

None.

Conformance Rules

1) Without Feature S024, “Enhanced structured types”, in conforming SQL language, the declared element type of a multiset operand of a multiset element grouping operation shall not be ST-ordered.

9.12 Ordering operations

Function

Specify the prohibitions and restrictions by data type on operations that involve ordering of data.

Syntax Rules

1) An *ordering operation* is any of the following:
   a) A <comparison predicate> that does not specify <equals operator> or <not equals operator>.
   b) A <quantified comparison predicate> that does not specify <equals operator> or <not equals operator>.
   c) A <between predicate>.
   d) An <overlaps predicate>.
   e) An <aggregate function> that specifies MAX or MIN.
   f) A <sort specification list>.
   g) A <user-defined ordering definition> that specifies ORDER FULL BY MAP.
2) An *operand of an ordering operation* is any of the following:
   a) A field of the declared row type of a <row value predicand> that is simply contained in a <comparison predicate> that does not specify <equals operator> or <not equals operator>.
   b) A field of the declared row type of a <row value predicand> that is simply contained in a <quantified comparison predicate> that does not specify <equals operator> or <not equals operator>.
   c) A column of the <table subquery> that is simply contained in a <quantified comparison predicate> that does not specify <equals operator> or <not equals operator>.
   d) A field of the declared row type of a <row value predicand> that is simply contained in a <between predicate>.
   e) A field of the declared row type of a <row value predicand> that is simply contained in an <overlaps predicate>.
   f) A <value expression> simply contained in an <aggregate function> that specifies MAX or MIN.
   g) A <value expression> simply contained in a <sort key>.
   h) The <returns data type> of the SQL-invoked function identified by a <map function specification> simply contained in a <user-defined ordering definition> that specifies ORDER FULL BY MAP.
3) The declared type of an operand of an ordering operation shall not be LOB-ordered, array-ordered, multiset-ordered, reference-ordered, UDT-EC-ordered, or UDT-NC-ordered.
4) If the declared type of an operand of an ordering operation is a character string type, then the Syntax Rules of Subclause 9.13, "Collation determination", apply.

Access Rules

None.

General Rules

None.

Conformance Rules

1) Without Feature S024, “Enhanced structured types”, in conforming SQL language, the declared type of an operand of an ordering operation shall not be ST-ordered.

9.13 Collation determination

Function

Specify rules for determining the collation to be used in the comparison of character strings.

Syntax Rules

1) Let *CCS* be the character set of the result of applying the rules of Subclause 9.3, "Data types of results of aggregations", to the declared types of all operands of the comparison operation.
2) At least one operand shall have a declared type collation.
3) Case:
   a) If the comparison operation is a <referential constraint definition>, then, for each referencing column, the collation to be used is the declared type collation of the corresponding column of the referenced table.
   b) If at least one operand has an *explicit* collation derivation, then every operand whose collation derivation is *explicit* shall have the same declared type collation *EDTC* and the collation to be used is *EDTC*.
   c) If the comparison operation is contained in a <preparable statement> that is prepared in the current SQL-session by an <execute immediate statement> or a <prepare statement>, or in a <direct SQL statement> that is invoked directly, and *CCS* has an SQL-session collation, then the collation to be used is that SQL-session collation.
   d) If *CCS* has an SQL-client module collation, then the collation to be used is that collation.
   e) Otherwise, every operand whose collation derivation is *implicit* shall have the same declared type collation *IDTC* and the collation to be used is *IDTC*.

Access Rules

None.

General Rules

None.

Conformance Rules

None.

9.14 Execution of array-returning functions

Function

Define the execution of an external function that returns an array value.

Syntax Rules

None.

Access Rules

None.

General Rules

1) Let *AR*, *ESPL*, and *P* be the *ARRAY*, *EFFECTIVE SQL PARAMETER LIST*, and *PROGRAM* specified in an application of this Subclause.
2) Let *ARC* be the cardinality of *AR*.
3) Let *EN* be the number of entries in *ESPL*.
4) Let ESP_i, 1 (one) ≤ *i* ≤ *EN*, be the *i*-th parameter in *ESPL*.
5) Let *FRN* be the number of result data items.
6) Let *PN* and *N* be the number of values in the static SQL argument list of *P*.
7) Let *E* be 0 (zero).
8) If the call type data item has a value of –1 (indicating "open call"), then *P* is executed with a list of *EN* parameters PD_i whose parameter names are PN_i and whose values are set as follows:
   a) Depending on whether the language of *R* specifies ADA, C, COBOL, FORTRAN, M, PASCAL, or PLI, let the *operative data type correspondences* table be Table 16, "Data type correspondences for Ada", Table 17, "Data type correspondences for C", Table 18, "Data type correspondences for COBOL", Table 19, "Data type correspondences for Fortran", Table 20, "Data type correspondences for M", Table 21, "Data type correspondences for Pascal", or Table 22, "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.
   b) For *i* varying from 1 (one) to *EN*, the data type DT_i of PD_i is the data type listed in the host data type column of the row in the data type correspondences table whose value in the SQL data type column corresponds to the data type of ESP_i.
   c) The value of PD_i is set to the value of ESP_i.
9) Case:

a) If the value of the exception data item is '00000' (corresponding to the completion condition *successful completion*) or the first 2 characters are '01' (corresponding to the completion condition *warning* with any subcondition), then set the call type data item to 0 (zero) (indicating *fetch call*).

b) If the exception data item is '02000' (corresponding to the completion condition *no data*):

   i) If each PD_i, for i ranging from $(PN+FRN)+N+1$ through $(PN+FRN)+N+FRN$ (that is, the SQL indicator arguments corresponding to the result data items), has the value –1, then set *AR* to the null value.

   ii) Set the call type data item to 1 (one) (indicating *close call*).

c) Otherwise, set the call type data item to 1 (one) (indicating *close call*).

10) The following steps are applied as long as the call type data item has a value 0 (zero) (corresponding to *fetch call*):

a) *P* is executed with a list of *EN* parameters PD_i whose parameter names are PN_i and whose values are set as follows:

   i) For i varying from 1 (one) to *EN*, the <data type> DT_i of PD_i is the data type listed in the host data type column of the row in the data type correspondences table whose value in the SQL data type column corresponds to the data type of ESP_i.

   ii) For i ranging from 1 (one) to $EN–2$, the value of PD_i is set to the value of ESP_i.

   iii) For the save area data item, for i equal to $EN–1$, the value of PD_i is set to the value returned in PD_i by the prior execution of *P*.

   iv) For the call type data item, for i equal to *EN*, the value of PD_i is set to 0 (zero).

b) Case:

   i) If the exception data item is '00000' (corresponding to completion condition *successful completion*) or the first 2 characters are '01' (corresponding to completion condition *warning* with any subcondition), then:

      1) Increment *E* by 1 (one).

      2) If $E > ARC$, then an exception condition is raised: *data exception — array element error*.

      3) If the call type data item is 0 (zero), then

         Case:

         A) If each PD_i, for i ranging from $(PN+FRN)+N+1$ through $(PN+FRN)+N+FRN$ (that is, the SQL indicator arguments corresponding to the result data items) is negative, then let the *E*-th element of *AR* be the null value.

         B) Otherwise,

            Case:

            I) If *FRN* is 1 (one), then let the *E*-th element of *AR* be the value of the result data item.

II) Otherwise:

1) Let RDI_i, 1 (one) $\leq i \leq FRN$, be the value of the *i*-th result data item.

2) Let the *E*-th element of *AR* be the value of the following <row value expression>:

```
ROW ( RDI1, ... , RDIFRN )
```

ii) If the exception data item is '02000' (corresponding to completion condition *no data*), then:

1) If the value of *E* is 0 (zero), then set *AR* to an empty array.

2) Set the call type data item to 1 (one) (indicating *close call*).

iii) Otherwise, set the value of the call type data item to 1 (one) (indicating *close call*).

11) If the call type data item has a value of 1 (one) (indicating *close call*), then *P* is executed with a list of *EN* parameters PD_i whose parameter names are PN_i and whose values are set as follows:

a) For *i* varying from 1 (one) to *EN*, the <data type> DT_i of PD_i is the data type listed in the host data type column of the row in the data type correspondences table whose value in the SQL data type column corresponds to the data type of ESP_i.

b) For *i* ranging from 1 (one) to *EN*–2, the value of PD_i is set to the value of ESP_i.

c) For the save area data item, for *i* equal to *EN*–1, the value of PD_i is set to the value returned in PD_i by the prior execution of *P*.

d) For the call type data item, for *i* equal to *EN*, the value of PD_i is set to 1 (one).

Conformance Rules

None.

9.15 Execution of multiset-returning functions

Function

Define the execution of an external function that returns a multiset value.

Syntax Rules

None.

Access Rules

None.

General Rules

1) Let *MU*, *ESPL*, and *P* be the *MULTISET*, *EFFECTIVE SQL PARAMETER LIST*, and *PROGRAM* specified in an application of this subclause.
2) Let *ET* be the element type of *MU*.
3) Let *C* be the maximum implementation-defined cardinality of array type with element type *ET*.
4) Let *AT* be the array type *ET* ARRAY[*C*].
5) Let *AR* be an array whose declared type is *AT*.
6) The General Rules of Subclause 9.14, "Execution of array-returning functions", are applied with *AR*, *ESPL*, and *P* as *ARRAY*, *EFFECTIVE SQL PARAMETER LIST*, and *PROGRAM*, respectively.
7) Let *MU* be the result of casting *AR* to the multiset type of *MU* according to the General Rules of Subclause 6.12, "<cast specification>".

Conformance Rules

None.

9.16 Data type identity

Function

Determine whether two data types are compatible and have the same characteristics.

Syntax Rules

1) Let *PM* and *P* be the two data types specified in an application of this Subclause.

2) *PM* and *P* shall be compatible.

3) If *PM* is a character string type, then the length of *PM* shall be equal to the length of *P*.

4) If *PM* is an exact numeric type, then the precision and scale of *PM* shall be equal to the precision and scale of *P*, respectively.

5) If *PM* is an approximate numeric type, then the precision of *PM* shall be equal to the precision of *P*.

6) If *PM* is a binary string type, then the maximum length of *PM* shall be equal to the maximum length of *P*.

7) If *PM* is a datetime data type with <time fractional seconds precision>, then the <time fractional seconds precision> of *PM* shall be equal to the <time fractional seconds precision> of *P*.

8) If *PM* is an interval type, then the <interval qualifier> of *PM* shall be equivalent to the <interval qualifier> of *P*.

9) If *PM* is a collection type, then:

   a) The kind of collection (ARRAY or MULTISET) of *PM* and the kind of collection of *P* shall be the same.

   b) If *PM* is an array type, then the maximum cardinality of *PM* shall be equal to the the maximum cardinality of *P*.

   c) The Syntax Rules of this Subclause are applied with the element type of *PM* and the element type of *P* as the two data types.

10) If *PM* is a row type, then:

   a) Let *N* be the degree of *PM*.

   b) Let $DTFPM_i$ and $DTFP_i$, 1 (one) $\leq i \leq N$, be the data type of the *i*-th field of *PM* and of *P*, respectively. For *i* varying from 1 (one) to *N*, the Syntax Rules of this Subclause are applied with $DTFPM_i$ and $DTFP_i$ the two data types.

Access Rules

None.

General Rules

None.

Conformance Rules

None.

9.17 Determination of a from-sql function

Function

Determine the from-sql function of a user-defined type given the name of a user-defined type and the name of the group.

Syntax Rules

1) Let *UDT* and *GN* be a *TYPE* and a *GROUP* specified in an application of this Subclause.
2) Let *SSUDT* be the set of supertypes of *UDT*.
3) Let *SUDT* be the data type, if any, in *SSUDT* such that the transform descriptor included in the data type descriptor of *SUDT* includes a group descriptor *GD* that includes a group name that is equivalent to *GN*.
4) The *applicable from-sql function* is the SQL-invoked function identified by the specific name of the from-sql function, if any, in *GD*.

Access Rules

None.

General Rules

None.

Conformance Rules

None.

9.18 Determination of a from-sql function for an overriding method

Function

Determine the from-sql function of a user-defined type given the name of an overriding method and the ordinal position of an SQL parameter.

Syntax Rules

1) Let *R* and *N* be a *ROUTINE* and a *POSITION* specified in an application of this Subclause.
2) Let *OM* be original method of *R*.
3) The *applicable from-sql function* is the from-sql function associated with the *N*-th SQL parameter of *OM*, if any.

Access Rules

None.

General Rules

None.

Conformance Rules

None.

9.19 Determination of a to-sql function

Function

Determine the to-sql function of a user-defined type given the name of a user-defined type and the name of a group.

Syntax Rules

1) Let *UDT* and *GN* be a *TYPE* and a *GROUP* specified in an application of this Subclause.

2) Let *SSUDT* be the set of supertypes of *UDT*.

3) Let *SUDT* be the data type, if any, in *SSUDT* such that the transform descriptor included in the data type descriptor of *SUDT* includes a group descriptor *GD* that includes a group name that is equivalent to *GN*.

4) The *applicable to-sql function* is the SQL-invoked function identified by the specific name of the to-sql function, if any, in *GD*.

Access Rules

None.

General Rules

None.

Conformance Rules

None.

9.20 Determination of a to-sql function for an overriding method

Function

Determine the to-sql function of a user-defined type given the name of an overriding method.

Syntax Rules

1) Let *R* be a *ROUTINE* specified in an application of this Subclause.
2) Let *OM* be the original method of *R*
3) The *applicable to-sql function* is the SQL-invoked function associated with the result of *OM*, if any.

Access Rules

None.

General Rules

None.

Conformance Rules

None.

9.21 Generation of the next value of a sequence generator

Function

Generate and return the next value of a sequence generator.

Syntax Rules

None.

Access Rules

None.

General Rules

1) Let *SEQ* be the *SEQUENCE* specified in an application of this Subclause.
2) Let *DT*, *CBV*, *INC*, *SMAX*, and *SMIN* be the data type, current base value, increment, maximum value and minimum value, respectively, of *SEQ*.
3) If there exists a non-negative integer *N* such that *SMIN* ≤ *CBV* + *N* * *INC* ≤ *SMAX* and the value (*CBV* + *N* * *INC*) has not already been returned in the current cycle, then let *V1* be (*CBV* + *N* * *INC*). Otherwise,

   Case:

   a) If the cycle option of *SEQ* is NO CYCLE, then an exception condition is raised: *data exception — sequence generator limit exceeded.*
   b) Otherwise, a new cycle is initiated.

      Case:

      i) If *SEQ* is an ascending sequence generator, then let *V1* be *SMIN*.
      ii) Otherwise, let *V1* be *SMAX*.
4) Case:

   a) If *SEQ* is an ascending sequence generator, the current base value of *SEQ* is set to the value of the lowest non-issued value in the cycle.
   b) Otherwise, the current base value of *SEQ* is set to the highest non-issued value in the cycle.
5) *V1* is returned as the *RESULT*.

Conformance Rules

None.

9.22 Creation of a sequence generator

Function

Complete the definition of an external or internal sequence generator.

Syntax Rules

1) Let *OPT* and *DT* be the *OPTIONS* and *DATA TYPE* specified in an application of this Subclause. *OPT* shall conform to the Format of <common sequence generator options>. The BNF nonterminal symbols used in the remainder of this Subclause refer to the contents of *OPT*.

2) Each of <sequence generator start with option>, <sequence generator increment by option>, <sequence generator maxvalue option>, <sequence generator minvalue option>, and <sequence generator cycle option> shall be specified at most once.

3) If <sequence generator increment by option> is specified, then let *INC* be <sequence generator increment>; otherwise, let *INC* be a <signed numeric literal> whose value is 1 (one).

4) The value of *INC* shall not be 0 (zero).

5) If the value of *INC* is negative, then *SEQ* is a *descending sequence generator*; otherwise, *SEQ* is an *ascending sequence generator*.

6) Case:

   a) If <sequence generator maxvalue option> is specified, then

      Case:

      i) If NO MAXVALUE is specified, then let *SMAX* be an implementation-defined <signed numeric literal> of declared type *DT*.

      ii) Otherwise, let *SMAX* be <sequence generator max value>.

   b) Otherwise, let *SMAX* be an implementation-defined <signed numeric literal> of declared type *DT*.

7) Case:

   a) If <sequence generator minvalue option> is specified, then

      Case:

      i) If NO MINVALUE is specified, then let *SMIN* be an implementation-defined <signed numeric literal> of declared type *DT*.

      ii) Otherwise, let *SMIN* be <sequence generator min value>.

   b) Otherwise, let *SMIN* be an implementation-defined <signed numeric literal> of declared type *DT*.

8) Case:

   a) If <sequence generator start with option> is specified, then let *START* be <sequence generator start value>.

b) Otherwise,

Case:

i) If *SEQ* is an ascending sequence generator, then let *START* be *SMIN*.

ii) Otherwise, let *START* be *SMAX*.

9) The values of *INC*, *START*, *SMAX*, and *SMIN* shall all be exactly representable with the precision and scale of *DT*.

10) The value of *SMAX* shall be greater than the value of *SMIN*.

11) The value of *START* shall be greater than or equal to the value of *SMIN* and lesser than or equal to the value of *SMAX*.

12) If <sequence generator cycle option> is specified, then let *CYC* be <sequence generator cycle option>; otherwise, let *CYC* be NO CYCLE.

Access Rules

None.

General Rules

1) A sequence generator descriptor *SEQDS* that describes *SEQ* is created. *SEQDS* includes:

a) The sequence generator name that is a zero-length character string.

NOTE 213 — The name of an external sequence generator is later set by GR 1) of Subclause 11.62, "<sequence generator definition>"; however, internal sequence generators are anonymous.

b) The data type descriptor of *DT*.

c) The increment specified by *INC*.

d) The maximum value specified by *SMAX*.

e) The minimum value specified by *SMIN*.

f) The cycle option specified by *CYC*.

g) The current base value, set to *START*.

Conformance Rules

None.

9.23 Altering a sequence generator

Function

Complete the alteration of an internal or external sequence generator.

Syntax Rules

1) Let *OPT* and *SEQ* be the *OPTIONS* and *SEQUENCE* specified in an application of this Subclause. *OPT* shall conform to the Format of <alter sequence generator options>. The BNF nonterminal symbols used in the remainder of this Subclause refer to the contents of *OPT*.

2) Let *DT* be the data type descriptor included in *SEQ*.

3) Each of <alter sequence generator restart option>, <sequence generator increment by option>, <sequence generator maxvalue option>, <sequence generator minvalue option>, and <sequence generator cycle option> shall be specified at most once.

4) Case:

   a) If <sequence generator increment> is specified, then:

      i) Let *NEWIV* be <sequence generator increment>.

      ii) The value of *NEWIV* shall not be 0 (zero).

   b) Otherwise, let *NEWIV* be the increment of *SEQ*.

5) Case:

   a) If <sequence generator maxvalue option> is specified, then

      Case:

      i) If NO MAXVALUE is specified, then let *NEWMAX* be an implementation-defined <signed numeric literal> of declared type *DT*.

      ii) Otherwise, let *NEWMAX* be <sequence generator max value>.

   b) Otherwise let *NEWMAX* be the maximum value of *SEQ*.

6) Case:

   a) If <sequence generator minvalue option> is specified, then

      Case:

      i) If NO MINVALUE is specified, then let *NEWMIN* be an implementation-defined <signed numeric literal> of declared type *DT*.

      ii) Otherwise, let *NEWMIN* be <sequence generator min value>.

   b) Otherwise let *NEWMIN* be the minimum value of *SEQ*.

7) If <sequence generator cycle option> is specified, then let *NEWCYCLE* be <sequence generator cycle option>; otherwise, let *NEWCYCLE* be the cycle option of *SEQ*.

8) If <alter sequence generator restart option> is specified, then let *NEWVAL* be <sequence generator restart value>; otherwise, let *NEWVAL* be the current base value of *SEQ*.

9) The values of *NEWIV*, *NEWMAX*, *NEWMIN*, and *NEWVAL* shall all be exactly representable with the precision and scale of *DT*.

10) The value of *NEWMIN* shall be less than the value of *NEWMAX*.

11) The value of *NEWVAL* shall be greater than or equal to the value of *NEWMIN* and lesser than or equal to the value of *NEWMAX*.

Access Rules

None.

General Rules

1) *SEQ* is modified as follows:
   a) The increment is set to *NEWIV*.
   b) The maximum value is set to *NEWMAX*.
   c) The minimum value is set to *NEWMIN*.
   d) The cycle option is set to *NEWCYCLE*.
   e) The current base value is set to *NEWVAL*.

Conformance Rules

None.

10 Additional common elements

10.1 <interval qualifier>

Function

Specify the precision of an interval data type.

Format

```
<interval qualifier> ::=
    <start field> TO <end field>
  | <single datetime field>

<start field> ::=
    <non-second primary datetime field>
    [ <left paren> <interval leading field precision> <right paren> ]

<end field> ::=
    <non-second primary datetime field>
  | SECOND [ <left paren> <interval fractional seconds precision> <right paren> ]

<single datetime field> ::=
    <non-second primary datetime field>
    [ <left paren> <interval leading field precision> <right paren> ]
  | SECOND [ <left paren> <interval leading field precision>
    [ <comma> <interval fractional seconds precision> ] <right paren> ]

<primary datetime field> ::=
    <non-second primary datetime field>
  | SECOND

<non-second primary datetime field> ::=
    YEAR
  | MONTH
  | DAY
  | HOUR
  | MINUTE

<interval fractional seconds precision> ::= <unsigned integer>

<interval leading field precision> ::= <unsigned integer>
```

Syntax Rules

1) There is an ordering of significance of <primary datetime field>s. In order from most significant to least significant, the ordering is: YEAR, MONTH, DAY, HOUR, MINUTE, and SECOND. A <start field> or

<single datetime field> with an <interval leading field precision> *i* is more significant than a <start field> or <single datetime field> with an <interval leading field precision> *j* if *i*>*j*. An <end field> or <single datetime field> with an <interval fractional seconds precision> *i* is less significant than an <end field> or <single datetime field> with an <interval fractional seconds precision> *j* if *i*>*j*.

2) If TO is specified, then <start field> shall be more significant than <end field> and <start field> shall not specify MONTH. If <start field> specifies YEAR, then <end field> shall specify MONTH.

3) The maximum value of <interval leading field precision> is implementation-defined, but shall not be less than 2.

4) The maximum value of <interval fractional seconds precision> is implementation-defined, but shall not be less than 6.

5) An <interval leading field precision>, if specified, shall be greater than 0 (zero) and shall not be greater than the implementation-defined maximum. If <interval leading field precision> is not specified, then an <interval leading field precision> of 2 is implicit.

6) An <interval fractional seconds precision>, if specified, shall be greater than or equal to 0 (zero) and shall not be greater than the implementation-defined maximum. If SECOND is specified and <interval fractional seconds precision> is not specified, then an <interval fractional seconds precision> of 6 is implicit.

7) The precision of a field other than the <start field> or <single datetime field> is

   Case:

   a) If the field is not SECOND, then 2.

   b) Otherwise, 2 digits before the decimal point and the explicit or implicit <interval fractional seconds precision> after the decimal point.

Access Rules

None.

General Rules

1) An item qualified by an <interval qualifier> contains the datetime fields identified by the <interval qualifier>.

   Case:

   a) If the <interval qualifier> specifies a <single datetime field>, then the <interval qualifier> identifies a single <primary datetime field>. Any reference to the *most significant* or *least significant* <primary datetime field> of the item refers to that <primary datetime field>.

   b) Otherwise, the <interval qualifier> identifies those datetime fields from <start field> to <end field>, inclusive.

2) An <interval leading field precision> specifies

   Case:

a) If the <primary datetime field> is SECOND, then the number of decimal digits of precision before the specified or implied decimal point of the seconds <primary datetime field>.

b) Otherwise, the number of decimal digits of precision of the first <primary datetime field>.

3) An <interval fractional seconds precision> specifies the number of decimal digits of precision following the specified or implied decimal point in the <primary datetime field> SECOND.

4) The length in positions of an item of type interval is computed as follows.

Case:

a) If the item is a year-month interval, then

Case:

i) If the <interval qualifier> is a <single datetime field>, then the length in positions of the item is the implicit or explicit <interval leading field precision> of the <single datetime field>.

ii) Otherwise, the length in positions of the item is the implicit or explicit <interval leading field precision> of the <start field> plus 2 (the length of the <non-second primary datetime field> that is the <end field>) plus 1 (one) (the length of the <minus sign> between the <years value> and the <months value> in a <year-month literal>).

b) Otherwise,

Case:

i) If the <interval qualifier> is a <single datetime field> that does not specify SECOND, then the length in positions of the item is the implicit or explicit <interval leading field precision> of the <single datetime field>.

ii) If the <interval qualifier> is a <single datetime field> that specifies SECOND, then the length in positions of the item is the implicit or explicit <interval leading field precision> of the <single datetime field> plus the implicit or explicit <interval fractional seconds precision>. If <interval fractional seconds precision> is greater than zero, then the length in positions of the item is increased by 1 (one) (the length in positions of the <period> between the <seconds integer value> and the <seconds fraction>).

iii) Otherwise, let *participating datetime fields* mean the datetime fields that are less significant than the <start field> and more significant than the <end field> of the <interval qualifier>. The length in positions of each participating datetime field is 2.

Case:

1) If <end field> is SECOND, then the length in positions of the item is the implicit or explicit <interval leading field precision>, plus 3 times the number of participating datetime fields (each participating datetime field has length 2 positions, plus the <minus sign>s or <colon>s that precede them have length 1 (one) position), plus the implicit or explicit <interval fractional seconds precision>, plus 3 (the length in positions of the <end field> other than any <interval fractional seconds precision> plus the length in positions of its preceding <colon>). If <interval fractional seconds precision> is greater than zero, then the length in positions of the item is increased by 1 (one) (the length in positions of the <period> within the field identified by the <end field>).

2) Otherwise, the length in positions of the item is the implicit or explicit <interval leading field precision>, plus 3 times the number of participating datetime fields (each participating datetime field has length 2 positions, plus the <minus sign>s or <colon>s that precede them have length 1 (one) position), plus 2 (the length in positions of the <end field>), plus 1 (one) (the length in positions of the <colon> preceding the <end field>).

Conformance Rules

1) Without Feature F052, "Intervals and datetime arithmetic", conforming SQL language shall not contain an <interval qualifier>.

10.2 <language clause>

Function

Specify a standard programming language.

Format

```
<language clause> ::= LANGUAGE <language name>

<language name> ::=
    ADA
  | C
  | COBOL
  | FORTRAN
  | M | MUMPS
  | PASCAL
  | PLI
  | SQL
```

Syntax Rules

1) If MUMPS is specified, then M is implicit.

Access Rules

None.

General Rules

1) The standard programming language specified by the <language clause> is defined in the International Standard identified by the <language name> keyword. Table 15, "Standard programming languages", specifies the relationship.

Table 15 — Standard programming languages

Language keyword	Relevant standard
ADA	ISO/IEC 8652
C	ISO/IEC 9899
COBOL	ISO 1989
FORTRAN	ISO 1539

Language keyword	**Relevant standard**
M	ISO/IEC 11756
PASCAL	ISO/IEC 7185 and ISO/IEC 10206
PLI	ISO 6160
SQL	ISO/IEC 9075

Conformance Rules

None.

10.3 <path specification>

Function

Specify an order for searching for an SQL-invoked routine.

Format

```
<path specification> ::= PATH <schema name list>

<schema name list> ::= <schema name> [ { <comma> <schema name> }... ]
```

Syntax Rules

1) No two <schema name>s contained in <schema name list> shall be equivalent.

Access Rules

None.

General Rules

None.

Conformance Rules

1) Without Feature S071, "SQL paths in function and type name resolution", conforming SQL language shall not contain a <path specification>.

10.4 <routine invocation>

Function

Invoke an SQL-invoked routine.

Format

```
<routine invocation> ::= <routine name> <SQL argument list>

<routine name> ::= [ <schema name> <period> ] <qualified identifier>

<SQL argument list> ::=
    <left paren> [ <SQL argument> [ { <comma> <SQL argument> }... ] ] <right paren>

<SQL argument> ::=
    <value expression>
  | <generalized expression>
  | <target specification>

<generalized expression> ::=
    <value expression> AS <path-resolved user-defined type name>
```

Syntax Rules

1) Let *RI* be the <routine invocation>, let *TP* be the SQL-path (if any), and let *UDTSM* be the user-defined type of the static SQL-invoked method (if any) specified in an application of this Subclause.

2) Let *RN* be the <routine name> immediately contained in the <routine invocation> *RI*.

3) If *RI* is immediately contained in a <call statement>, then the <SQL argument list> of *RI* shall not contain a <generalized expression> without an intervening <routine invocation>.

4) Case:

   a) If *RI* is immediately contained in a <call statement>, then an SQL-invoked routine *R* is a *possibly candidate routine* for *RI* (henceforth, simply "possibly candidate routine") if *R* is an SQL-invoked procedure and the <qualified identifier> of the <routine name> of *R* is equivalent to the <qualified identifier> of *RN*.

   b) If *RI* is immediately contained in a <method selection>, then an SQL-invoked routine *R* is a *possibly candidate routine* for *RI* if *R* is an instance SQL-invoked method and the <qualified identifier> of the <routine name> of *R* is equivalent to the <qualified identifier> of *RN*.

   c) If *RI* is immediately contained in a <constructor method selection>, then an SQL-invoked routine *R* is a *possibly candidate routine* for *RI* if *R* is an SQL-invoked constructor method and the <qualified identifier> of the <routine name> of *R* is equivalent to the <qualified identifier> of *RN*.

   d) If *RI* is immediately contained in a <static method selection>, then an SQL-invoked routine *R* is a *possibly candidate routine* for *RI* if *R* is a static SQL-invoked method and the <qualified identifier> of the <routine name> of *R* is equivalent to the <qualified identifier> of *RN* and the method specification descriptor for R is included in a user-defined type descriptor for *UDTSM* or for some supertype of *UDTSM*.

e) Otherwise, an SQL-invoked routine R is a *possibly candidate routine* for *RI* if R is an SQL-invoked regular function and the <qualified identifier> of the <routine name> of R is equivalent to the <qualified identifier> of *RN*.

5) Case:

a) If *RI* is contained in an <SQL schema statement>, then an <SQL-invoked routine> R is an *executable routine* if and only if R is a possibly candidate routine and the applicable privileges for the <authorization identifier> that owns the containing schema include EXECUTE on R.

b) Otherwise, an <SQL-invoked routine> R is an *executable routine* if and only if R is a possibly candidate routine and the current privileges include EXECUTE on R.

NOTE 214 — "applicable privileges" and "current privileges" are defined in Subclause 12.3, "<privileges>".

6) Case:

a) If <SQL argument list> does not immediately contain at least one <SQL argument>, then an *invocable routine* is an executable routine that has no SQL parameters.

b) Otherwise:

i) Let *NA* be the number of <SQL argument>s in the <SQL argument list> *AL* of *RI*. Let A_i, 1 (one) $\leq i \leq NA$, be the i-th <SQL argument> in *AL*.

ii) Let the *static SQL argument list* of *RI* be *AL*.

iii) Let P_i be the i-th SQL parameter of an executable routine. An *invocable routine* is an SQL-invoked routine *SIR* that is an executable routine such that:

1) *SIR* has *NA* SQL parameters.

2) If *RI* is not immediately contained in a <call statement>, then for each A_i that is not a <dynamic parameter specification>,

Case:

A) If the declared type of P_i is a user-defined type, then:

I) Let ST_i be the set of subtypes of the declared type of A_i.

II) The type designator of the declared type of P_i shall be in the type precedence list of the data type of some type in ST_i.

NOTE 215 — "type precedence list" is defined in Subclause 9.5, "Type precedence list determination".

B) Otherwise, the type designator of the declared type of P_i shall be in the type precedence list of the declared type of A_i.

NOTE 216 — "type precedence list" is defined in Subclause 9.5, "Type precedence list determination".

7) If <SQL argument list> does not immediately contain at least one <SQL argument>, then:

a) Let *AL* be an empty list of SQL arguments.

b) The subject routine of *RI* is defined as follows:

   i) If *RN* does not contain a <schema name>, then:

      1) Case:

         A) If *RI* is immediately contained in a <method selection>, <static method selection>, or a <constructor method selection>, then let *DP* be *TP*.

         B) If the routine execution context of the current SQL-session indicates that an SQL-invoked routine is active, then let *DP* be the routine SQL-path of that routine execution context.

         C) Otherwise,

            Case:

            I) If *RI* is contained in a <schema definition>, then let *DP* be the SQL-path of that <schema definition>.

            II) If *RI* is contained in a <preparable statement> that is prepared in the current SQL-session by an <execute immediate statement> or a <prepare statement> or in a <direct SQL statement> that is invoked directly, then let *DP* be the SQL-path of the current SQL-session.

            III) Otherwise, let *DP* be the SQL-path of the <SQL-client module definition> that contains *RI*.

      2) The *subject routine* of *RI* is an SQL-invoked routine *SIRSR* such that:

         A) *SIRSR* is an invocable routine.

         B) The <schema name> of the schema of *SIRSR* is in *DP*.

         C) Case:

            I) If the routine descriptor of *SIRSR* does not include a STATIC indication, then there is no other invocable routine *R2* for which the the <schema name> of the schema that includes *R2* precedes in *DP* the <schema name> of the schema that includes *SIRSR*.

            II) If the routine descriptor of *SIRSR* includes a STATIC indication, then there is no other invocable routine *R2* for which the user-defined type described by the descriptor that includes the routine descriptor of *R2* is a subtype of the user-defined type described by the user-defined type descriptor that includes the routine descriptor of *SIRSR*.

   ii) If *RN* contains a <schema name> *SN*, then *SN* shall be the <schema name> of a schema *S*. The *subject routine* of *RI* is the invocable routine (if any) contained in *S*.

c) There shall be exactly one subject routine of *RI*.

d) If *RI* is not immediately contained in a <call statement>, then the *effective returns data type* of *RI* is the result data type of the subject routine of *RI*.

e) Let the *static SQL argument list* of *RI* be an empty list of SQL arguments.

8) If <SQL argument list> immediately contains at least one <SQL argument>, then:

a) The <data type> of each <value expression> immediately contained in a <generalized expression> shall be a subtype of the structured type identified by the <user-defined type name> simply contained in the <path-resolved user-defined type name> that is immediately contained in <generalized expression>.

b) The set of *candidate routines* of *RI* is defined as follows:

Case:

i) If *RN* does not contain a <schema name>, then:

1) Case:

A) If *RI* is immediately contained in a <method selection>, a <static method selection>, or a <constructor method selection>, then let *DP* be *TP*.

B) If the routine execution context of the current SQL-session indicates that an SQL-invoked routine is active, then let *DP* be the routine SQL-path of that routine execution context.

C) Otherwise,

Case:

I) If *RI* is contained in a <schema definition>, then let *DP* be the SQL-path of that <schema definition>.

II) If *RI* is contained in a <preparable statement> that is prepared in the current SQL-session by an <execute immediate statement> or a <prepare statement> or in a <direct SQL statement> that is invoked directly, then let *DP* be the SQL-path of the current SQL-session.

III) Otherwise, let *DP* be the SQL-path of the <SQL-client module definition> that contains *RI*.

2) The candidate routines of *RI* are the set union of invocable routines of all schemas whose <schema name> is in *DP*.

ii) If *RN* contains a <schema name> *SN*, then *SN* shall be the <schema name> of a schema *S*. The candidate routines of *RI* are the invocable routines (if any) contained in *S*.

c) Case:

i) If *RI* is immediately contained in a <call statement>, then:

1) Let *XAL* be *AL*.

2) The subject routine *SR* of *XAL* is the SQL-invoked routine *SIRCR1* that is a candidate routine of *RI* such that there is no other candidate routine *R2* for which the <schema name> of the schema that includes *R2* precedes in *DP* the <schema name> of the schema that includes *SIRCR1*.

3) Let *PL* be the list of SQL parameters P_i of *SR*.

4) For each P_i that is an output SQL parameter or both an input SQL parameter and an output SQL parameter, A_i shall be a <target specification>.

   A) If *RI* is contained in a <triggered SQL statement> of an AFTER trigger, then A_i shall not be a <column reference>.

   B) If A_i is an <embedded variable specification> or a <host parameter specification>, then P_i shall be assignable to A_i, according to the Syntax Rules of Subclause 9.1, "Retrieval assignment", with A_i and P_i as *TARGET* and *VALUE*, respectively.

   C) If A_i is an <SQL parameter reference>, a <column reference>, or a <target array element specification>, then P_i shall be assignable to A_i, according to the Syntax Rules of Subclause 9.2, "Store assignment", with A_i and P_i as *TARGET* and *VALUE*, respectively.

   NOTE 217 — The <column reference> can only be a new transition variable column reference.

5) For each P_i that is an input SQL parameter but not an output SQL parameter, A_i shall be a <value expression>.

6) For each P_i that is an input SQL parameter or both an input SQL parameter and an output SQL parameter, A_i shall be assignable to P_i, according to the Syntax Rules of Subclause 9.2, "Store assignment", with P_i and A_i as *TARGET* and *VALUE*, respectively.

ii) Otherwise:

1) A_i shall be a <value expression> or <generalized expression>.

2) Case:

   A) If A_i is a <generalized expression>, then let TS_i be the data type identified by the <user-defined type name> simply contained in the <path-resolved user-defined type name> that is immediately contained in the <generalized expression>.

   B) Otherwise, let TS_i be the data type whose data type name is included in the data type descriptor of the data type of A_i.

3) The *subject routine* is defined as follows:

   A) For each A_i,

   Case:

      I) If A_i is a <dynamic parameter specification>, then let V_i be A_i.

      II) Otherwise, let V_i be a value arbitrarily chosen whose declared type is TS_i.

   B) Let *XAL* be an <SQL argument list> with *N* <SQL argument>s derived from the V_is ordered according to their ordinal position *i* in *XAL*. The Syntax Rules of Subclause 9.4, "Subject routine determination", are applied to the candidate routines of *RI* and *XAL*, yielding a set of candidate subject routines *CSR*.

   C) Case:

I) If *RN* contains a <schema name>, then there shall be exactly one candidate subject routine in *CSR*. The subject routine *SR* is the candidate subject routine in *CSR*.

II) Otherwise:

1) There shall be at least one candidate subject routine in *CSR*.

2) Case:

a) If there is exactly one candidate subject routine in *CSR*, then the subject routine *SR* is the candidate subject routine in *CSR*.

b) If there is more than one candidate subject routine in *CSR*, then

Case:

i) If *RI* is not immediately contained in a <static method selection>, then there shall be an SQL-invoked routine *SIRCR2* in *CSR* such that there is no other candidate subject routine *R2* in *CSR* for which any of the following is true:

1) The <schema name> of the schema that includes *R2* precedes in *DP* the <schema name> of the schema that includes *SIRCR2*.

2) The <schema name> of the schema that includes *R2* is equivalent to the <schema name> of the schema that includes *SIRCR2*.

The subject routine *SR* is *SIRCR2*.

ii) Otherwise, there shall be an SQL-invoked routine *SIRCR3* in *CSR* such that there is no other candidate subject routine *R2* in *CSR* for which the user-defined type described by the user-defined type descriptor that includes the routine descriptor of *R2* is a subtype of the user-defined type described by the user-defined type descriptor that includes the routine descriptor of *SIRCR3*. The subject routine *SR* is *SIRCR3*.

4) The subject routine of *RI* is the subject routine *SR*.

5) Let *PL* be the list of SQL parameters P_i of *SR*.

6) For each P_i, A_i shall be assignable to P_i according to the Syntax Rules of Subclause 9.2, "Store assignment", with P_i and A_i as *TARGET* and *VALUE*, respectively.

7) The *effective returns data type* of *RI* is defined as follows:

A) Case:

I) If *SR* is a type-preserving function, then let P_i be the result SQL parameter of *SR*. If A_i contains a <generalized expression>, then let *RT* be the declared type of the <value expression> contained in the <generalized expression> of A_i; otherwise, let *RT* be the declared type of A_i.

II) Otherwise, let *RT* be the result data type of *SR*.

B) The *effective returns data type* of *RI* is *RT*.

9) If *SR* is a constructor function, then *RI* shall be simply contained in a <new invocation>.

Access Rules

None.

General Rules

1) Let *SAL* and *SR* be the static SQL argument list and subject routine of the <routine invocation> as specified in an application of this Subclause.

   NOTE 218 — "static SQL argument list" and "subject routine" are defined by the Syntax Rules of this Subclause.

2) Case:

   a) If *SAL* is empty, then let the *dynamic SQL argument list DAL* be *SAL*.

   b) Otherwise:

      i) Each SQL argument A_i in *SAL* is evaluated, in an implementation-dependent order, to obtain a value V_i.

      ii) Let the *dynamic SQL argument list DAL* be the list of values V_i in order.

      iii) If *SR* is type preserving and the null value is substituted for the result parameter, then

      Case:

         1) If *SR* is a mutator function, then an exception condition is raised: *data exception — null value substituted for mutator subject parameter.*

         2) Otherwise, the value of *RI* is the null value and the remaining General Rules of this Subclause are not applied.

      iv) Case:

         1) If *SR* is an instance SQL-invoked method, then:

            A) If V_1 is the null value, then the value of *RI* is the null value and the remaining General Rules of this Subclause are not applied.

            B) Let *SM* be the set of SQL-invoked methods *M* that satisfy the following conditions:

               I) The <routine name> of *SR* and the <routine name> of *M* have equivalent <qualified identifier>s.

               II) *SR* and *M* have the name number *N* of SQL parameters. Let PSR_i, 1 (one) $\leq i \leq N$, be the *i*-th SQL parameter of *SR* and PM_i, 1 (one) $\leq i \leq N$, be the *i*-th SQL parameter of *M*.

III) The declared type of the subject parameter of *M* is a subtype of the declared type of the subject parameter of *SR*.

IV) For *j* varying from 2 to *N*, the Syntax Rules of Subclause 9.16, "Data type identity", are applied with the declared type of PM_j and the declared type of PSR_j.

NOTE 219 — *SR* is an element of the set *SM*.

C) *SM* is the *set of overriding methods* of *SR* and every SQL-invoked method *M* in *SM* is an *overriding method* of *SR*.

D) Case:

I) If the first SQL argument *A1* in *SAL* contains a <generalized expression>, then let *DT1* be the data type identified by the <user-defined type name> contained in the <generalized expression> of *A1*.

II) Otherwise, let *DT1* be the most specific type of V_1.

E) Let *R* be the SQL-invoked method in *SM* such that there is no other SQL-invoked method *M1* in *SM* for which the type designator of the declared type of the subject parameter of *M1* precedes that of the declared type of the subject parameter of *R* in the type precedence list of *DT1*.

2) Otherwise, let *R* be *SR*.

3) Let *N* and *PN* be the number of values V_i in *DAL*. Let T_i be the declared type of the *i*-th SQL parameter P_i of *R*. For *i* ranging from 1 (one) to *PN*,

Case:

a) If P_i is an input SQL parameter or both an input SQL parameter and an output SQL parameter, then let CPV_i be the result of the assignment of V_i to a target of type T_i according to the rules of Subclause 9.2, "Store assignment".

b) Otherwise,

Case:

i) If *R* is an SQL routine, then let CPV_i be the null value.

ii) Otherwise, let CPV_i be an implementation-defined value of most specific type T_i.

4) If *R* is an external routine, then:

a) Let *P* be the program identified by the external name of *R*.

b) For *i* ranging from 1 (one) to *N*, let P_i be the *i*-th SQL parameter of *R* and let T_i be the declared type of P_i.

Case:

i) If P_i is an input SQL parameter or both an input SQL parameter and an output SQL parameter, then

Case:

1) If P_i is a locator parameter, then CPV_i is replaced by the locator value that uniquely identifies the value of CPV_i.

2) If T_i is a user-defined type, and P_i is not a locator parameter, then:

   A) Let FSF_i be the SQL-invoked routine identified by the specific name of the from-sql function associated with P_i in the routine descriptor of R. Let RT_i be the result data type of FSF_i.

   B) The General Rules of this Subclause are applied with a static SQL argument list that has a single argument that is CPV_i and subject routine FSF_i.

   C) Let RV_i be the result of the invocation of FSF_i. CPV_i is replaced by RV_i.

ii) Otherwise,

Case:

1) If P_i is a locator parameter, then CPV_i is replaced with an implementation-dependent value of type INTEGER.

2) If T_i is a user-defined type and P_i is not a locator parameter, then:

   A) Let FSF_i be the SQL-invoked routine identified by the specific name of the from-sql function associated with P_i in the routine descriptor of R. Let RT_i be the result data type of FSF_i.

   B) CPV_i is replaced by an implementation-defined value of type RT_i.

5) Preserve the current SQL-session context *CSC* and create a new SQL-session context *RSC* derived from *CSC* as follows:

a) Set the current default catalog name, the current default unqualified schema name, the current default character set name, the SQL-path of the current SQL-session, the current default time zone displacement of the current SQL-session, and the contents of all SQL dynamic descriptor areas to implementation-defined values.

b) Set the values of the current SQL-session identifier, the identities of all instances of global temporary tables, the current constraint mode for each integrity constraint, the current transaction access mode, the current transaction isolation level, and the current transaction condition area limit to their values in *CSC*.

c) The diagnostics area stack in *CSC* is copied to *RSC* and the General Rules of Subclause 22.2, “Pushing and popping the diagnostics area stack”, are applied with “PUSH” as *OPERATION* and the diagnostics area stack in *RSC* as *STACK*.

d) Case:

i) If *R* is an SQL routine, then remove from *RSC* the identities of all instances of created local temporary tables, declared local temporary tables that are defined by <temporary table declara-

tion>s that are contained in <SQL-client module definition>s, and the cursor position of all open cursors.

ii) Otherwise:

1) Remove from *RSC* the identities of all instances of created local temporary tables that are referenced in <SQL-client module definition>s that are not the <SQL-client module definition> of *P*, declared local temporary tables that are defined by <temporary table declaration>s that are contained in <SQL-client module definition>s that are not the <SQL-client module definition> of *P*, and the cursor position of all open cursors that are defined by <declare cursor>s that are contained in <SQL-client module definition>s that are not the <SQL-client module definition> of *P*.

2) It is implementation-defined whether the identities of all instances of created local temporary tables that are referenced in the <SQL-client module definition> of *P*, declared local temporary tables that are defined by <temporary table declaration>s that are contained in the <SQL-client module definition> of *P*, and the cursor position of all open cursors that are defined by <declare cursor>s that are contained in the <SQL-client module definition> of *P* are removed from *RSC*.

e) Indicate in the routine execution context of *RSC* that the SQL-invoked routine *R* is active.

f) Case:

i) If the SQL-data access indication of *CSC* specifies possibly contains SQL and *R* possibly reads SQL-data or *R* possibly modifies SQL-data, then:

1) If *R* is an external routine, then an exception condition is raised: *external routine exception — reading SQL-data not permitted.*

2) Otherwise, an exception condition is raised: *SQL routine exception — reading SQL-data not permitted.*

ii) If the SQL-data access indication of *CSC* specifies possibly reads SQL and *R* possibly modifies SQL-data, then:

1) If *R* is an external routine, then an exception condition is raised: *external routine exception — modifying SQL-data not permitted.*

2) Otherwise, an exception condition is raised: *SQL routine exception — modifying SQL-data not permitted.*

g) Case:

i) If *R* does not possibly contain SQL, then set the SQL-data access indication in the routine execution context of *RSC* to *does not possibly contain SQL.*

ii) If *R* possibly contains SQL, then set the SQL-data access indication in the routine execution context of *RSC* to *possibly contains SQL.*

iii) If *R* possibly reads SQL-data, then set the SQL-data access indication in the routine execution context of *RSC* to *possibly reads SQL-data.*

iv) If *R* possibly modifies SQL-data, then set the SQL-data access indication in the routine execution context of *RSC* to *possibly modifies SQL-data.*

h) The authorization stack of *RSC* is set to a copy of the authorization stack of *CSC*.

i) A copy of the top cell is pushed onto the authorization stack of *RSC*.

j) Case:

  i) If *R* is an external routine, then:

    1) Case:

      A) If the external security characteristic of *R* is IMPLEMENTATION DEFINED, then the current user identifier and the current role name of *RSC* are implementation-defined.

      B) If the external security characteristic of *R* is DEFINER, then the top cell of the authorization stack of *RSC* is set to contain only the external routine authorization identifier of *R*.

    2) Set the routine SQL-path of *RSC* to be the external routine SQL-path of *R*.

  ii) Otherwise:

    1) If the SQL security characteristic of *R* is DEFINER, then the current authorization identifier of *RSC* is set to the routine authorization identifier of *R*.

    2) Set the routine SQL-path of *RSC* to be the routine SQL-path of *R*.

k) *RSC* becomes the current SQL-session context.

6) If the descriptor of *R* includes an indication that a new savepoint level is to be established when *R* is invoked, then a new savepoint level is established.

7) If *R* is an SQL routine, then

Case:

a) If *R* is a null-call function and if any of CPV_i is the null value, then let *RV* be the null value.

b) Otherwise:

  i) For *i* ranging from 1 (one) to *PN*, set the value of P_i to CPV_i.

  ii) The General Rules of Subclause 13.5, "<SQL procedure statement>", are evaluated with the SQL routine body of *R* as the *executing statement*.

  iii) If, before the completion of the execution of the SQL routine body of *R*, an attempt is made to execute an SQL-connection statement, then an exception condition is raised: *SQL routine exception — prohibited SQL-statement attempted.*

  iv) Case:

    1) If the SQL-implementation does not support Feature T272, "Enhanced savepoint management", and, before the completion of the execution of the SQL routine body of *R*, an attempt is made to execute an SQL-transaction statement, then an exception condition is raised: *SQL routine exception — prohibited SQL-statement attempted.*

    2) If, before the completion of the execution of the SQL routine body of *R*, an attempt is made to execute an SQL-transaction statement that is not a <savepoint statement> or a <release

savepoint statement>, or is a <rollback statement> that does not specify a <savepoint clause>, then an exception condition is raised: *SQL routine exception — prohibited SQL-statement attempted.*

v) If the SQL implementation does not support Feature T651, "SQL-schema statements in SQL routines", and, before the completion of the execution of the SQL routine body of *R*, an attempt is made to execute an SQL-schema statement, an exception condition is raised: *SQL routine exception — prohibited SQL-statement attempted.*

vi) If the SQL implementation does not support Feature T652, "SQL-dynamic statements in SQL routines", and, before the completion of the execution of the SQL routine body of *R*, an attempt is made to execute an SQL-dynamic statement, an exception condition is raised: *SQL routine exception — prohibited SQL-statement attempted.*

vii) If the SQL-data access indication of *RSC* specifies *possibly contains SQL* and, before the completion of the execution of the SQL routine body of *R*, an attempt is made to execute an SQL-statement that possibly reads SQL-data, or an attempt is made to execute an SQL-statement that possibly modifies SQL-data, then an exception condition is raised: *SQL routine exception — reading SQL-data not permitted.*

viii) If the SQL-data access indication of *RSC* specifies *possibly reads SQL-data* and, before the completion of the execution of the SQL routine body of *R*, an attempt is made to execute an SQL-statement that possibly modifies SQL-data then an exception condition is raised: *SQL routine exception — modifying SQL-data not permitted.*

ix) If *R* is an SQL-invoked function, then

Case:

1) If no <return statement> is executed before completion of the execution of the SQL routine body of *R*, then an exception condition is raised: *SQL routine exception — function executed no return statement.*

2) Otherwise, let *RV* be the returned value of the execution of the SQL routine body of *R*.

NOTE 220 — "Returned value" is defined in Subclause 15.2, "<return statement>".

x) If *R* is an SQL-invoked procedure, then for each SQL parameter of *R* that is an output SQL parameter or both an input SQL parameter and an output SQL parameter, set the value of CPV_i to the value of P_i.

8) If *R* is an external routine, then:

a) The method and time of binding of *P* to the schema or SQL-server module that includes *R* is implementation-defined.

b) If *R* specifies PARAMETER STYLE SQL, then

i) Case:

1) If *R* is an SQL-invoked function, then the effective SQL parameter list *ESPL* of *R* is set as follows:

A) If *R* is an array-returning external function or a multiset-returning external function with the element type being a row type, then let *FRN* be the degree of the element type; otherwise, let *FRN* be 1 (one).

B) For *i* ranging from 1 (one) to *PN*, the *i*-th entry in *ESPL* is set to CPV_i.

C) For *i* ranging from *PN*+1 to *PN*+*FRN*, the *i*-th entries in *ESPL* are the *result data items*.

D) For *i* ranging from (*PN*+*FRN*)+1 to (*PN*+*FRN*)+*N*, the *i*-th entry in *ESPL* is the *SQL indicator argument* corresponding to $CPV_{i-(PN+FRN)}$.

E) For *i* ranging from (*PN*+*FRN*)+*N*+1 to (*PN*+*FRN*)+*N*+*FRN*, the *i*-th entries in *ESPL* are the SQL indicator arguments corresponding to the result data items.

F) For *i* equal to (*PN*+*FRN*)+(*N*+*FRN*)+1, the *i*-th entry in *ESPL* is the *exception data item*.

G) For *i* equal to (*PN*+*FRN*)+(*N*+*FRN*)+2, the *i*-th entry in *ESPL* is the *routine name text item*.

H) For *i* equal to (*PN*+*FRN*)+(*N*+*FRN*)+3, the *i*-th entry in *ESPL* is the *specific name text item*.

I) For *i* equal to (*PN*+*FRN*)+(*N*+*FRN*)+4, the *i*-th entry in *ESPL* is the *message text item*.

J) If *R* is an array-returning external function or a multiset-returning external function, then for *i* equal to (*PN*+*FRN*)+(*N*+*FRN*)+5, the *i*-th entry in *ESPL* is the *save area data item* and for *i* equal to (*PN*+*FRN*)+(*N*+*FRN*)+6, the *i*-th entry in *ESPL* is the *call type data item*.

K) Set the values of the SQL indicator arguments corresponding to the result data items (that is, SQL argument value list entries from (*PN*+*FRN*)+*N*+1 through (*PN*+*FRN*)+*N*+*FRN*, inclusive, to 0 (zero).

L) For *i* ranging from 1 (one) to *PN*, if CPV_i is the null value, then set entry (*PN*+*FRN*)+*i* (that is, the *i*-th SQL indicator argument corresponding to CPV_i) to –1; otherwise, set entry (*PN*+*FRN*)+*i* (that is, the *i*-th SQL indicator argument corresponding to CPV_i) to 0 (zero).

M) If *R* is an array-returning external function or a multiset-returning external function, then set the value of the save area data item (that is, SQL argument value list entry (*PN*+*FRN*)+(*N*+*FRN*)+5) to 0 (zero) and set the value of the call type data item (that is, SQL argument value list entry (*PN*+*FRN*)+(*N*+*FRN*)+6) to –1.

2) Otherwise, the effective SQL parameter list *ESPL* of *R* is set as follows:

A) For *i* ranging from 1 (one) to *PN*, the *i*-th entry in *ESPL* is CPV_i.

B) For *i* ranging from *PN*+1 to *PN*+*N*, the *i*-th entry in *ESPL* is the *SQL indicator argument* corresponding to CPV_{i-PN}.

C) For *i* equal to (*PN*+*N*)+1, the *i*-th entry in *ESPL* is the *exception data item*.

D) For *i* equal to (*PN*+*N*)+2, the *i*-th entry in *ESPL* is the *routine name text item*.

E) For i equal to $(PN+N)+3$, the i-th entry in *ESPL* is the *specific name text item*.

F) For i equal to $(PN+N)+4$, the i-th entry in *ESPL* is the *message text item*.

G) For i ranging from 1 (one) to *PN*, if CPV_i is the null value, then set entry $PN+i$ in *ESPL* (that is, the i-th SQL indicator argument corresponding to CPV_i) to –1; otherwise, set entry $PN+i$ in *ESPL* (that is, the i-th SQL indicator argument corresponding to CPV_i) to 0 (zero).

ii) The exception data item is set to '00000'.

iii) The routine name text item is set to the <schema qualified name> of the routine name of *R*.

iv) The specific name text item is set to the <qualified identifier> of the specific name of *R*.

v) The message text item is set to a zero-length string.

c) If *R* specifies PARAMETER STYLE GENERAL, then the effective SQL parameter list *ESPL* of *R* is set as follows:

i) If *R* is not a null-call function and, for i ranging from 1 (one) to *PN*, CPV_i is the null value, then an exception condition is raised: *external routine invocation exception — null value not allowed*.

ii) For i ranging from 1 (one) to *PN*, if no CPV_i is the null value, then the for j ranging from 1 (one) to *PN*, if the j-th entry in *ESPL* is set to CPV_j.

d) If *R* specifies DETERMINISTIC and if different executions of *P* with identical SQL argument value lists do not produce identical results, then the results are implementation-dependent.

e) Let *EN* be the number of entries in *ESPL*. Let ESP_i be the i-th effective SQL parameter in *ESPL*.

f) Case:

i) If *R* is a null-call function and if any of CPV_i is the null value, then *P* is assumed to have been executed.

ii) Otherwise:

1) If *R* is not an array-returning external function or a multiset-returning external function, then *P* is executed with a list of *EN* parameters PD_i whose parameter names are PN_i and whose values are set as follows:

A) Depending on whether the language of *R* specifies ADA, C, COBOL, FORTRAN, M, PASCAL, or PLI, let the *operative data type correspondences table* be Table 16, “Data type correspondences for Ada”, Table 17, “Data type correspondences for C”, Table 18, “Data type correspondences for COBOL”, Table 19, “Data type correspondences for Fortran”, Table 20, “Data type correspondences for M”, Table 21, “Data type correspondences for Pascal”, or Table 22, “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.

B) For i varying from 1 (one) to *EN*, the data type DT_i of PD_i is the data type listed in the host data type column of the row in the data type correspondences table whose value in the SQL data type column corresponds to the data type of ESP_i.

C) The value of PD_i is set to the value of ESP_i.

2) If *R* is an array-returning external function, then:

A) Let *AR* be an array whose declared type is the result data type of *R*.

B) The General Rules of Subclause 9.14, "Execution of array-returning functions", are applied with *AR*, *ESPL*, and *P* as *ARRAY*, *EFFECTIVE SQL PARAMETER LIST*, and *PROGRAM*, respectively.

3) If *R* is a multiset-returning external function, then:

A) Let *MU* be a multiset whose declared type is the result data type of *R*.

B) The General Rules of Subclause 9.15, "Execution of multiset-returning functions", are applied with *MU*, *ESPL*, and *P* as *MULTISET*, *EFFECTIVE SQL PARAMETER LIST*, and *PROGRAM*, respectively.

4) If the SQL-data access indication of *RSC* specifies *does not possibly contain SQL* and, before the completion of any execution of *P*, an attempt is made to execute an SQL-statement, then an exception condition is raised: *external routine exception — containing SQL not permitted.*

5) If, before the completion of any execution of *P*, an attempt is made to execute an SQL-connection statement, then an exception condition is raised: *external routine exception — prohibited SQL-statement attempted.*

6) Case:

A) If the SQL-implementation does not support Feature T272, "Enhanced savepoint management", and, before the completion of the execution of *P*, an attempt is made to execute an SQL-transaction statement, then an exception condition is raised: *SQL routine exception — prohibited SQL-statement attempted.*

B) If, before the completion of the execution of *P*, an attempt is made to execute an SQL-transaction statement that is not <savepoint statement> or <release savepoint statement>, or is a <rollback statement> that does not specify a <savepoint clause>, then an exception condition is raised: *external routine exception — prohibited SQL-statement attempted.*

7) If the SQL implementation does not support Feature T653, "SQL-schema statements in external routines", and, before the completion of any execution of *P*, an attempt is made to execute an SQL-schema statement, then an exception condition is raised: *external routine exception — prohibited SQL-statement attempted.*

8) If the SQL implementation does not support Feature T654, "SQL-dynamic statements in external routines", and, before the completion of any execution of *P*, an attempt is made to execute an SQL-dynamic statement, then an exception condition is raised: *external routine exception — prohibited SQL-statement attempted.*

9) If the SQL-data access indication of *RSC* specifies *possibly contains SQL* and, before the completion of any execution of *P*, an attempt is made to execute an SQL-statement that

possibly reads SQL-data, or an attempt is made to execute an SQL-statement that possibly modifies SQL-data, then an exception condition is raised: *external routine exception — reading SQL-data not permitted.*

10) If the SQL-data access indication of *RSC* specifies *possibly reads SQL* and, before the completion of any execution of *P*, an attempt is made to execute an SQL-statement that possibly modifies SQL-data, then an exception condition is raised: *external routine exception — modifying SQL-data not permitted.*

11) If the language specifies ADA (respectively C, COBOL, FORTRAN, M, PASCAL, PLI) and *P* is not a standard-conforming Ada program (respectively C, COBOL, Fortran, M, Pascal, PL/I program), then the results of any execution of *P* are implementation-dependent.

g) After the completion of any execution of *P*:

i) It is implementation-defined whether:

1) For every open cursor *CR* that is associated with *RSC* and that is defined by a <declare cursor> that is contained in the <SQL-client module definition> of *P*:

A) The following SQL-statement is effectively executed:

```
CLOSE CR
```

B) *CR* is destroyed.

2) Every instance of created local temporary tables and every instance of declared local temporary tables that is associated with *RSC* is destroyed.

3) For every prepared statement *PS* prepared by *P* in the current SQL-transaction that has not been deallocated by *P*:

A) Let *SSN* be the <SQL statement name> that identifies *PS*.

B) The following SQL-statement is effectively executed:

```
DEALLOCATE PREPARE SSN
```

ii) For *i* varying from 1 (one) to *EN*, the value of ESP_i is set to the value of PD_i. If *R* specifies PARAMETER STYLE SQL, then

Case:

1) If the exception data item has the value '00000', then the execution of *P* was successful.

2) If the first two characters of the exception data item are equal to the SQLSTATE condition code class value for *warning*, then a completion condition is raised: *warning*, using a subclass code equal to the final three characters of the value of the exception data item.

3) Otherwise, an exception condition is raised using a class code equal to the first two characters of the value of the exception data item and a subclass code equal to the final three characters of the value of the exception data item.

iii) If the exception data item is not '00000' and *R* specified PARAMETER STYLE SQL, then the message text item is stored in the first diagnostics area.

h) If *R* is an SQL-invoked function, then:

i) Case:

1) If *R* is an SQL-invoked method whose routine descriptor does not include a STATIC indication and if CPV_1 is the null value, then let *RDI* be the null value.

2) If *R* is a null-call function, *R* is not an array-returning external function or a multiset-returning external function, and if any of CPV_i is the null value, then let *RDI* be the null value.

3) If *R* is not a null-call function, *R* specifies PARAMETER STYLE SQL, and entry (*PN*+1)+*N*+1 in *ESPL* (that is, SQL indicator argument *N*+1 corresponding to the result data item) is negative, then let *RDI* be the null value.

4) Otherwise,

A) Case:

I) If *R* is not an array-returning external function or a multiset-returning external function, *R* specifies PARAMETER STYLE SQL, and entry (*PN*+1)+*N*+1 in *ESPL* (that is, SQL indicator argument *N*+1 corresponding to the result data item) is not negative, then let *ERDI* be the value of the result data item.

II) If *R* is an array-returning external function, and *R* specifies PARAMETER STYLE SQL, then let *ERDI* be *AR*.

III) If *R* is a multiset-returning function, and *R* specifies PARAMETER STYLE SQL, then let *ERDI* be *MU*.

IV) If *R* specifies PARAMETER STYLE GENERAL, then let *ERDI* be the value returned from *P*.

NOTE 221 — The value returned from *P* is passed to the SQL-implementation in an implementation-dependent manner. An argument value list entry is not used for this purpose.

B) Case:

I) If the routine descriptor of *R* indicates that the return value is a locator, then

Case:

1) If *RT* is a binary large object type, then let *RDI* be the binary large object value corresponding to *ERDI*.

2) If *RT* is a character large object type, then let *RDI* be the large object character string corresponding to *ERDI*.

3) If *RT* is an array type, then let *RDI* be the array value corresponding to *ERDI*.

4) If *RT* is a multiset type, then let *RDI* be the multiset value corresponding to *ERDI*.

5) If *RT* is a user-defined type, then let *RDI* be the user-defined type value corresponding to *ERDI*.

II) Otherwise, if *R* specifies <result cast>, then let *CRT* be the <data type> specified in <result cast>; otherwise, let *CRT* be the <returns data type> of *R*.

Case:

1) If *R* specifies <result cast> and the routine descriptor of *R* indicates that the <result cast> has a locator indication, then

   Case:

   a) If *CRT* is a binary large object type, then let *RDI* be the binary large object value corresponding to *ERDI*.

   b) If *CRT* is a character large object type, then let *RDI* be the large object character string corresponding to *ERDI*.

   c) If *CRT* is an array type, then let *RDI* be the array value corresponding to *ERDI*.

   d) If *CRT* is a multiset type, then let *RDI* be the multiset value corresponding to *ERDI*.

   e) If *CRT* is a user-defined type, then let *RDI* be the user-defined type value corresponding to *ERDI*.

2) Otherwise,

   Case:

   a) If *CRT* is a user-defined type, then:

      i) Let *TSF* be the SQL-invoked routine identified by the specific name of the to-sql function associated with the result of *R*.

      ii) Case:

         1) If *TSF* is an SQL-invoked method, then:

            A) If *R* is a type-preserving function, then let *MAT* be the most specific type of the value of the argument substituted for the result SQL parameter of *R*; otherwise, let *MAT* be *CRT*.

            B) The General Rules of this Subclause are applied with a static SQL argument list whose first element is the value returned by the invocation of:

            ```
            MAT()
            ```

            and whose second element is *ERDI*, and the subject routine *TSF*.

         2) Otherwise, the General Rules of this Subclause are applied with a static SQL argument list that has a single SQL argument that is *ERDI*, and the subject routine *TSF*.

iii) Let *RDI* be the result of invocation of *TSF*.

b) Otherwise, let *RDI* be *ERDI*.

ii) If *R* specified a <result cast>, then let *RT* be the <returns data type> of *R* and let *RV* be the result of:

```
CAST ( RDI AS RT )
```

Otherwise, let *RV* be *RDI*.

i) If *R* is an SQL-invoked procedure, then for each P_i, 1 (one) ≤ *i* ≤ *PN*, that is an output SQL parameter or both an input SQL parameter and an output SQL parameter,

Case:

i) If *R* specifies PARAMETER STYLE SQL and entry (*PN*+1)+*i* in *ESPL* (that is, the *i*-th SQL indicator argument corresponding to CPV_i) is negative, then CPV_i is set to the null value.

ii) If *R* specifies PARAMETER STYLE SQL, and entry (*PN*+1)+*i* in *ESPL* (that is, the *i*-th SQL indicator argument corresponding to CPV_i) is not negative, and a value was not assigned to the *i*-th entry in *ESPL*, then CPV_i is set to an implementation-defined value of type T_i.

iii) Otherwise:

NOTE 222 — In this case, either *R* specifies PARAMETER STYLE SQL and entry (*PN*+1)+*i* in *SQPL* (that is, the *i*-th SQL indicator argument corresponding to CPV_i) is not negative and a value was assigned to the *i*-th entry in *ESPL*, or else *R* specifies PARAMETER STYLE GENERAL.

1) Let EV_i be the *i*-th entry in *ESPL*. Let T_i be the <data type> of P_i.

2) Case:

A) If P_i is a locator parameter, then

Case:

I) If T_i is a binary large object type, then CPV_i is set to the binary large object value corresponding to EV_i.

II) If T_i is a character large object type, then CPV_i is set to the large object character string corresponding to EV_i.

III) If T_i is an an array type, then CPV_i is set to the array value corresponding to EV_i.

IV) If T_i is an a multiset type, then CPV_i is set to the multiset value corresponding to EV_i.

V) If T_i is a user-defined type, then CPV_i is set to the user-defined type value corresponding to EV_i.

B) If T_i is a user-defined type, then:

I) Let *TSF_i* be the SQL-invoked function identified by the specific name of the to-sql function associated with P_i in the routine descriptor of *R*.

II) Case:

1) If *TSF* is an SQL-invoked method, then the General Rules of this Subclause are applied with a static SQL argument list whose first element is the value returned by the invocation of:

```
Ti()
```

and whose second element is EV_i, and the subject routine TSF_i.

2) Otherwise, the General Rules of this Subclause are applied with a static SQL argument list that has a single SQL argument that is EV_i, and the subject routine TSF_i.

III) CPV_i is set to the result of an invocation of TSF_i.

C) Otherwise, CPV_i is set to EV_i.

9) Case:

a) If *R* is an SQL-invoked function, then:

i) If *R* is a type-preserving function, then:

1) Let *MAT* be the most specific type of the value of the argument substituted for the result SQL parameter of *R*.

2) If *RV* is not the null value and the most specific type of *RV* is not compatible with *MAT*, then an exception condition is raised: *data exception — most specific type mismatch.*

ii) Let *ERDT* be the effective returns data type of the <routine invocation>.

iii) Let the result of the <routine invocation> be the result of assigning *RV* to a target of declared type *ERDT* according to the rules of Subclause 9.2, “Store assignment”.

b) Otherwise, for each SQL parameter P_i of *R* that is an output SQL parameter or both an input SQL parameter and an output SQL parameter, let TS_i be the <target specification> of the corresponding <SQL argument> A_i.

Case:

i) If TS_i is a <host parameter specification> or an <embedded variable specification>, then CPV_i is assigned to TS_i according to the rules of Subclause 9.1, “Retrieval assignment”.

ii) If TS_i is an <SQL parameter reference>, a <column reference>, or a <target array element specification>, then

NOTE 223 — The <column reference> can only be a new transition variable column reference.

Case:

1) If <target array element specification> is specified, then

Case:

A) If the value of TS_i, denoted by C, is null, then an exception condition is raised: *data exception — null value in array target.*

B) Otherwise:

I) Let N be the maximum cardinality of C.

II) Let M be the cardinality of the value of C.

III) Let I be the value of the <simple value specification> immediately contained in TS_i.

IV) Let EDT be the element type of C.

V) Case:

1) If I is greater than zero and less than or equal to M, then the value of C is replaced by an array A with element type EDT and cardinality M derived as follows:

a) For j varying from 1 (one) to I–1 and from I+1 to M, the j-th element in A is the value of the j-th element in C.

b) The I-th element of A is set to the value of CPV_i , denoted by SV, by applying the General Rules of Subclause 9.2, "Store assignment", to the I-th element of A and SV as *TARGET* and *VALUE*, respectively.

2) If I is greater than M and less than or equal to N, then the value of C is replaced by an array A with element type EDT and cardinality I derived as follows:

a) For j varying from 1 (one) to M, the j-th element in A is the value of the j-th element in C.

b) For j varying from M+1 to I, the j-th element in A is the null value.

c) The I-th element of A is set to the value of CPV_i, denoted by SV, by applying the General Rules of Subclause 9.2, "Store assignment", to the I-th element of A and SV as *TARGET* and *VALUE*, respectively.

3) Otherwise, an exception condition is raised: *data exception — array element error.*

2) Otherwise, CPV_i is assigned to TS_i according to the rules of Subclause 9.2, "Store assignment".

10) If the subject routine is a procedure whose descriptor PR includes a maximum number of dynamic result sets that is greater than zero, then a sequence of result sets RRS is returned to INV.

a) Let MAX be maximum number of dynamic result sets included in PR.

b) Let OPN be the actual number of result set cursors declared in the body of the subject routine that remain open when control is returned to INV.

c) Case:

  i) If *OPN* is greater than *MAX*, then:

    1) Let *RTN* be *MAX*.

    2) A completion condition is raised: *warning — attempt to return too many result sets.*

  ii) Otherwise, let *RTN* be *OPN*.

d) Let *FRC* be the ordered set of result set cursors that remain open when *PR* returns to *INV*. Let FRC_i, 1 (one) $\leq i \leq RTN$, be the *i*-th cursor in *FRC*, let $FRCN_i$ be the <cursor name> that identifies FRC_i, and let RS_i be the result set of FRC_i.

e) Case:

  i) If $FRCN_i$, 1 (one) $\leq i \leq RTN$, is a scrollable cursor, then let NXT_i be 1 (one).

  ii) Otherwise, let NXT_i, 1 (one) $\leq i \leq RTN$, be the ordinal number of the row of RS_i that would be retrieved if the following SQL-statement were executed:

```
FETCH NEXT FROM FRCNi
INTO...
```

f) Let TOT_i, 1 (one) $\leq i \leq RTN$, be the original cardinality of RS_i when established by the opening of FRC_i.

g) Let *RRS* be the ordered set of returned result sets RRS_i, 1 (one) $\leq i \leq RTN$, comprising the rows of RS_i at ordinal positions $ROW_{i,j}$, $NXT_i \leq j \leq TOT_i$.

h) A completion condition is raised: *warning — dynamic result sets returned.*

i) RS_i, 1 (one) $\leq i \leq RTN$, is returned to *INV*.

11) Prepare *CSC* to become the current SQL-session context:

  a) Set the value of the current constraint mode for each integrity constraint in *CSC* to the value of the current constraint mode for each integrity constraint in *RSC*.

  b) Set the value of the current transaction access mode in *CSC* to the value of the current transaction access mode in *RSC*.

  c) Set the value of the current transaction isolation level in *CSC* to the value of the current transaction isolation level in *RSC*.

  d) Set the value of the current transaction condition area limit in *CSC* to the value of the current transaction condition area limit *CAL* in *RSC*.

  e) For each occupied condition area *CA* in the first diagnostics area of *RSC*, if the value of RETURNED_SQLSTATE in *CA* does not represent *successful completion*, then

  Case:

i) If the number of occupied condition areas in the first diagnostics area *DA1* in *CSC* is less than *CAL*, then *CA* is copied to the first vacant condition area in *DA1*.

NOTE 224 — This causes the first vacant condition area in *DA1* to become occupied.

ii) Otherwise, the value of MORE in the statement information area of *DA1* is set to 'Y'.

f) Replace the identities of all instances of global temporary tables in *CSC* with the identities of the instances of global temporary tables in *RSC*.

g) Remove the top cell from the authorization stack of *RSC* and set the authorization stack of *CSC* to a copy of the authorization stack of *RSC*.

NOTE 225 — The copying of *RSC*'s authorization stack into *CSC* is necessary in order to carry back any change in the SQL-session user identifier.

12) If *R* is an SQL-invoked function or if *R* is an SQL-invoked procedure and the descriptor of *R* includes an indication that a new savepoint level is to be established when *R* is invoked, then the current savepoint level is destroyed.

13) *CSC* becomes the current SQL-session context.

Conformance Rules

1) Without Feature S023, “Basic structured types”, conforming SQL language shall not contain a <generalized expression>.

2) Without Feature S201, “SQL-invoked routines on arrays”, conforming SQL language shall not contain an <SQL argument> whose declared type is an array type.

3) Without Feature S202, “SQL-invoked routines on multisets”, conforming SQL language shall not contain an <SQL argument> whose declared type is a multiset type.

4) Without Feature B033, “Untyped SQL-invoked function arguments”, conforming SQL language shall not contain a <routine invocation> that is not simply contained in a <call statement> that simply contains an <SQL argument> that is a <dynamic parameter specification>.

10.5 <character set specification>

Function

Identify a character set.

Format

```
<character set specification> ::=
    <standard character set name>
  | <implementation-defined character set name>
  | <user-defined character set name>

<standard character set name> ::= <character set name>

<implementation-defined character set name> ::= <character set name>

<user-defined character set name> ::= <character set name>
```

Syntax Rules

1) The <standard character set name>s and <implementation-defined character set name>s that are supported are implementation-defined.

2) A character set identified by a <standard character set name>, or by an <implementation-defined character set name> has associated with it a privilege descriptor that was effectively defined by the <grant statement>

```
GRANT USAGE ON CHARACTER SET CS TO PUBLIC
```

where *CS* is the <character set name> contained in the <character set specification>. The grantor of the privilege descriptor is set to the special grantor value "_SYSTEM".

3) The <standard character set name>s shall include SQL_CHARACTER and those character sets specified in Subclause 4.2.7, "Character sets", as defined by this and other standards.

4) The <implementation-defined character set name>s shall include SQL_TEXT and SQL_IDENTIFIER.

5) Let *C* be the <character set name> contained in the <character set specification>. The schema identified by the explicit or implicit qualifier of the <character set name> shall include the descriptor of *C*.

6) If a <character set specification> is not contained in a <schema definition>, then the <character set name> immediately contained in the <character set definition> shall contain an explicit <schema name> that is not equivalent to INFORMATION_SCHEMA.

Access Rules

1) Case:

   a) If <character set specification> is contained, without an intervening <SQL routine spec> that specifies SQL SECURITY INVOKER, in an <SQL schema statement>, then the applicable privileges of the <authorization identifier> that owns the containing schema shall include USAGE on *C*.

b) Otherwise, the current privileges shall include USAGE on *C*.

NOTE 226 — “applicable privileges” and “current privileges” are defined in Subclause 12.3, “<privileges>”.

General Rules

1) A <character set specification> identifies a character set. Let the identified character set be *CS*.

2) A <standard character set name> specifies the name of a character set that is defined by a national or international standard. The character repertoire of *CS* is defined by the standard defining the character set identified by that <standard character set name>. The default collation of the character set is defined by the order of the characters in the standard and has the PAD SPACE characteristic.

3) An <implementation-defined character set name> specifies the name of a character set that is implementation-defined. The character repertoire of *CS* is implementation-defined. The default collation of the character set and whether the collation has the NO PAD characteristic or the PAD SPACE characteristic is implementation-defined.

4) A <user-defined character set name> identifies a character set whose descriptor is included in some schema whose <schema name> is not equivalent to INFORMATION_SCHEMA.

NOTE 227 — The default collation of the character set is defined as in Subclause 11.31, “<character set definition>”.

5) There is a character set descriptor for every character set that can be specified by a <character set specification>.

Conformance Rules

1) Without Feature F461, “Named character sets”, conforming SQL language shall not contain a <character set specification>.

10.6 <specific routine designator>

Function

Specify an SQL-invoked routine.

Format

```
<specific routine designator> ::=
    SPECIFIC <routine type> <specific name>
  | <routine type> <member name> [ FOR <schema-resolved user-defined type name> ]

<routine type> ::=
    ROUTINE
  | FUNCTION
  | PROCEDURE
  | [ INSTANCE | STATIC | CONSTRUCTOR ] METHOD

<member name> ::= <member name alternatives> [ <data type list> ]

<member name alternatives> ::=
    <schema qualified routine name>
  | <method name>

<data type list> ::=
    <left paren> [ <data type> [ { <comma> <data type> }... ] ] <right paren>
```

Syntax Rules

1) If a <specific name> *SN* is specified, then the <specific routine designator> shall identify an SQL-invoked routine whose <specific name> is *SN*.

2) If <routine type> specifies METHOD and none of INSTANCE, STATIC, or CONSTRUCTOR is specified, then INSTANCE is implicit.

3) If a <member name> *MN* is specified, then:

   a) If <schema-resolved user-defined type name> is specified, then <routine type> shall specify METHOD. If METHOD is specified, then <schema-resolved user-defined type name> shall be specified.

   b) Case:

      i) If <routine type> specifies METHOD, then <method name> shall be specified. Let *SCN* be the implicit or explicit <schema name> of <schema-resolved user-defined type name>, let *METH* be the <method name>, and let *RN* be *SCN*.*METH*.

      ii) Otherwise, <schema qualified routine name> shall be specified. Let *RN* be the <schema qualified routine name> of *MN* and let *SCN* be the <schema name> of *MN*.

   c) Case:

      i) If *MN* contains a <data type list>, then:

1) If <routine type> specifies FUNCTION, then there shall be exactly one SQL-invoked regular function in the schema identified by *SCN* whose <schema qualified routine name> is *RN* such that for all *i* the Syntax Rules of Subclause 9.16, "Data type identity", when applied with the declared type of its *i*-th SQL parameter and the *i*-th <data type> in the <data type list> of *MN*, are satisfied. The <specific routine designator> identifies that SQL-invoked function.

2) If <routine type> specifies PROCEDURE, then there shall be exactly one SQL-invoked procedure in the schema identified by *SCN* whose <schema qualified routine name> is *RN* such that for all *i* the Syntax Rules of Subclause 9.16, "Data type identity", when applied with the declared type of its *i*-th SQL parameter and the *i*-th <data type> in the <data type list> of *MN*, are satisfied. The <specific routine designator> identifies that SQL-invoked procedure.

3) If <routine type> specifies METHOD, then

   Case:

   A) If STATIC is specified, then there shall be exactly one static SQL-invoked method of the type identified by <schema-resolved user-defined type name> whose <method name> is *METH*, such that for all *i* the Syntax Rules of Subclause 9.16, "Data type identity", when applied with the declared data type of its *i*-th SQL parameter and the *i*-th <data type> in the <data type list> of *MN*, are satisfied. The <specific routine designator> identifies that static SQL-invoked method.

   B) If CONSTRUCTOR is specified, then there shall be exactly one SQL-invoked constructor method of the type identified by <schema-resolved user-defined type name> whose <method name> is *METH*, such that for all *i* the Syntax Rules of Subclause 9.16, "Data type identity", when applied with the declared data type of its *i*-th SQL parameter in the unaugmented <SQL parameter declaration list> amd the *i*-th <data type> in the <data type list> of *MN*, are satisfied. The <specific routine designator> identifies that SQL-invoked constructor method.

   C) Otherwise, there shall be exactly one instance SQL-invoked method of the type identified by <schema-resolved user-defined type name> whose <method name> is *METH*, such that for all *i* the Syntax Rules of Subclause 9.16, "Data type identity", when applied with the declared data type of its *i*-th SQL parameter in the unaugmented <SQL parameter declaration list> and the *i*-th <data type> in the <data type list> of *MN*, are satisfied. The <specific routine designator> identifies that instance SQL-invoked method.

4) If <routine type> specifies ROUTINE, then there shall be exactly one SQL-invoked routine in the schema identified by *SCN* whose <schema qualified routine name> is *RN* such that for all *i* the Syntax Rules of Subclause 9.16, "Data type identity", when applied with the declared type of its *i*-th SQL parameter and the *i*-th <data type> in the <data type list> of *MN*, are satisfied. The <specific routine designator> identifies that SQL-invoked routine.

ii) Otherwise:

1) If <routine type> specifies FUNCTION, then there shall be exactly one SQL-invoked function in the schema identified by *SCN* whose <schema qualified routine name> is *RN*. The <specific routine designator> identifies that SQL-invoked function.

2) If <routine type> specifies PROCEDURE, then there shall be exactly one SQL-invoked procedure in the schema identified by *SCN* whose <schema qualified routine name> is *RN*. The <specific routine designator> identifies that SQL-invoked procedure.

3) If <routine type> specifies METHOD, then

   Case:

   A) If STATIC is specified, then there shall be exactly one static SQL-invoked method of the user-defined type identified by <schema-resolved user-defined type name> whose <method name> is *METH*. The <specific routine designator> identifies that static SQL-invoked method.

   B) If CONSTRUCTOR is specified, then there shall be exactly one SQL-invoked constructor method of the user-defined type identified by <schema-resolved user-defined type name> whose <method name> is *METH*. The <specific routine designator> identifies that SQL-invoked constructor method.

   C) Otherwise, there shall be exactly one instance SQL-invoked method of the user-defined type identified by <schema-resolved user-defined type name> whose <method name> is *METH*. The <specific routine designator> identifies that instance SQL-invoked method.

4) If <routine type> specifies ROUTINE, then there shall be exactly one SQL-invoked routine in the schema identified by *SCN* whose <schema qualified routine name> is *RN*. The <specific routine designator> identifies that SQL-invoked routine.

4) If FUNCTION is specified, then the SQL-invoked routine that is identified shall be an SQL-invoked regular function. If PROCEDURE is specified, then the SQL-invoked routine that is identified shall be an SQL-invoked procedure. If STATIC METHOD is specified, then the SQL-invoked routine that is identified shall be a static SQL-invoked method. If CONSTRUCTOR METHOD is specified, then the SQL-invoked routine shall be an SQL-invoked constructor method. If INSTANCE METHOD is specified or implicit, then the SQL-invoked routine shall be an instance SQL-invoked method. If ROUTINE is specified, then the SQL-invoked routine that is identified is either an SQL-invoked function or an SQL-invoked procedure.

Access Rules

None.

General Rules

None.

Conformance Rules

1) Without Feature S024, "Enhanced structured types", conforming SQL language shall not contain a <specific routine designator> that contains a <routine type> that immediately contains METHOD.

10.7 <collate clause>

Function

Specify a default collation.

Format

```
<collate clause> ::= COLLATE <collation name>
```

Syntax Rules

1) Let *C* be the <collation name> contained in the <collate clause>. The schema identified by the explicit or implicit qualifier of the <collation name> shall include the descriptor of *C*.

Access Rules

1) Case:

   a) If <collate clause> is contained, without an intervening <SQL routine spec> that specifies SQL SECURITY INVOKER, in an <SQL schema statement>, then the applicable privileges of the <authorization identifier> that owns the containing schema shall include USAGE on *C*.

   b) Otherwise, the current privileges shall include USAGE on *C*.

   NOTE 228 — "applicable privileges" and "current privileges" are defined in Subclause 12.3, "<privileges>".

General Rules

None.

Conformance Rules

1) Without Feature F690, "Collation support", conforming SQL language shall not contain a <collate clause>.

10.8 <constraint name definition> and <constraint characteristics>

Function

Specify the name of a constraint and its characteristics.

Format

```
<constraint name definition> ::= CONSTRAINT <constraint name>

<constraint characteristics> ::=
    <constraint check time> [ [ NOT ] DEFERRABLE ]
  | [ NOT ] DEFERRABLE [ <constraint check time> ]

<constraint check time> ::=
    INITIALLY DEFERRED
  | INITIALLY IMMEDIATE
```

Syntax Rules

1) If a <constraint name definition> is contained in a <schema definition>, and if the <constraint name> contains a <schema name>, then that <schema name> shall be equivalent to the specified or implicit <schema name> of the containing <schema definition>.
2) The <qualified identifier> of <constraint name> shall not be equivalent to the <qualified identifier> of the <constraint name> of any other constraint defined in the same schema.
3) If <constraint check time> is not specified, then INITIALLY IMMEDIATE is implicit.
4) Case:
    a) If INITIALLY DEFERRED is specified, then:
        i) NOT DEFERRABLE shall not be specified.
        ii) If DEFERRABLE is not specified, then DEFERRABLE is implicit.
    b) If INITIALLY IMMEDIATE is specified or implicit and neither DEFERRABLE nor NOT DEFERRABLE is specified, then NOT DEFERRABLE is implicit.

Access Rules

None.

General Rules

1) A <constraint name> identifies a constraint. Let the identified constraint be *C*.
2) If NOT DEFERRABLE is specified, then *C* is not deferrable; otherwise it is deferrable.

3) If <constraint check time> is INITIALLY DEFERRED, then the initial constraint mode for *C* is *deferred*; otherwise, the initial constraint mode for *C* is *immediate*.

4) If, on completion of any SQL-statement, the constraint mode of any constraint is immediate, then that constraint is effectively checked.

   NOTE 229 — This includes the cases where SQL-statement is a <set constraints mode statement>, a <commit statement>, or the statement that causes a constraint with a constraint mode of *initially immediate* to be created.

5) When a constraint is effectively checked, if the constraint is not satisfied, then an exception condition is raised: *integrity constraint violation*. If this exception condition is raised as a result of executing a <commit statement>, then SQLSTATE is not set to *integrity constraint violation*, but is set to *transaction rollback — integrity constraint violation* (see the General Rules of Subclause 16.6, "<commit statement>").

Conformance Rules

1) Without Feature F721, "Deferrable constraints", conforming SQL language shall not contain a <constraint characteristics>.

   NOTE 230 — This means that INITIALLY IMMEDIATE NOT DEFERRABLE is implicit.

2) Without Feature F491, "Constraint management", conforming SQL language shall not contain a <constraint name definition>.

10.9 <aggregate function>

Function

Specify a value computed from a collection of rows.

Format

```
<aggregate function> ::=
    COUNT <left paren> <asterisk> <right paren> [ <filter clause> ]
  | <general set function> [ <filter clause> ]
  | <binary set function> [ <filter clause> ]
  | <ordered set function> [ <filter clause> ]

<general set function> ::=
    <set function type> <left paren> [ <set quantifier> ]
    <value expression> <right paren>

<set function type> ::= <computational operation>

<computational operation> ::=
    AVG
  | MAX
  | MIN
  | SUM
  | EVERY
  | ANY
  | SOME
  | COUNT
  | STDDEV_POP
  | STDDEV_SAMP
  | VAR_SAMP
  | VAR_POP
  | COLLECT
  | FUSION
  | INTERSECTION

<set quantifier> ::=
    DISTINCT
  | ALL

<filter clause> ::=
    FILTER <left paren> WHERE <search condition> <right paren>

<binary set function> ::=
    <binary set function type> <left paren> <dependent variable expression> <comma>
    <independent variable expression> <right paren>

<binary set function type> ::=
    COVAR_POP
  | COVAR_SAMP
  | CORR
  | REGR_SLOPE
  | REGR_INTERCEPT
  | REGR_COUNT
  | REGR_R2
  | REGR_AVGX
  | REGR_AVGY
  | REGR_SXX
  | REGR_SYY
  | REGR_SXY

<dependent variable expression> ::= <numeric value expression>

<independent variable expression> ::= <numeric value expression>

<ordered set function> ::=
    <hypothetical set function>
  | <inverse distribution function>

<hypothetical set function> ::=
    <rank function type> <left paren>
    <hypothetical set function value expression list> <right paren>
    <within group specification>

<within group specification> ::=
    WITHIN GROUP <left paren> ORDER BY <sort specification list> <right paren>

<hypothetical set function value expression list> ::=
    <value expression> [ { <comma> <value expression> }... ]

<inverse distribution function> ::=
    <inverse distribution function type> <left paren>
    <inverse distribution function argument> <right paren>
    <within group specification>

<inverse distribution function argument> ::= <numeric value expression>

<inverse distribution function type> ::=
    PERCENTILE_CONT
  | PERCENTILE_DISC
```

Syntax Rules

1) Let *AF* be the <aggregate function>.

2) If STDDEV_POP, STDDEV_SAMP, VAR_POP, or VAR_SAMP is specified, then <set quantifier> shall not be specified.

3) If <general set function> is specified and <set quantifier> is not specified, then ALL is implicit.

4) The argument source of an <aggregate function> is

   Case:

   a) If *AF* is immediately contained in a <set function specification>, then a table or group of a grouped table as specified in Subclause 7.10, "<having clause>", and Subclause 7.12, "<query specification>".

   b) Otherwise, the collection of rows in the current row's window frame defined by the window structure descriptor identified by the <window function> that simply contains *AF*, as defined in Subclause 7.11, "<window clause>".

5) Let *T* be the argument source of *AF*.

6) If COUNT is specified, then the declared type of the result is an implementation-defined exact numeric type scale of 0 (zero).

7) If <general set function> is specified, then:

   a) The <value expression> *VE* shall not contain a <window function>.

   b) Let *DT* be the declared type of the <value expression>.

   c) If *AF* specifies a <general set function> whose <set quantifier> is DISTINCT, then *VE* is an operand of a grouping operation. The Syntax Rules of Subclause 9.10, "Grouping operations", apply.

   d) If *AF* specifies a <set function type> that is MAX or MIN, then *VE* is an operand of an ordering operation. The Syntax Rules of Subclause 9.12, "Ordering operations", apply.

   e) If EVERY, ANY, or SOME is specified, then *DT* shall be boolean and the declared type of the result is boolean.

   f) If MAX or MIN is specified, then the declared type of the result is *DT*.

   g) If SUM or AVG is specified, then:

      i) *DT* shall be a numeric type or an interval type.

      ii) If SUM is specified and *DT* is exact numeric with scale *S*, then the declared type of the result is an implementation-defined exact numeric type with scale *S*.

      iii) If AVG is specified and *DT* is exact numeric, then the declared type of the result is an implementation-defined exact numeric type with precision not less than the precision of *DT* and scale not less than the scale of *DT*.

      iv) If *DT* is approximate numeric, then the declared type of the result is an implementation-defined approximate numeric with precision not less than the precision of *DT*.

      v) If *DT* is interval, then the declared type of the result is interval with the same precision as *DT*.

   h) If VAR_POP or VAR_SAMP is specified, then the declared type of the result is an implementation-defined approximate numeric type. If *DT* is an approximate numeric type, then the precision of the result is not less than the precision of *DT*.

   i) `STDDEV_POP(X)` is equivalent to `SQRT(VAR_POP(X))`.

   j) `STDDEV_SAMP(X)` is equivalent to `SQRT(VAR_SAMP(X))`.

   k) If COLLECT is specified, then the declared type of the result is *DT* MULTISET.

   l) `COLLECT (X)` is equivalent to `FUSION (MULTISET [X])`.

   m) If FUSION is specified, then *DT* shall be a multiset type, and DISTINCT shall not be specified. The declared type of the result is *DT*.

n) If INTERSECTION is specified, then *DT* shall be a multiset type, and DISTINCT shall not be specified. *VE* is a multiset operand of a multiset element grouping operation, and the Syntax Rules of Subclause 9.11, "Multiset element grouping operations", apply. The declared type of the result is *DT*.

8) A <filter clause> shall not contain a <subquery>, a <window function>, or an outer reference.

9) If <binary set function> is specified, then:

a) The <dependent variable expression> *DVE* and the <independent variable expression> *IVE* shall not contain a <window function>.

b) Let *DTDVE* be the declared type of *DVE* and let *DTIVE* be the declared type of *IVE*.

c) Case:

i) The declared type of REGR_COUNT is an implementation-defined exact numeric type with scale of 0 (zero).

ii) Otherwise, the declared type of the result is an implementation-defined approximate numeric type. If *DTDVE* is an approximate numeric type, then the precision of the result is not less than the precision of *DTDVE*. If *DTIVE* is an approximate numeric type, then the precision of the result is not less than the precision of *DTIVE*.

10) If <hypothetical set function> is specified, then:

a) The <hypothetical set function> shall not contain a <window function>, a <set function specification>, or a <subquery>.

b) The number of <value expression>s simply contained in <hypothetical set function value expression list> shall be the same as the number of <sort key>s simply contained in the <sort specification list>.

c) For each <value expression> *HSFVE* simply contained in the <hypothetical set function value expression list>, let *SK* be the corresponding <sort key> simply contained in the <sort specification list>.

Case:

i) If the declared type of *HSFVE* is a character string type, then the declared type of *SK* shall be a character string type with the same character repertoire as that of *HSFVE*. The collation is determined by applying Subclause 9.13, "Collation determination", with operands *HSFVE* and *SK*.

ii) Otherwise the declared types of *HSFVE* and *SK* shall be compatible.

d) Case:

i) If RANK or DENSE_RANK is specified, then the declared type of the result is exact numeric with implementation-defined precision and with scale 0 (zero).

ii) Otherwise, the declared type of the result is approximate numeric with implemenation-defined precision.

11) If <inverse distribution function> is specified, then:

a) The <within group specification> shall contain a single <sort specification>.

b) The <inverse distribution function> shall not contain a <window function>, a <set function specification>, or a <subquery>.

c) Let *DT* be the declared type of the <value expression> simply contained in the <sort specification>.

d) If PERCENTILE_CONT is specified, then *DT* shall be numeric or interval.

e) The declared type of the result is

Case:

i) If *DT* is numeric, then approximate numeric with implementation-defined precision.

ii) If *DT* is interval, then *DT*.

Access Rules

None.

General Rules

1) If, during the computation of the result of *AF*, an intermediate result is not representable in the declared type of the site that contains that intermediate result, then

Case:

a) If the most specific type of the result of *AF* is an interval type, then an exception condition is raised: *data exception — interval value out of range.*

b) If the most specific type of the result of *AF* is a multiset type, then an exception condition is raised: *data exception — multiset value overflow.*

c) Otherwise, an exception condition is raised: *data exception — numeric value out of range.*

2) Case:

a) If <filter clause> is specified, then the <search condition> is applied to each row of *T*. Let *T1* be the collection of rows of *T* for which the result of the <search condition> is *True*.

b) Otherwise, let *T1* be *T*.

3) If COUNT(*) is specified, then the result is the cardinality of *T1*.

4) If <general set function> is specified, then:

a) Let *TX* be the single-column table that is the result of applying the <value expression> to each row of *T1* and eliminating null values. If one or more null values are eliminated, then a completion condition is raised: *warning — null value eliminated in set function.*

b) Case:

i) If DISTINCT is specified, then let *TXA* be the result of eliminating redundant duplicate values from *TX*, using the comparison rules specified in Subclause 8.2, "<comparison predicate>", to identify the redundant duplicate values.

ii) Otherwise, let *TXA* be *TX*.

c) Let *N* be the cardinality of *TXA*.

d) Case:

i) If COUNT is specified, then the result is *N*.

ii) If *TXA* is empty, then the result is the null value.

iii) If AVG is specified, then the result is the average of the values in *TXA*.

iv) If MAX or MIN is specified, then the result is respectively the maximum or minimum value in *TXA*. These results are determined using the comparison rules specified in Subclause 8.2, "<comparison predicate>". If *DT* is a user-defined type and the comparison of two values in *TXA* results in *Unknown*, then the maximum or minimum of *TXA* is implementation-dependent.

v) If SUM is specified, then the result is the sum of the values in *TXA*. If the sum is not within the range of the declared type of the result, then an exception condition is raised: *data exception — numeric value out of range*.

vi) If EVERY is specified, then

Case:

1) If the value of some element of *TXA* is *False*, then the result is *False*.

2) Otherwise, the result is *True*.

vii) If ANY or SOME is specified, then

Case:

1) If the value of some element of *TXA* is *True*, then the result is *True*.

2) Otherwise, the result is *False*.

viii) If VAR_POP or VAR_SAMP is specified, then let *S1* be the sum of values in the column of *TXA*, and *S2* be the sum of the squares of the values in the column of *TXA*.

1) If VAR_POP is specified, then the result is `(S2-S1*S1/N)/N`.

2) If VAR_SAMP is specified, then

Case:

A) If *N* is 1 (one), then the result is the null value.

B) Otherwise, the result is `(S2-S1*S1/N)/(N-1)`

ix) If FUSION is specified, then the result is the multiset *M* such that for each value *V* in the element type of *DT*, including the null value, the number of elements of *M* that are identical to *V* is the sum of the number of identical copies of *V* in the multisets that are the values of the column in each row of *TXA*.

x) If INTERSECTION is specified, then the result is a multiset *M* such that for each value *V* in the element type of *DT*, including the null value, the number of duplicates of *V* in *M* is the minimum of the number of duplicates of *V* in the multisets that are the values of the column in each row of *TXA*.

NOTE 231 — This rule says "the result is a multiset" rather than "the result is the multiset" because the precise duplicate values are not specified. Thus this calculation is non-deterministic for certain element types, namely those based on character string, datetime with time zone and user-defined types.

5) If <binary set function type> is specified, then:

a) Let *TXA* be the two-column table that is the result of applying the <dependent variable expression> and the <independent variable expression> to each row of *T1* and eliminating each row in which either <dependent variable expression> or <independent variable expression> is the null value. If one or more null values are eliminated, then a completion condition is raised: *warning — null value eliminated in set function*.

b) Let *N* be the cardinality of *TXA*, let *SUMX* be the sum of the column of values of <independent variable expression>, let *SUMY* be the sum of the column of values of <dependent variable expression>, let *SUMX2* be the sum of the squares of values in the <independent variable expression> column, let *SUMY2* be the sum of the squares of values in the <dependent variable expression> column, and let *SUMXY* be the sum of the row-wise products of the value in the <independent variable expression> column times the value in the <dependent variable expression> column.

c) Case:

i) If REGR_COUNT is specified, then the result is *N*.

ii) If *N* is 0 (zero), then the result is the null value.

iii) If REGR_SXX is specified, then the result is `(SUMX2-SUMX*SUMX/N)`.

iv) If REGR_SYY is specified, then the result is `(SUMY2-SUMY*SUMY/N)`.

v) If REGR_SXY is specified, then the result is `(SUMXY-SUMX*SUMY/N)`.

vi) If REGR_AVGX is specified, then the result is `SUMX/N`.

vii) If REGR_AVGY is specified, then the result is `SUMY/N`.

viii) If COVAR_POP is specified, then the result is `(SUMXY-SUMX*SUMY/N)/N`.

ix) If COVAR_SAMP is specified, then

Case:

1) If *N* is 1 (one), then the result is the null value.

2) Otherwise, the result is `(SUMXY-SUMX*SUMY/N)/(N-1)`

x) If CORR is specified, then

Case:

1) If *N*SUMX2* equals *SUMX*SUMX*, then the result is the null value.

NOTE 232 — In this case, all remaining values of <independent variable expression> are equal, and consequently the <independent variable expression> does not correlate with the <dependent variable expression>.

2) If *N*SUMY2* equals *SUMY*SUMY*, then the result is the null value.

NOTE 233 — In this case, all remaining values of <dependent variable expression> are equal, and consequently the <dependent variable expression> does not correlate with the <independent variable expression>.

3) Otherwise, the result is `SQRT(POWER(`*N*`*`*SUMXY*`-`*SUMX*`*`*SUMY*`,2) / ((`*N*`*`*SUMX2*`-`*SUMX*`*`*SUMX*`) * (`*N*`*`*SUMY2*`-`*SUMY*`*`*SUMY*`)))`. If the exponent of the approximate mathematical result of the operation is not within the implementation-defined exponent range for the result data type, then the result is the null value.

xi) If REGR_R2 is specified, then

Case:

1) If *N*SUMX2* equals *SUMX*SUMX*, then the result is the null value.

NOTE 234 — In this case, all remaining values of <independent variable expression> are equal, and consequently the least-squares fit line would be vertical, or, if *N* = 1 (one), there is no uniquely determined least-squares-fit line.

2) If *N*SUMY2* equals *SUMY*SUMY*, then the result is 1 (one).

NOTE 235 — In this case, all remaining values of <dependent variable expression> are equal, and consequently the least-squares fit line is horizontal.

3) Otherwise, the result is `POWER(`*N*`*`*SUMXY*`-`*SUMX*`*`*SUMY*`,2) / ((`*N*`*`*SUMX2*`-`*SUMX*`*`*SUMX*`) * (`*N*`*`*SUMY2*`-`*SUMY*`*`*SUMY*`))`. If the exponent of the approximate mathematical result of the operation is not within the implementation-defined exponent range for the result data type, then the result is the null value.

xii) If REGR_SLOPE(*Y*, *X*) is specified, then

Case:

1) If *N*SUMX2* equals *SUMX*SUMX*, then the result is the null value.

NOTE 236 — In this case, all remaining values of <independent variable expression> are equal, and consequently the least-squares fit line would be vertical, or, if *N* = 1 (one), then there is no uniquely determined least-squares-fit line.

2) Otherwise, the result is `(`*N*`*`*SUMXY*`-`*SUMX*`*`*SUMY*`) / (`*N*`*`*SUMX2*`-`*SUMX*`*`*SUMX*`)`. If the exponent of the approximate mathematical result of the operation is not within the implementation-defined exponent range for the result data type, then the result is the null value.

xiii) If REGR_INTERCEPT is specified, then

Case:

1) If *N*SUMX2* equals *SUMX*SUMX*, then the result is the null value.

NOTE 237 — In this caes, all remaining values of <independent variable expression> are equal, and consequently the least-squares fit line would be vertical, or, if *N* = 1 (one), then there is no uniquely determined least-squares-fit line.

2) Otherwise, the result is `(`*SUMY*`*`*SUMX2*`-`*SUMX*`*`*SUMXY*`) / (`*N*`*`*SUMX2*`-`*SUMX*`*`*SUMX*`)`. If the exponent of the approximate mathematical result of the operation is not within the implementation-defined exponent range for the result data type, then the result is the null value.

6) If <hypothetical set function> is specified, then

a) Let *WIFT* be the <rank function type>.

b) Let *TNAME* be an implementation-dependent name for *T1*.

c) Let *K* be the number of <value expression>s simply contained in <hypothetical set function value expression list>.

d) Let VE_1, ..., VE_K be the <value expression>s simply contained in the <hypothetical set function value expression list>.

e) Let *WIFTVAL*, *MARKER* and CN_1, ..., CN_K be distinct implementation-dependent column names.

f) Let SP_1, ..., SP_K be the <sort specification>s simply contained in the <sort specification list>. For each *i*, let WSP_i be the <sort specification> obtained from SP_i by replacing the <sort key> with CN_i.

g) The result is the result of the <scalar subquery>

```
( SELECT WIFTVAL
 FROM ( SELECT MARKER, WIFT() OVER
            ( ORDER BY WSP1, ..., WSPK )
        FROM ( SELECT 0, SK1, ..., SKK
               FROM TNAME
                 UNION ALL
                    VALUES (1, VE1, ..., VEK) )
                 AS TXNAME (MARKER, CN1, ..., CNK )
      ) AS TEMPTABLE (MARKER, WIFTVAL)
 WHERE MARKER = 1 )
```

7) If <inverse distribution function> is specified, then

a) Let *NVE* be the value of the <inverse distribution function argument>.

b) If *NVE* is the null value, then the result is the null value.

c) If *NVE* is less than 0 (zero) or greater than 1 (one), then an exception condition is raised: *data exception — numeric value out of range.*

d) Let *TXA* be the single-column table that is the result of applying the <value expression> simply contained in the <sort specification> to each row of *T1* and eliminating null values. If one or more null values are eliminated, then a completion condition is raised: *warning — null value eliminated in set function.* *TXA* is ordered by the <sort specification> as specified in the General Rules of Subclause 10.10, "<sort specification list>".

e) Let *TXANAME* be an implementation-dependent name for *TXA*.

f) Let *TXCOLNAME* be an implementation-dependent column name for the column of *TXA*.

g) Let *WSP* be obtained from the <sort specification> by replacing the <sort key> with *TXCOLNAME.*

h) Case:

i) If PERCENTILE_CONT is specified, then:

1) Let *ROW0* be the greatest exact numeric value with scale 0 (zero) that is less than or equal to *NVE**(*N*–1). Let *ROWLIT0* be a <literal> representing *ROW0.*

2) Let *ROW1* be the least exact numeric value with scale 0 (zero) that is greater than or equal to *NVE**(*N*–1). Let *ROWLIT1* be a <literal> representing *ROW1*.

3) Let *FACTOR* be an <approximate numeric literal> representing *NVE**(*N*–1)–*ROW0*.

4) The result is the result of the <scalar subquery>

```
( WITH TEMPTABLE(X, Y) AS
      ( SELECT ROW_NUMBER()
                   OVER (ORDER BY WSP) - 1,
                 TXCOLNAME
          FROM TXANAME )
 SELECT CAST ( T0.Y + FACTOR * (T1.Y - T0.Y) AS DT )
 FROM TEMPTABLE T0, TEMPTABLE T1
 WHERE T0.ROWNUMBER = ROWLIT0
   AND T1.ROWNUMBER = ROWLIT1 )
```

NOTE 238 — Although ROW_NUMBER is nondeterministic, the values of T0.Y and T1.Y are determined by this expression. Note that the only column of *TXA* is completely ordered by *WSP*. If *NVE**(*N*–1) is a whole number, then the rows selected from T0 and T1 are the same and the result is just T0.Y. Otherwise, the subquery performs a linear interpolation between the two consecutive values whose row numbers in the ordered set, seen as proportions of the whole, bound the argument of the PERCENTILE_CONT operator.

ii) If PERCENTILE_DISC is specified, then

1) If the <ordering specification> simply contained in *WSP* is DESC, then let *MAXORMIN* be MAX; otherwise let *MAXORMIN* be MIN.

2) Let *NVELIT* be a <literal> representing the value of *NVE*.

3) The result is the result of the <scalar subquery>

```
( SELECT MAXORMIN (TXCOLNAME)
 FROM ( SELECT TXCOLNAME,
                 CUME_DIST() OVER (ORDER BY WSP)
 FROM TXANAME ) AS TEMPTABLE (TXCOLNAME, CUMEDIST)
 WHERE CUMEDIST >= NVELIT )
```

Conformance Rules

1) Without Feature T031, "BOOLEAN data type", conforming SQL language shall not contain a <computational operation> that immediately contains EVERY, ANY, or SOME.

2) Without Feature F561, "Full value expressions", or Feature F801, "Full set function", conforming SQL language shall not contain a <general set function> that immediately contains DISTINCT and contains a <value expression> that is not a column reference.

3) Without Feature F441, "Extended set function support", conforming SQL language shall not contain a <general set function> that contains a <computational operation> that immediately contains COUNT and does not contain a <set quantifier> that immediately contains DISTINCT.

4) Without Feature F441, "Extended set function support", conforming SQL language shall not contain a <general set function> that does not contain a <set quantifier> that immediately contains DISTINCT and that contains a <value expression> that contains a column reference that does not reference a column of *T*.

5) Without Feature F441, "Extended set function support", conforming SQL language shall not contain a <binary set function> that does not contain either a <dependent variable expression> or an <independent variable expression> that contains a column reference that references a column of *T*.

6) Without Feature F441, "Extended set function support", conforming SQL language shall not contain a <value expression> simply contained in a <general set function> that contains a column reference that is an outer reference where the <value expression> is not a column reference.

7) Without Feature F441, "Extended set function support", conforming SQL language shall not contain a <numeric value expression> simply contained in a <dependent variable expression> or an <independent variable expression> that contains a column reference that is an outer reference and in which the <numeric value expression> is not a column reference.

8) Without Feature F441, "Extended set function support", conforming SQL language shall not contain a column reference contained in an <aggregate function> that contains a reference to a column derived from a <value expression> that generally contains an <aggregate function> *SFS2* without an intervening <routine invocation>.

9) Without Feature T621, "Enhanced numeric functions", conforming SQL language shall not contain a <computational operation> that immediately contains STDDEV_POP, STDDEV_SAMP, VAR_POP, or VAR_SAMP.

10) Without Feature T621, "Enhanced numeric functions", conforming SQL language shall not contain a <binary set function type>.

11) Without Feature T612, "Advanced OLAP operations", conforming SQL language shall not contain a <hypothetical set function> or an <inverse distribution function>.

12) Without Feature T612, "Advanced OLAP operations", conforming SQL language shall not contain a <filter clause>.

13) Without Feature S271, "Basic multiset support", conforming SQL language shall not contain a <computational operation> that immediately contains COLLECT.

14) Without Feature S275, "Advanced multiset support", conforming SQL language shall not contain a <computational operation> that immediately contains FUSION or INTERSECTION.

NOTE 239 — If INTERSECTION is specified, then the Conformance Rules of Subclause 9.11, "Multiset element grouping operations", also apply.

15) Without Feature T052, "MAX and MIN for row types", conforming SQL language shall not contain a <computational operation> that immediately contains MAX or MIN in which the declared type of the <value expression> is a row type.

NOTE 240 — If DISTINCT is specified, then the Conformance Rules of Subclause 9.10, "Grouping operations", also apply. If MAX or MIN is specified, then the Conformance Rules of Subclause 9.12, "Ordering operations", also apply.

16) Without Feature F442, "Mixed column references in set functions", conforming SQL language shall not contain a <hypothetical set function value expression list> or a <sort specification list> that simply contains a <value expression> that contains more than one column reference, one of which is an outer reference.

17) Without Feature F442, "Mixed column references in set functions", conforming SQL language shall not contain an <inverse distribution function> that contains an <inverse distribution function argument> or a <sort specification> that contains more than one column reference, one of which is an outer reference.

18) Without Feature F442, “Mixed column references in set functions”, conforming SQL language shall not contain an <aggregate function> that contains a <general set function> whose simply contained <value expression> contains more than one column reference, one of which is an outer reference.

19) Without Feature F442, “Mixed column references in set functions”, conforming SQL language shall not contain an <aggregate function> that contains a <binary set function> whose simply contained <dependent variable expression> or <independent variable expression> contains more than one column reference, one of which is an outer reference.

10.10 <sort specification list>

Function

Specify a sort order.

Format

```
<sort specification list> ::=
    <sort specification> [ { <comma> <sort specification> }... ]

<sort specification> ::=
    <sort key> [ <ordering specification> ] [ <null ordering> ]

<sort key> ::= <value expression>

<ordering specification> ::=
    ASC
  | DESC

<null ordering> ::=
    NULLS FIRST
  | NULLS LAST
```

Syntax Rules

1) Let *DT* be the declared type of the <value expression> simply contained in the <sort key> contained in a <sort specification>.

2) Each <value expression> simply contained in the <sort key> contained in a <sort specification> is an operand of an ordering operation. The Syntax Rules of Subclause 9.12, "Ordering operations", apply.

3) If <null ordering> is not specified, then an implementation-defined <null ordering> is implicit. The implementation-defined default for <null ordering> shall not depend on the context outside of <sort specification list>.

Access Rules

None.

General Rules

1) A <sort specification list> defines an ordering of rows, as follows:

   a) Let *N* be the number of <sort specification>s.

   b) Let K_i, 1 (one) $\leq i \leq N$, be the <sort key> contained in the *i*-th <sort specification>.

c) Each <sort specification> specifies the *sort direction* for the corresponding sort key K_i. If DESC is not specified in the i-th <sort specification>, then the sort direction for K_i is ascending and the applicable <comp op> is the <less than operator>. Otherwise, the sort direction for K_i is descending and the applicable <comp op> is the <greater than operator>.

d) Let P be any row of the collection of rows to be ordered, and let Q be any other row of the same collection of rows.

e) Let PV_i and QV_i be the values of K_i in P and Q, respectively. The relative position of rows P and Q in the result is determined by comparing PV_i and QV_i according to the rules of Subclause 8.2, "<comparison predicate>" where the <comp op> is the applicable <comp op> for K_i, with the following special treatment of null values.

Case:

i) If PV_i and QV_i are both null, then they are considered equal to each other.

ii) If PV_i is null and QV_i is not null, then

Case:

1) If NULLS FIRST is specified or implied, then PV_i <comp op> QV_i is considered to be *True*.

2) If NULLS LAST is specified or implied, then PV_i <comp op> QV_i is considered to be *False*.

iii) If PV_i is not null and QV_i is null, then

Case:

1) If NULLS FIRST is specified or implied, then PV_i <comp op> QV_i is considered to be *False*.

2) If NULLS LAST is specified or implied, then PV_i <comp op> QV_i is considered to be *True*.

f) PV_i is said to *precede* QV_i if the value of the <comparison predicate> "`PVi <comp op> QVi`" is *True* for the applicable <comp op>.

g) If PV_i and QV_i are not null and the result of "PV_i <comp op> QV_i" is *Unknown*, then the relative ordering of PV_i and QV_i is implementation-dependent.

h) The relative position of row P is before row Q if PV_n precedes QV_n for some n, 1 (one) $\leq n \leq N$, and PV_i is not distinct from QV_i for all $i < n$.

i) Two rows that are not distinct with respect to the <sort specification>s are said to be *peers* of each other. The relative ordering of peers is implementation-dependent.

Conformance Rules

1) Without Feature T611, "Elementary OLAP operations", conforming SQL language shall not contain a <null ordering>.

NOTE 241 — The Conformance Rules of Subclause 9.12, "Ordering operations", also apply.

11 Schema definition and manipulation

11.1 <schema definition>

Function

Define a schema.

Format

```
<schema definition> ::=
    CREATE SCHEMA <schema name clause>
    [ <schema character set or path> ]
    [ <schema element>... ]

<schema character set or path> ::=
    <schema character set specification>
  | <schema path specification>
  | <schema character set specification> <schema path specification>
  | <schema path specification> <schema character set specification>

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

<schema authorization identifier> ::= <authorization identifier>

<schema character set specification> ::=
    DEFAULT CHARACTER SET <character set specification>

<schema path specification> ::= <path specification>

<schema element> ::=
    <table definition>
  | <view definition>
  | <domain definition>
  | <character set definition>
  | <collation definition>
  | <transliteration definition>
  | <assertion definition>
  | <trigger definition>
  | <user-defined type definition>
  | <user-defined cast definition>
  | <user-defined ordering definition>
  | <transform definition>
  | <schema routine>
  | <sequence generator definition>
  | <grant statement>
  | <role definition>
```

Syntax Rules

1) If <schema name> is not specified, then a <schema name> equal to <schema authorization identifier> is implicit.
2) If AUTHORIZATION <schema authorization identifier> is not specified, then

   Case:

   a) If the <schema definition> is contained in an SQL-client module that has a <module authorization identifier> specified, then an <authorization identifier> equal to that <module authorization identifier> is implicit for the <schema definition>.
   b) Otherwise, an <authorization identifier> equal to the SQL-session user identifier is implicit.
3) The <unqualified schema name> of the explicit or implicit <schema name> shall not be equivalent to the <unqualified schema name> of the <schema name> of any other schema in the catalog identified by the <catalog name> of <schema name>.
4) If a <schema definition> is contained in an <externally-invoked procedure> in an <SQL-client module definition>, then the effective <schema authorization identifier> and <schema name> during processing of the <schema definition> are, respectively, the <schema authorization identifier> and <schema name> specified or implicit in the <schema definition>.

   NOTE 242 — Other SQL-statements executed in <externally-invoked procedure>s in the SQL-client module have the <module authorization identifier> and <schema name> specified or implicit for the SQL-client module.
5) If <schema character set specification> is not specified, then a <schema 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 <schema path specification> is not specified, then a <schema path specification> containing an implementation-defined <schema name list> that contains the <schema name> contained in <schema name clause> is implicit.
7) The explicit or implicit <catalog name> of each <schema name> contained in the <schema name list> of the <schema path specification> shall be equivalent to the <catalog name> of the <schema name> contained in the <schema name clause>.
8) The <schema name list> of the explicit or implicit <schema path specification> is used as the SQL-path of the schema. The SQL-path is used to effectively qualify unqualified <routine name>s that are immediately contained in <routine invocation>s that are contained in the <schema definition>.

   NOTE 243 — <routine name> is defined in Subclause 5.4, "Names and identifiers".

Access Rules

1) The privileges necessary to execute the <schema definition> are implementation-defined.

General Rules

1) A <schema definition> creates an SQL-schema *S* in a catalog. *S* includes:
   a) A schema name that is equivalent to the explicit or implicit <schema name>.
   b) A schema authorization identifier that is equivalent to the explicit or implicit <authorization identifier>.
   c) A schema character set name that is equivalent to the explicit or implicit <schema character set specification>.
   d) A schema SQL-path that is equivalent to the explicit or implicit <schema path specification>.
   e) The descriptor created by every <schema element> of the <schema definition>.
2) The owner of *S* is schema authorization identifier.
3) The explicit or implicit <character set specification> is used as the default character set used for all <column definition>s and <domain definition>s that do not specify an explicit character set.

Conformance Rules

1) Without Feature S071, "SQL paths in function and type name resolution", conforming SQL language shall not contain a <schema path specification>.
2) Without Feature F461, "Named character sets", conforming SQL language shall not contain a <schema character set specification>.
3) Without Feature F171, "Multiple schemas per user", conforming SQL language shall not contain a <schema name clause> that contains a <schema name>.

11.2 <drop schema statement>

Function

Destroy a schema.

Format

```
<drop schema statement> ::= DROP SCHEMA <schema name> <drop behavior>

<drop behavior> ::=
    CASCADE
  | RESTRICT
```

Syntax Rules

1) Let *S* be the schema identified by <schema name>.

2) *S* shall identify a schema in the catalog identified by the explicit or implicit <catalog name>.

3) If RESTRICT is specified, then *S* shall not contain any persistent base tables, global temporary tables, created local temporary tables, views, domains, assertions, character sets, collations, transliterations, triggers, user-defined types, SQL-invoked routines, sequence generators, or roles, and the <schema name> of *S* shall not be generally contained in the SQL routine body of any routine descriptor.

   NOTE 244 — If CASCADE is specified, then such objects will be dropped by the effective execution of the SQL schema manipulation statements specified in the General Rules of this Subclause.

Access Rules

1) The enabled authorization identifiers shall include the <authorization identifier> that owns the schema identified by the <schema name>.

General Rules

1) Let *T* be the <table name> included in the descriptor of any base table or temporary table included in *S*. The following <drop table statement> is effectively executed:

   ```
   DROP TABLE T CASCADE
   ```

2) Let *V* be the <table name> included in the descriptor of any view included in *S*. The following <drop view statement> is effectively executed:

   ```
   DROP VIEW V CASCADE
   ```

3) Let *D* be the <domain name> included in the descriptor of any domain included in *S*. The following <drop domain statement> is effectively executed:

   ```
   DROP DOMAIN D CASCADE
   ```

4) Let *A* be the <constraint name> included in the descriptor of any assertion included in *S*. The following <drop assertion statement> is effectively executed:

```
DROP ASSERTION A CASCADE
```

5) Let *CD* be the <collation name> included in the descriptor of any collation included in *S*. The following <drop collation statement> is effectively executed:

```
DROP COLLATION CD CASCADE
```

6) Let *TD* be the <transliteration name> included in the descriptor of any transliteration included in *S*. The following <drop transliteration statement> is effectively executed:

```
DROP TRANSLATION TD
```

7) Let *RD* be the <character set name> included in the descriptor of any character set included in *S*. The following <drop character set statement> is effectively executed:

```
DROP CHARACTER SET RD
```

8) Let *DT* be the <user-defined type name> included in the descriptor of any user-defined type included in *S*. The following <drop data type statement> is effectively executed:

```
DROP TYPE DT CASCADE
```

9) Let *TT* be the <trigger name> included in the descriptor of any trigger included in *S*. The following <drop trigger statement> is effectively executed:

```
DROP TRIGGER TT
```

10) For every SQL-invoked routine *R* whose descriptor is included in *S*, let *SN* be the <specific name> of *R*. The following <drop routine statement> is effectively executed for every *R*:

```
DROP SPECIFIC ROUTINE SN CASCADE
```

11) Let *R* be any SQL-invoked routine whose routine descriptor includes an SQL routine body that contains the <schema name> of *S*. Let *SN* be the <specific name> of *R*. The following <drop routine statement> is effectively executed without further Access Rule checking:

```
DROP SPECIFIC ROUTINE SN CASCADE
```

12) Let *RO* be the name included in the descriptor of any role included in *S*. The following <drop role statement> is effectively executed:

```
DROP ROLE RO CASCADE
```

13) Let *SEQN* be the sequence generator name included in the descriptor of any sequence generator included in *S*. The following <drop sequence generator statement> is effectively executed:

```
DROP SEQUENCE SEQN CASCADE
```

14) *S* is destroyed.

Conformance Rules

1) Without Feature F381, "Extended schema manipulation", conforming SQL language shall not contain a <drop schema statement>.

11.3 <table definition>

Function

Define a persistent base table, a created local temporary table, or a global temporary table.

Format

```
<table definition> ::=
    CREATE [ <table scope> ] TABLE <table name> <table contents source>
    [ ON COMMIT <table commit action> ROWS ]

<table contents source> ::=
    <table element list>
  | <typed table clause>
  | <as subquery clause>

<table scope> ::= <global or local> TEMPORARY

<global or local> ::=
    GLOBAL
  | LOCAL

<table commit action> ::=
    PRESERVE
  | DELETE

<table element list> ::=
    <left paren> <table element> [ { <comma> <table element> }... ] <right paren>

<table element> ::=
    <column definition>
  | <table constraint definition>
  | <like clause>

<typed table clause> ::=
    OF <path-resolved user-defined type name> [ <subtable clause> ]
    [ <typed table element list> ]

<typed table element list> ::=
    <left paren> <typed table element>
    [ { <comma> <typed table element> }... ] <right paren>

<typed table element> ::=
    <column options>
  | <table constraint definition>
  | <self-referencing column specification>

<self-referencing column specification> ::=
    REF IS <self-referencing column name> [ <reference generation> ]

<reference generation> ::=
    SYSTEM GENERATED
  | USER GENERATED
  | DERIVED

<self-referencing column name> ::= <column name>

<column options> ::= <column name> WITH OPTIONS <column option list>

<column option list> ::=
    [ <scope clause> ] [ <default clause> ] [ <column constraint definition>... ]

<subtable clause> ::= UNDER <supertable clause>

<supertable clause> ::= <supertable name>

<supertable name> ::= <table name>

<like clause> ::= LIKE <table name> [ <like options> ]

<like options> ::= <like option>...

<like option> ::=
    <identity option>
  | <column default option>
  | <generation option>

<identity option> ::=
    INCLUDING IDENTITY
  | EXCLUDING IDENTITY

<column default option> ::=
    INCLUDING DEFAULTS
  | EXCLUDING DEFAULTS

<generation option> ::=
    INCLUDING GENERATED
  | EXCLUDING GENERATED

<as subquery clause> ::=
    [ <left paren> <column name list> <right paren> ] AS <subquery>
    <with or without data>

<with or without data> ::=
    WITH NO DATA
  | WITH DATA
```

Syntax Rules

1) Let *T* be the table defined by the <table definition> *TD*. Let *TN* be the <table name> simply contained in *TD*.

2) If a <table definition> is contained in a <schema definition> *SD* and *TN* contains a <local or schema qualifier>, then that <local or schema qualifier> shall be equivalent to the implicit or explicit <schema name> of *SD*.

3) The schema identified by the explicit or implicit schema name of *TN* shall not include a table descriptor whose table name is *TN*.

4) If the <table definition> is contained in a <schema definition>, then let *A* be the explicit or implicit <authorization identifier> of the <schema definition>. Otherwise, let *A* be the <authorization identifier> that owns the schema identified by the implicit or explicit <schema name> of *TN*.

5) If <table element> *TEL* is specified, then:

   a) *TEL* shall contain at least one <column definition> or <like clause>.

   b) For each <like clause> *LC* that is directly contained in *TEL*:

      i) Let *LT* be the table identified by the <table name> contained in *LC*.

      ii) If *LT* is a viewed table, then <like options> shall not be specified.

      iii) Let *D* be the degree of *LT*. For *i*, 1 (one) ≤ *i* ≤ *D*:

         1) Let LCD_i, be the column descriptor of the *i*-th column of *LT*.

         2) Let LCN_i be the column name included in LCD_i.

         3) Let LDT_i be the data type included in LCD_i.

         4) If the nullability characteristic included in LCD_i is known not nullable, then let LNC_i be NOT NULL; otherwise, let LNC_i be the zero-length string.

         5) Let CD_i be the <column definition>

         ```
         LCNi LDTi LNCi
         ```

      iv) If <like options> is specified, then:

         1) <identity option> shall not be specified more than once, <column default option> shall not be specified more than once, and <generation option> shall not be specified more than once.

         2) If <identity option> is not specified, then EXCLUDING IDENTITY is implicit.

         3) If <column default option> is not specified, then EXCLUDING DEFAULTS is implicit.

         4) If <generation option> is not specified, then EXCLUDING GENERATED is implicit.

         5) If INCLUDING IDENTITY is specified and *LT* includes an identity column, then let *ICD* be the column descriptor of that column included in the table descriptor of *LT*. Let *SGD* be the sequence generator descriptor included in *ICD*.

            A) Let *SV* be the start value included in *ICD*.

            B) Let *IV* be the increment included in *SGD*.

            C) Let *MAX* be the maximum value included in *SGD*.

            D) Let *MIN* be the minimum value included in *SGD*.

            E) Let *CYC* be the cycle option included in *SGD*.

            F) Let *k* be the ordinal position in which the column described by *ICD* appears in the table identified by *LT*.

G) Case:

I) If *ICD* indicates that values are always generated, then let *G* be GENERATED ALWAYS.

II) If *ICD* indicates that values are generated by default, then let *G* be GENERATED BY DEFAULT.

H) The value of CD_k is replaced by:

```
LCNk LDTk
  G AS IDENTITY ( START WITH SV, INCREMENT BY IV,
                  MAXVALUE MAX, MINVALUE MIN, CYC ) LNCk
```

6) If INCLUDING GENERATED is specified, then let GCD_j, 1 (one) $\leq j \leq D$, be the column descriptors included in the descriptor of *LT*, with *j* being the ordinal position of the column described by GCD_j. For each GCD_j that indicates that the column it describes is a generated column:

A) Let GE_j be the <generation expression> included in GCD_j, where the <table name> contained in any contained <column reference> is replaced by *TN*.

B) The value of CD_j is replaced by

```
LCNj LDTj GENERATED ALWAYS AS GEj LCNj
```

7) If INCLUDING DEFAULTS is specified, then let DCD_m, 1 (one) $\leq m \leq D$, be the column descriptors included in the descriptor of *LT*, with *m* being the ordinal position of the column described by DCD_m.

For each DCD_m, if DCD_m includes a <default option> DO_m, then the value of CD_m is replaced by

```
LCNm LDTm DEFAULT  DOm LCNm
```

v) *LC* is effectively replaced by:

```
CD1, ..., CDD
```

NOTE 245 — <column constraint>s, except for NOT NULL, are not included in CD_i; <column constraint definition>s are effectively transformed to <table constraint definition>s and are thereby also excluded.

6) If <as subquery clause> is specified, then:

a) Let *QT* be the table specified by the <subquery>.

b) If any two columns in *QT* have equivalent <column name>s, or if any column of *QT* has an implementation-dependent name, then <column name list> shall be specified.

c) Let *D* be the degree of *QT*.

d) <column name list> shall not contain two or more equivalent <column name>s.

e) The number of <column name>s in <column name list> shall be *D*.

f) For *i*, 1 (one) ≤ *i* ≤ *D*:

   i) Case:

      1) If <column name list> is specified, then let QCN_i be the *i*-th <column name> in that <column name list>.

      2) Otherwise, let QCN_i be the <column name> of the *i*-th column of *QT*.

   ii) Let QDT_i be the declared type of the *i*-th column of *QT*.

   iii) If the nullability characteristic of the *i*-th column of *QT* is known not nullable, then let QNC_i be NOT NULL; otherwise, let QNC_i be the zero-length string.

   iv) Let CD_i be the <column definition>

      QCN_i QDT_i QNC_i

g) <as subquery clause> is effectively replaced by a <table element list> *TEL* of the form:

   CD_1, ..., CD_D

7) If <typed table clause> *TTC* is specified, then:

a) The <user-defined type name> simply contained in <path-resolved user-defined type name> shall identify a structured type *ST*.

b) If <subtable clause> is specified, then <self-referencing column specification> shall not be specified. Otherwise, <self-referencing column specification> shall be specified exactly once.

c) If <self-referencing column specification> *SRCS* is specified, then let *RST* be the reference type REF(*ST*).

   i) <subtable clause> shall not be specified.

   ii) <table scope> shall not be specified.

   iii) If SYSTEM GENERATED is specified, then *RST* shall have a system-defined representation.

   iv) If USER GENERATED is specified, then *RST* shall have a user-defined representation.

   v) If DERIVED is specified, then *RST* shall have a derived representation.

   vi) If *RST* has a derived representation, then let *m* be the number of attributes included in the list of attributes of the derived representation of *RST* and let A_i, 1 (one) ≤ *i* ≤ *m*, be those attributes.

      1) *TD* shall contain a <table constraint definition> that specifies a <unique constraint definition> *UCD* whose <unique column list> contains the attribute names of A_1, A_2, ..., A_m in that order.

2) If *UCD* does not specify PRIMARY KEY, then for every attribute A_i, 1 (one) $\leq i \leq m$, *TD* shall contain a <column options> CO_i with a <column name> that is equivalent to the <attribute name> of A_i and with a <column constraint definition> that specifies NOT NULL.

vii) Let CD_0 be the <column definition>:

```
CN0 RST SCOPE(TN) UNIQUE NOT NULL
```

where CN_0 denotes the <self-referencing column name> simply contained in *SRCS*.

d) If <subtable clause> is specified, then:

i) The <table name> contained in the <subtable clause> identifies the *direct supertable* of *T*, which shall be a base table. *T* is called a *direct subtable* of the direct supertable of *T*.

ii) *ST* shall be a direct subtype of the structured type of the direct supertable of *T*.

iii) The SQL-schema identified by the explicit or implicit <schema name> of the <table name> of *T* shall include the descriptor of the direct supertable of *T*.

iv) The subtable family of *T* shall not include a member, other than *T* itself, whose associated structured type is *ST*.

v) *TD* shall not contain a <table constraint definition> that specifies PRIMARY KEY.

vi) Let the term *inherited column* of *T* refer to a column of *T* that corresponds to an inherited attribute of *ST*. For every such inherited attribute *IA*, there is a column *CA* of the direct supertable of *T* such that the <column name> of *CA* is equivalent to the <attribute name> of *IA*. *CA* is called the *direct supercolumn* of *IA* in the direct supertable of *T*.

vii) Let CD_0 be the <column definition>:

```
CN0 RST SCOPE(TN) UNIQUE NOT NULL
```

where CN_0 denotes the <self-referencing column name> simply contained in *SRCS*.

e) Let the term *originally-defined column* of *T* refer to a column of *T* that corresponds to an originally-defined attribute of *ST*.

f) Let *n* be the number of attributes of *ST*. Let AD_i, 1 (one) $\leq i \leq n$, be the attribute descriptors included in the data type descriptor of *ST* and let CD_i be the <column definition> CN_i DT_i DC_i, where:

i) CN_i is the attribute name included in AD_i.

ii) DT_i is some <data type> that, under the General Rules of Subclause 6.1, "<data type>", would result in the creation of the data type descriptor included in AD_i.

iii) If AD_i describes an inherited attribute *IA*, then

Case:

1) If the column descriptor of the direct supercolumn of *IA* includes a default value, then DC_i is some <default clause> whose <default option> denotes this default value.

2) Otherwise, DC_i is the zero-length string.

iv) If AD_i describes an originally-defined attribute *OA*, then

Case:

1) If AD_i includes a default value, then DC_i is some <default clause> whose <default option> denotes this default value.

2) Otherwise, DC_i is the zero-length string.

g) If <typed table element list> *TTEL* is specified and <column options> *CO* is specified, then:

i) The <column name> *CN* simply contained in *CO* shall be equivalent to the <column name> CN_j specified in some <column definition> CD_j and shall refer to an originally-defined column of *T*.

ii) *CN* shall not be equivalent to the <column name> simply contained in any other <column options> contained in *TTEL*.

iii) A <column option list> shall immediately contain either a <scope clause> or a <default clause>, or at least one <column constraint definition>.

iv) If *CO* specifies a <scope clause> *SC*, then DT_j shall be a <reference type> *RT*. If *RT* contains a <scope clause>, then that <scope clause> is replaced by *SC*; otherwise, *RT* is replaced by *RT SC*.

NOTE 246 — Changes to the scope of a column of a typed table do not affect the scope defined for the underlying attribute. Such an attribute scope serves as a kind of default for the column's scope, at the time the typed table is defined, and is not restored if a column's scope is dropped.

v) If *CO* specifies a <default clause> *DC*, then DC_j is replaced by *DC* in CD_j.

vi) If *CO* specifies a non-empty list *CCDL* of <column constraint definition>s, then CD_j is replaced by CD_j *CCDL*.

vii) *CO* is deleted from *TTEL*.

h) *T* is a referenceable table.

i) If *TTEL* is empty, then let *TEL* be a <table element list> of the form

CD_0, . . . , CD_n

Otherwise, then let *TEL* be a <table element list> of the form

CD_0, . . . , CD_n *TTEL*

8) If ON COMMIT is specified, then TEMPORARY shall be specified.

9) If TEMPORARY is specified and ON COMMIT is not specified, then ON COMMIT DELETE ROWS is implicit.

10) Every referenceable table referenced by a <scope clause> contained in a <column definition> or <column options> contained in *TD* shall be

Case:

a) If *TD* specifies no <table scope>, then a persistent base table.

b) If *TD* specifies GLOBAL TEMPORARY, then a global temporary table.

c) If *TD* specifies LOCAL TEMPORARY, then a created local temporary table.

11) At most one <table element> shall be a <column definition> that contains an <identity column specification>.

12) The scope of the <table name> is the <table definition>, excluding the <as subquery clause>.

Access Rules

1) If a <table definition> is contained in an <SQL-client module definition>, then the enabled authorization identifiers shall include *A*.

2) If a <like clause> is contained in a <table definition>, then the applicable privileges for *A* shall include SELECT privilege on the table identified in the <like clause>.

3) *A* shall have in its applicable privileges the UNDER privilege on the <supertable name> specified in <subtable clause>.

4) If "OF <path-resolved user-defined type name>" is specified, then the applicable privileges for *A* shall include USAGE on *ST*.

General Rules

1) A <table definition> defines either a persistent base table, a global temporary table or a created local temporary table. If GLOBAL is specified, then a global temporary table is defined. If LOCAL is specified, then a created local temporary table is defined. Otherwise, a persistent base table is defined.

2) The degree of *T* is initially set to 0 (zero); the General Rules of Subclause 11.4, "<column definition>", specify the degree of *T* during the definition of the columns of *T*.

3) If <path-resolved user-defined type name> is specified, then:

a) Let *R* be the structured type identified by the <user-defined type name> simply contained in <path-resolved user-defined type name>.

b) *R* is the structured type associated with *T*.

4) A table descriptor *TDS* is created that describes *T*. *TDS* includes:

a) The table name *TN*.

b) The column descriptors of every column of *T*, according to the Syntax Rules and General Rules of Subclause 11.4, "<column definition>", applied to the <column definition>s contained in *TEL*, in the order in which they were specified.

c) If <typed table clause> is specified, then:

i) An indication that the table is a referenceable table.

ii) An indication that the column at ordinal position 1 (one) is the self-referencing column of *T*. The column descriptor included in *TDS* that describes that column is marked as identifying a self-referencing column.

iii) If *RST* has a system-defined representation, then an indication that the self-referencing column is a system-generated self-referencing column.

iv) If *RST* has a derived representation, then an indication that the self-referencing column is a derived self-referencing column.

v) If *RST* has a user-defined representation, then an indication that the self-referencing column is a user-generated self-referencing column.

d) The table constraint descriptors specified by each <table constraint definition> contained in *TEL*.

e) If a <path-resolved user-defined type name> is specified, then the user-defined type name of *R*.

f) If <subtable clause> is specified, then the table name of the direct supertable of *T* contained in the <subtable clause>.

g) A non-empty set of functional dependencies, according to the rules given in Subclause 4.18, "Functional dependencies".

h) A non-empty set of candidate keys.

i) A preferred candidate key, which may or may not be additionally designated the primary key, according to the Rules in Subclause 4.18, "Functional dependencies".

j) An indication of whether the table is a persistent base table, a global temporary table, a created local temporary table, or a declared local temporary table.

k) If TEMPORARY is specified, then

Case:

i) If ON COMMIT PRESERVE ROWS is specified, then the table descriptor includes an indication that ON COMMIT PRESERVE ROWS is specified.

ii) Otherwise, the table descriptor includes an indication that ON COMMIT DELETE ROWS is specified or implied.

l) Case:

i) If <typed table clause> is not specified, then an indication that *T* is insertable-into.

ii) Otherwise,

Case:

1) If the data type descriptor of *R* indicates that *R* is instantiable, then an indication that *T* is insertable-into.

2) Otherwise, an indication that *T* is not insertable-into.

5) In the descriptor of each direct supertable of *T*, *TN* is added to the end of the list of direct subtables.

6) If <subtable clause> is specified, then a set of privilege descriptors is created that defines the privileges SELECT, UPDATE, and REFERENCES for every inherited column of this table to the <authorization identifier> that owns the schema identified by the implicit or explicit <schema name> of the <table name> of the direct supertable from which that column was inherited. These privileges are grantable. The grantor for each of these privilege descriptors is set to the special grantor value "_SYSTEM".

7) A set of privilege descriptors is created that define the privileges INSERT, SELECT, UPDATE, DELETE, TRIGGER, and REFERENCES on this table and SELECT, INSERT, UPDATE, and REFERENCES for every <column definition> in the table definition. If OF <path-resolved user-defined type name> is specified, then a table/method privilege descriptor is created on this table for every method of the structured type identified by the <path-resolved user-defined type name> and the table SELECT privilege has the WITH HIERARCHY OPTION. These privileges are grantable.

   The grantor for each of these privilege descriptors is set to the special grantor value "_SYSTEM". The grantee is <authorization identifier> *A*.

8) If <subtable clause> is specified, then let *ST* be the set of supertables of *T*. Let *PDS* be the set of privilege descriptors that defined SELECT WITH HIERARCHY OPTION privilege on a table in *ST*. For every privilege descriptor in *PDS*, with grantee *G*, grantor *A*,

   Case:

   a) If the privilege is grantable, then let *WGO* be "WITH GRANT OPTION".

   b) Otherwise, let *WGO* be a zero-length string.

   The following <grant statement> is effectively executed without further Access Rule checking:

   ```
   GRANT SELECT ON T TO G WGO FROM A
   ```

9) The row type *RT* of the table *T* defined by the <table definition> is the set of pairs (<field name>, <data type>) where <field name> is the name of a column *C* of *T* and <data type> is the declared type of *C*. This set of pairs contains one pair for each column of *T*, in the order of their ordinal position in *T*.

10) If <as subquery clause> is specified and WITH DATA is specified, then let *QE* be the <query expression> immediately contained in the <subquery>. The following <insert statement> is effectively executed without further Access Rule checking:

   ```
   INSERT INTO TN QE
   ```

Conformance Rules

1) Without Feature T171, "LIKE clause in table definition", conforming SQL language shall not contain a <like clause>.

2) Without Feature F531, "Temporary tables", conforming SQL language shall not contain a <table scope> and shall not reference any global or local temporary table.

3) Without Feature S051, "Create table of type", conforming SQL language shall not contain "OF <path-resolved user-defined type name>".

4) Without Feature S043, “Enhanced reference types”, conforming SQL language shall not contain a <column option list> that contains a <scope clause>.

5) Without Feature S043, “Enhanced reference types”, conforming SQL language shall not contain <reference generation> that does not contain SYSTEM GENERATED.

6) Without Feature S081, “Subtables”, conforming SQL language shall not contain a <subtable clause>.

7) Without Feature T172, “AS subquery clause in table definition”, conforming SQL language shall not contain an <as subquery clause>.

8) Without Feature T173, “Extended LIKE clause in table definition”, a <like clause> shall not contain <like options>.

11.4 <column definition>

Function

Define a column of a base table.

Format

```
<column definition> ::=
    <column name> [ <data type or domain name> ]
    [ <default clause> | <identity column specification> | <generation clause> ]
    [ <column constraint definition>... ]
    [ <collate clause> ]

<data type or domain name> ::=
    <data type>
  | <domain name>

<column constraint definition> ::=
    [ <constraint name definition> ] <column constraint> [ <constraint characteristics> ]

<column constraint> ::=
    NOT NULL
  | <unique specification>
  | <references specification>
  | <check constraint definition>

<identity column specification> ::=
    GENERATED { ALWAYS | BY DEFAULT } AS IDENTITY
    [ <left paren> <common sequence generator options> <right paren> ]

<generation clause> ::= <generation rule> AS <generation expression>

<generation rule> ::= GENERATED ALWAYS

<generation expression> ::= <left paren> <value expression> <right paren>
```

Syntax Rules

1) Case:

    a) If the <column definition> is contained in a <table definition>, then let *T* be the table defined by that <table definition>.

    b) If the <column definition> is contained in a <temporary table declaration>, then let *T* be the table declared by that <temporary table declaration>.

    c) If the <column definition> is contained in an <alter table statement>, then let *T* be the table identified in the containing <alter table statement>.

    The <column name> in the <column definition> shall not be equivalent to the <column name> of any other column of *T*.

2) Let *A* be the <authorization identifier> that owns *T*.

3) Let *C* be the <column name> of the <column definition>.

4) <data type or domain name> shall unambiguously reference either a <data type> or a <domain name>.

5) If <domain name> is specified, then let *D* be the domain identified by the <domain name>.

6) If <generation clause> *GC* is specified, then:

   a) Let *GE* be the <generation expression> contained in *GC*.

   b) *C* is a generated column.

   c) Every <column reference> contained in *GE* shall reference a base column of *T*.

   d) *GE* shall be deterministic.

   e) *GE* shall not contain a <routine invocation> whose subject routine possibly reads SQL-data.

   f) *GE* shall not contain a <subquery>.

7) If <generation clause> is omitted, then either <data type> or <domain name> shall be specified.

8) Case:

   a) If <column definition> immediately contains <domain name>, then it shall not also immediately contain <collate clause>.

   b) Otherwise, <collate clause> shall not be both specified in <data type> and immediately contained in <column definition>. If <collate clause> is immediately contained in <column definition>, then it is equivalent to specifying an equivalent <collate clause> in <data type>.

9) The declared type of the column is

   Case:

   a) If <data type> is specified, then that data type. If <generation clause> is also specified, then the declared type of <generation expression> shall be assignable to the declared type of the column.

   b) If <domain name> is specified, then the declared type of *D*. If <generation clause> is also specified, then the declared type of <generation expression> shall be assignable to the declared type of the column.

   c) If <generation clause> is specified, then the declared type of *GE*.

10) If a <data type> is specified, then:

   a) Let *DT* be the <data type>.

   b) If *DT* specifies CHARACTER, CHARACTER VARYING, or CHARACTER LARGE OBJECT and does not specify a <character set specification>, then the <character set specification> specified or implicit in the <schema character set specification> of the <schema definition> that created the schema identified by the <schema name> immediately contained in the <table name> of the containing <table definition> or <alter table statement> is implicit.

11) If <identity column specification> *ICS* is specified, then:

   a) Case:

i) If the declared type of the column being defined is a distinct type *DIST*, then the source type of *DIST* shall be exact numeric with scale 0 (zero). Let *ICT* be the source type of *DIST*.

ii) Otherwise, the declared type of the column being defined shall be exact numeric with scale 0 (zero). Let *ICT* be the declared type of the column being defined.

b) Let *SGO* be the <common sequence generator options>.

c) The Syntax Rules of Subclause 9.22, "Creation of a sequence generator", are applied with *SGO* as *OPTIONS* and *ICT* as *DATA TYPE*.

d) The <column constraint definition> NOT NULL NOT DEFERRABLE is implicit.

12) If a <column constraint definition> is specified, then let *CND* be the <constraint name definition> if one is specified and let *CND* be a zero-length string otherwise; let *CA* be the <constraint characteristics> if specified and let *CA* be a zero-length string otherwise. The <column constraint definition> is equivalent to a <table constraint definition> as follows:

Case:

a) If a <column constraint definition> is specified that contains the <column constraint> NOT NULL, then it is equivalent to the following <table constraint definition>:

```
CND CHECK ( C IS NOT NULL ) CA
```

b) If a <column constraint definition> is specified that contains a <unique specification> *US*, then it is equivalent to the following <table constraint definition>:

```
CND US (C) CA
```

NOTE 247 — The <unique specification> is defined in Subclause 11.7, "<unique constraint definition>".

c) If a <column constraint definition> is specified that contains a <references specification> *RS*, then it is equivalent to the following <table constraint definition>:

```
CND FOREIGN KEY (C) RS CA
```

NOTE 248 — The <references specification> is defined in Subclause 11.8, "<referential constraint definition>".

d) If a <column constraint definition> is specified that contains a <check constraint definition> *CCD*, then it is equivalent to the following <table constraint definition>:

```
CND CCD CA
```

Each column reference directly contained in the <search condition> shall reference column *C*.

13) The schema identified by the explicit or implicit qualifier of the <domain name> shall include the descriptor of *D*.

Access Rules

1) If <domain name> is specified, then the applicable privileges for *A* shall include USAGE on *D*.

General Rules

1) A <column definition> defines a column in a table.

2) If the <column definition> specifies <data type>, then a data type descriptor is created that describes the declared type of the column being defined.

3) The degree of the table *T* being defined in the containing <table definition> or <temporary table declaration>, or being altered by the containing <alter table statement> is increased by 1 (one).

4) A column descriptor is created that describes the column being defined. The column descriptor includes:

   a) *C*, the name of the column.

   b) Case:

      i) If the <column definition> specifies a <data type> or a <generation clause>, then the data type descriptor of the declared type of the column.

      ii) Otherwise, the domain of the column.

   c) The ordinal position of the column, which is equal to the degree of *T*.

   d) The nullability characteristic of the column, determined according to the rules in Subclause 4.13, "Columns, fields, and attributes".

      NOTE 249 — Both <column constraint definition>s and <table constraint definition>s shall be analyzed to determine the nullability characteristics of all columns.

   e) If <default clause> is specified, then the <default option>.

   f) If <identity column specification> is specified, then:

      i) An indication that the column is an identity column.

      ii) If ALWAYS is specified, then an indication that values are always generated.

      iii) If BY DEFAULT is specified, then an indication that values are generated by default.

      iv) The descriptor of the sequence generator descriptor *SG* resulting from application of the General Rules of Subclause 9.22, "Creation of a sequence generator", with *SGO* as *OPTIONS* and *ICT* as *DATA TYPE*.

      v) The next available value of *SG* as the start value.

   g) If <generation clause> is specified, then *GE*.

Conformance Rules

1) Without Feature F692, "Extended collation support", conforming SQL language shall not contain a <column definition> that immediately contains a <collate clause>.

2) Without Feature T174, "Identity columns", conforming SQL language shall not contain an <identity column specification>.

3) Without Feature T175, “Generated columns”, conforming SQL language shall not contain a <generation clause>.

11.5 <default clause>

Function

Specify the default for a column, domain, or attribute.

Format

```
<default clause> ::= DEFAULT <default option>

<default option> ::=
    <literal>
  | <datetime value function>
  | USER
  | CURRENT_USER
  | CURRENT_ROLE
  | SESSION_USER
  | SYSTEM_USER
  | CURRENT_PATH
  | <implicitly typed value specification>
```

Syntax Rules

1) The subject data type of a <default clause> is the data type specified in the descriptor identified by the containing <column definition>, <domain definition>, <attribute definition>, <alter column definition>, or <alter domain statement>.

2) If USER is specified, then CURRENT_USER is implicit.

3) Case:

   a) If the subject data type of the <default clause> is a user-defined type, a reference type, or a row type, then <default option> shall specify <null specification>.

   b) If the subject data type of the <default clause> is a collection type, then <default option> shall specify <implicitly typed value specification>. If the <default option> specifies an <empty specification> that specifies ARRAY, then the subject data type shall be an array type. If the <default option> specifies an <empty specification> that specifies MULTISET, then the subject data type shall be a multiset type.

4) Case:

   a) If a <literal> is specified, then

      Case:

      i) If the subject data type is character string, then the <literal> shall be a <character string literal>. If the length of the subject data type is fixed, then the length in characters of the <character string literal> shall not be greater than the length of the subject data type. If the length of the subject data type is variable, then the length in characters of the <character string literal> shall not be greater than the maximum length of the subject data type. The <literal> shall have the same character repertoire as the subject data type.

ii) If the subject data type is binary string, then the <literal> shall be a <binary string literal> that has an even number of <hexit>s. The length in octets of the <binary string literal> shall not be greater than the maximum length of the subject data type.

iii) If the subject data type is exact numeric, then the <literal> shall be a <signed numeric literal> that simply contains an <exact numeric literal>. There shall be a representation of the value of the <literal> in the subject data type that does not lose any significant digits.

iv) If the subject data type is approximate numeric, then the <literal> shall be a <signed numeric literal>.

v) If the subject data type is datetime, then the <literal> shall be a <datetime literal> with the same primary datetime fields and the same time zone datetime fields as the subject data type. If SECOND is one of these fields, then the fractional seconds precision of the <datetime literal> shall be less than or equal to the fractional seconds precision of the subject data type.

vi) If the subject data type is interval, then the <literal> shall be an <interval literal> and shall contain the same <interval qualifier> as the subject data type.

vii) If the subject data type is boolean, then the <literal> shall be a <boolean literal>.

b) If CURRENT_USER, CURRENT_ROLE, SESSION_USER, or SYSTEM_USER is specified, then the subject data type shall be character string with character set SQL_IDENTIFIER. If the length of the subject data type is fixed, then its length shall not be less than 128 characters. If the length of the subject data type is variable, then its maximum length shall not be less than 128 characters.

c) If CURRENT_PATH is specified, then the subject data type shall be character string with character set SQL_IDENTIFIER. If the length of the subject data type is fixed, then its length shall not be less than 1031 characters. If the length of the subject data type is variable, then its maximum length shall not be less than 1031 characters.

d) If <datetime value function> is specified, then the subject data type shall be datetime with the same declared datetime data type of the <datetime value function>.

e) If <empty specification> is specified, then the subject data type shall be a collection type.

Access Rules

None.

General Rules

1) The default value inserted in the column descriptor, if the <default clause> is to apply to a column, or in the domain descriptor, if the <default clause> is to apply to a domain, or in the attribute descriptor, if the <default clause> is to apply to an attribute, is the <default option>.

2) The value specified by a <default option> is

Case:

a) If the <default option> contains a <literal>, then

Case:

   i) If the subject data type is numeric, then the numeric value of the <literal>.

   ii) If the subject data type is character string with variable length, then the value of the <literal>.

   iii) If the subject data type is character string with fixed length, then the value of the <literal>, extended as necessary on the right with <space>s to the length in characters of the subject data type.

   iv) If the subject data type is binary string, then the value of the <literal>.

   v) If the subject data type is datetime or interval, then the value of the <literal>.

   vi) If the subject data type is boolean, then the value of the <literal>.

b) If the <default option> specifies CURRENT_USER, CURRENT_ROLE, SESSION_USER, SYSTEM_USER, or CURRENT_PATH, then

   Case:

   i) If the subject data type is character string with variable length, then the value obtained by an evaluation of CURRENT_USER, SESSION_USER, SYSTEM_USER, or CURRENT_PATH at the time that the default value is required.

   ii) If the subject data type is character string with fixed length, then the value obtained by an evaluation of CURRENT_USER, SESSION_USER, CURRENT_PATH, or SYSTEM_USER at the time that the default value is required, extended as necessary on the right with <space>s to the length in characters of the subject data type.

c) If the <default option> contains a <datetime value function>, then the value of an evaluation of the <datetime value function> at the time that the default value is required.

d) If the <default option> specifies <empty specification>, then an empty collection.

3) When a site *S* is set to its default value,

   Case:

   a) If the descriptor of *S* indicates that it represents a column of which some underlying column is an identity column or a generated column, then *S* is marked as *unassigned.*

   NOTE 250 — The notion of a site being unassigned is only for definitional purposes in this International Standard. It is not a state that can persist so as to be visible in SQL-data. The treatment of unassigned sites is given in Subclause 14.19, "Effect of inserting tables into base tables", and Subclause 14.22, "Effect of replacing rows in base tables".

   b) If the data descriptor for the site includes a <default option>, then *S* is set to the value specified by that <default option>.

   c) If the data descriptor for the site includes a <domain name> that identifies a domain descriptor that includes a <default option>, then *S* is set to the value specified by that <default option>.

   d) If the default value is for a column *C* of a candidate row for insertion into or update of a derived table *DT* and *C* has a single counterpart column *CC* in a leaf generally underlying table of *DT*, then *S* is set to the default value of *CC*, which is obtained by applying the General Rules of this Subclause.

   e) Otherwise, *S* is set to the null value.

NOTE 251 — If <default option> specifies CURRENT_USER, SESSION_USER, SYSTEM_USER, CURRENT_ROLE or CURRENT_PATH, then the "value in the column descriptor" will effectively be the text of the <default option>, whose evaluation occurs at the time that the default value is required.

4) If the <default clause> is contained in an <SQL schema statement> and character representation of the <default option> cannot be represented in the Information Schema without truncation, then a completion condition is raised: *warning — default value too long for information schema.*

NOTE 252 — The Information Schema is defined in ISO/IEC 9075-11.

Conformance Rules

1) Without Feature S071, "SQL paths in function and type name resolution", conforming SQL language shall not contain a <default option> that contains CURRENT_PATH.

2) Without Feature F321, "User authorization", conforming SQL language shall not contain a <default option> that contains CURRENT_USER, SESSION_USER, or SYSTEM_USER.

NOTE 253 — Although CURRENT_USER and USER are semantically the same, without Feature F321, "User authorization", CURRENT_USER shall be specified as USER.

3) Without Feature T332, "Extended roles", conforming SQL language shall not contain a <default option> that contains CURRENT_ROLE.

11.6 <table constraint definition>

Function

Specify an integrity constraint.

Format

```
<table constraint definition> ::=
    [ <constraint name definition> ] <table constraint>
    [ <constraint characteristics> ]

<table constraint> ::=
    <unique constraint definition>
  | <referential constraint definition>
  | <check constraint definition>
```

Syntax Rules

1) If <constraint characteristics> is not specified, then INITIALLY IMMEDIATE NOT DEFERRABLE is implicit.

2) If <constraint name definition> is specified and its <constraint name> contains a <schema name>, then that <schema name> shall be equivalent to the explicit or implicit <schema name> of the <table name> of the table identified by the containing <table definition> or <alter table statement>.

3) If <constraint name definition> is not specified, then a <constraint name definition> that contains an implementation-dependent <constraint name> is implicit. The assigned <constraint name> shall obey the Syntax Rules of an explicit <constraint name>.

Access Rules

None.

General Rules

1) A <table constraint definition> defines a table constraint.

2) A table constraint descriptor is created that describes the table constraint being defined. The table constraint descriptor includes the <constraint name> contained in the explicit or implicit <constraint name definition>.

   The table constraint descriptor includes an indication of whether the constraint is deferrable or not deferrable and whether the initial constraint mode of the constraint is *deferred* or *immediate*.

   Case:

   a) If <unique constraint definition> is specified, then the table constraint descriptor is a unique constraint descriptor that includes an indication of whether it was defined with PRIMARY KEY or UNIQUE, and the names of the unique columns specified in the <unique column list>.

b) If <referential constraint definition> is specified, then the table constraint descriptor is a referential constraint descriptor that includes a list of the names of the referencing columns specified in the <referencing columns>, the name of the referenced table specified in the <referenced table and columns> and a list of the names of the referenced columns specified in the <referenced table and columns>, the value of the <match type>, if specified, and the <referential triggered action>s, if specified. The ordering of the lists of referencing column names and referenced column names is implementation-defined, but shall be such that corresponding column names occupy corresponding positions in each list.

c) If <check constraint definition> is specified, then the table constraint descriptor is a table check constraint descriptor that includes the <search condition>.

3) If the <table constraint> is a <check constraint definition>, then let *SC* be the <search condition> immediately contained in the <check constraint definition> and let *T* be the table name included in the corresponding table constraint descriptor; the table constraint is not satisfied if and only if

```
EXISTS ( SELECT * FROM T WHERE NOT ( SC ) )
```

is *True*.

Conformance Rules

None.

11.7 <unique constraint definition>

Function

Specify a uniqueness constraint for a table.

Format

```
<unique constraint definition> ::=
    <unique specification> <left paren> <unique column list> <right paren>
  | UNIQUE ( VALUE )

<unique specification> ::=
    UNIQUE
  | PRIMARY KEY

<unique column list> ::= <column name list>
```

Syntax Rules

1) Each column identified by a <column name> in the <unique column list> is an operand of a grouping operation. The Syntax Rules of Subclause 9.10, "Grouping operations", apply.

2) Let *T* be the table identified by the containing <table definition> or <alter table statement>. Let *TN* be the <table name> of *T*.

3) If <unique column list> *UCL* is specified, then

   a) Each <column name> in the <unique column list> shall identify a column of *T*, and the same column shall not be identified more than once.

   b) The set of columns in the <unique column list> shall be distinct from the unique columns of any other unique constraint descriptor that is included in the base table descriptor of *T*.

   c) Case:

      i) If the <unique specification> specifies PRIMARY KEY, then let *SC* be the <search condition>:
      ```
      UNIQUE ( SELECT UCL FROM TN )
          AND
      ( UCL ) IS NOT NULL
      ```

      ii) Otherwise, let *SC* be the <search condition>:
      ```
      UNIQUE ( SELECT UCL FROM TN )
      ```

4) If UNIQUE (VALUE) is specified, then let *SC* be the <search condition>:
   ```
   UNIQUE ( SELECT TN.* FROM TN )
   ```

5) If the <unique specification> specifies PRIMARY KEY, then for each <column name> in the explicit or implicit <unique column list> for which NOT NULL is not specified, NOT NULL is implicit in the <column definition>.

6) A <table definition> shall specify at most one implicit or explicit <unique constraint definition> that specifies PRIMARY KEY.

7) If a <unique constraint definition> that specifies PRIMARY KEY is contained in an <add table constraint definition>, then the table identified by the <table name> immediately contained in the containing <alter table statement> shall not have a unique constraint that was defined by a <unique constraint definition> that specified PRIMARY KEY.

Access Rules

None.

General Rules

1) A <unique constraint definition> defines a unique constraint.

NOTE 254 — Subclause 10.8, "<constraint name definition> and <constraint characteristics>", specifies when a constraint is effectively checked.

2) The unique constraint is not satisfied if and only if

```
EXISTS ( SELECT * FROM TN WHERE NOT ( SC ) )
```

is True.

Conformance Rules

1) Without Feature S291, "Unique constraint on entire row", conforming SQL language shall not contain UNIQUE(VALUE).

2) Without Feature T591, "UNIQUE constraints of possibly null columns", in conforming SQL language, if UNIQUE is specified, then the <column definition> for each column whose <column name> is contained in the <unique column list> shall contain NOT NULL.

NOTE 255 — The Conformance Rules of Subclause 9.10, "Grouping operations", also apply.

11.8 <referential constraint definition>

Function

Specify a referential constraint.

Format

```
<referential constraint definition> ::=
    FOREIGN KEY <left paren> <referencing columns> <right paren>
    <references specification>

<references specification> ::=
    REFERENCES <referenced table and columns>
    [ MATCH <match type> ] [ <referential triggered action> ]

<match type> ::=
    FULL
  | PARTIAL
  | SIMPLE

<referencing columns> ::= <reference column list>

<referenced table and columns> ::=
    <table name> [ <left paren> <reference column list> <right paren> ]

<reference column list> ::= <column name list>

<referential triggered action> ::=
    <update rule> [ <delete rule> ]
  | <delete rule> [ <update rule> ]

<update rule> ::= ON UPDATE <referential action>

<delete rule> ::= ON DELETE <referential action>

<referential action> ::=
    CASCADE
  | SET NULL
  | SET DEFAULT
  | RESTRICT
  | NO ACTION
```

Syntax Rules

1) If <match type> is not specified, then SIMPLE is implicit.

2) Let *referencing table* be the table identified by the containing <table definition> or <alter table statement>. Let *referenced table* be the table identified by the <table name> in the <referenced table and columns>. Let *referencing columns* be the column or columns identified by the <reference column list> in the <referencing columns> and let *referencing column* be one such column.

3) Case:

a) If the <referenced table and columns> specifies a <reference column list>, then there shall be a one-to-one correspondence between the set of <column name>s contained in that <reference column list> and the set of <column name>s contained in the <unique column list> of a unique constraint of the referenced table such that corresponding <column name>s are equivalent. Let *referenced columns* be the column or columns identified by that <reference column list> and let *referenced column* be one such column. Each referenced column shall identify a column of the referenced table and the same column shall not be identified more than once.

b) If the <referenced table and columns> does not specify a <reference column list>, then the table descriptor of the referenced table shall include a unique constraint that specifies PRIMARY KEY. Let *referenced columns* be the column or columns identified by the unique columns in that unique constraint and let *referenced column* be one such column. The <referenced table and columns> shall be considered to implicitly specify a <reference column list> that is identical to that <unique column list>.

4) The table constraint descriptor describing the <unique constraint definition> whose <unique column list> identifies the referenced columns shall indicate that the unique constraint is not deferrable.

5) The referenced table shall be a base table.

Case:

a) If the referencing table is a persistent base table, then the referenced table shall be a persistent base table.

b) If the referencing table is a global temporary table, then the referenced table shall be a global temporary table.

c) If the referencing table is a created local temporary table, then the referenced table shall be either a global temporary table or a created local temporary table.

d) If the referencing table is a declared local temporary table, then the referenced table shall be either a global temporary table, a created local temporary table or a declared local temporary table.

6) If the referenced table is a temporary table with ON COMMIT DELETE ROWS specified, then the referencing table shall specify ON COMMIT DELETE ROWS.

7) Each referencing column shall identify a column of the referencing table, and the same column shall not be identified more than once.

8) Each referencing column is an operand of a grouping operation. The Syntax Rules of Subclause 9.10, "Grouping operations", apply.

9) The <referencing columns> shall contain the same number of <column name>s as the <referenced table and columns>. The *i*-th column identified in the <referencing columns> corresponds to the *i*-th column identified in the <referenced table and columns>. The declared type of each referencing column shall be comparable to the declared type of the corresponding referenced column. There shall not be corresponding constituents of the declared type of a referencing column and the declared type of the corresponding referenced column such that one constituent is datetime with time zone and the other is datetime without time zone.

10) If a <referential constraint definition> does not specify any <update rule>, then an <update rule> with a <referential action> of NO ACTION is implicit.

11) If a <referential constraint definition> does not specify any <delete rule>, then a <delete rule> with a <referential action> of NO ACTION is implicit.

12) If any referencing column is a generated column, then:

   a) <referential action> shall not specify SET NULL or SET DEFAULT.

   b) <update rule> shall not specify ON UPDATE CASCADE.

13) Let *T* be the referenced table. The schema identified by the explicit or implicit qualifier of the <table name> shall include the descriptor of *T*.

Access Rules

1) The applicable privileges for the owner of *T* shall include REFERENCES for each referenced column.

General Rules

1) A <referential constraint definition> defines a referential constraint.

   NOTE 256 — Subclause 10.8, "<constraint name definition> and <constraint characteristics>", specifies when a constraint is effectively checked.

2) Let R_f be the referencing columns and let R_t be the referenced columns in the referenced table *T*. The referencing table and the referenced table satisfy the referential constraint if and only if:

   Case:

   a) SIMPLE is specified or implicit and for each row of the referencing table, the <match predicate>

   ```
   Rf MATCH SIMPLE ( SELECT Rt FROM T )
   ```

   is <u>*True*</u>.

   b) PARTIAL is specified and for each row of the referencing table, the <match predicate>

   ```
   Rf MATCH PARTIAL ( SELECT Rt FROM T )
   ```

   is <u>*True*</u>.

   c) FULL is specified and for each row of the referencing table, the <match predicate>

   ```
   Rf MATCH FULL ( SELECT Rt FROM T )
   ```

   is <u>*True*</u>.

3) Case:

   a) If SIMPLE is specified or implicit, or if FULL is specified, then for a given row in the referenced table, every row that is a subrow or a superrow of a row *R* in the referencing table such that the referencing column values equal the corresponding referenced column values in *R* for the referential constraint is a *matching row*.

   b) If PARTIAL is specified, then:

i) For a given row in the referenced table, every row that is a subrow or a superrow of a row *R* in the referencing table such that *R* has at least one non-null referencing column value and the non-null referencing column values of *R* equal the corresponding referenced column values for the referential constraint is a *matching row*.

ii) For a given row in the referenced table, every matching row for that given row that is a matching row only to the given row in the referenced table for the referential constraint is a *unique matching row*. For a given row in the referenced table, a matching row for that given row that is not a unique matching row for that given row for the referential constraint is a *non-unique matching row*.

4) For each row of the referenced table, its matching rows, unique matching rows, and non-unique matching rows are determined immediately prior to the execution of any <SQL procedure statement>. No new matching rows are added during the execution of that <SQL procedure statement>.

The association between a referenced row and a non-unique matching row is dropped during the execution of that SQL-statement if the referenced row is either marked for deletion or updated to a distinct value on any referenced column that corresponds to a non-null referencing column. This occurs immediately after such a mark for deletion or update of the referenced row. Unique matching rows and non-unique matching rows for a referenced row are evaluated immediately after dropping the association between that referenced row and a non-unique matching row.

5) Let *CTEC* be the current trigger execution context. Let *SSC* be the set of state changes in *CTEC*. Let SC_i be a state change in *SSC*.

6) Let *F* be a subtable or supertable of the referencing table.

a) Let *FL* be the set of all columns of *F*. Let *SRC* be the set of referencing columns in *F*. Let *SS* be the set whose elements are the empty set and each subset of *FL* that contains at least one column in *SRC*. Let *NSS* be the number of sets in *SS*.

b) Let *PMC* be the set of referencing columns in *F* that correspond with the referenced columns. Let *PSS* be the set whose elements are the empty set and each subset of *FL* that contains at least one column in *PMC*. Let *PNSS* be the number of sets in *PSS*.

c) Let *UMC* be the set of referencing columns that correspond with updated referenced columns. Let *USS* be the set whose elements are the empty set and each subset of *FL* that contains at least one column in *UMC*. Let *UNSS* be the number of sets in *USS*.

7) For every row of the referenced table that is marked for deletion and has not previously been marked for deletion,

Case:

a) If SIMPLE is specified or implicit, or if FULL is specified, then

Case:

i) If the <delete rule> specifies CASCADE, then for every *F*:

1) Every matching row in *F* is marked for deletion.

2) If no SC_i has subject table *F*, trigger event DELETE, and an empty column list, then a state change SC_j is added to *SSC* as follows:

A) The trigger event of SC_j is DELETE.

B) The subject table of SC_j is F.

C) The column list of SC_j is empty.

D) The set of transitions of SC_j is empty.

NOTE 257 — The set of transitions will have been replaced by a nonempty set by the time any triggers activated by this state change are executed.

E) The set of statement-level triggers for which SC_j is considered as executed is empty.

F) The set of row-level triggers consists of each row-level trigger that is activated by SC_j, paired with the empty set (of rows considered as executed).

ii) If the <delete rule> specifies SET NULL, then:

1) For every *F*, for each matching row *MR* in *F*, a transition is formed by pairing *MR* with the value formed by copying *MR* and setting each referencing column in the copy to the null value.

2) For every *F*, for every generated column *GC* in *F* that depends on some referencing column, for every site *GCS* corresponding to *GC* in the new row *NR* of a transition for *F*, let *GCR* be the result of evaluating, for *NR*, the generation expression included in the column descriptor of *GC*. The General Rules of Subclause 9.2, "Store assignment", are applied with *GCS* as *TARGET* and *GCR* as *VALUE*.

3) The General Rules of Subclause 14.25, "Execution of BEFORE triggers", are applied with the following set of state changes *BTSS*:

A) For every *F*, for *k* ranging from 1 (one) to *NSS*, let SS_k be the *k*-th set of *SS*.

B) *BTSS* contains a state change SC_k as follows:

I) The trigger event of SC_k is UPDATE.

II) The subject table of SC_k is *F*.

III) The column list of SC_k is SS_k.

IV) The set of transitions of SC_k is the set of transitions for *F*.

V) The set of statement-level triggers for which SC_k is considered as executed is empty.

VI) The set of row-level triggers consists of each row-level trigger that is activated by SC_k , paired with the empty set (of rows considered as executed).

4) For every matching row *MR* in every *F*, *F* is identified for replacement processing and *MR* is identified for replacement in *F*. The General Rules of Subclause 14.22, "Effect of replacing rows in base tables", are applied.

5) For every *F*, for *k* ranging from 1 (one) to *NSS*, let SS_k be the *k*-th set of *SS*.

Case:

A) If no SC_i has subject table F, trigger event UPDATE, and column list that is SS_k, then a state change SC_j is added to SSC as follows:

I) The trigger event of SC_j is UPDATE.

II) The subject table of SC_j is F.

III) The column list of SC_j is SS_k.

IV) The set of transitions of SC_j is the set of transitions for F.

V) The set of statement-level triggers for which SC_j is considered as executed is empty.

VI) The set of row-level triggers consists of each row-level trigger that is activated by SC_j, paired with the empty set (of rows considered as executed).

B) Otherwise, let SC_j be the state change in SSC that has subject table F, trigger event UPDATE, and a column list that is SS_k. The set of transitions of SC_j is the set of transitions for F.

iii) If the <delete rule> specifies SET DEFAULT, then:

1) For every F, for each matching row MR in F, a transition is formed by pairing MR with the value formed by copying MR and setting each referencing column in the copy to the default value specified in the General Rules of Subclause 11.5, "<default clause>".

2) For every F, for every generated column GC in F that depends on some referencing column, for every site GCS corresponding to GC in the new row NR of a transition for F, let GCR be the result of evaluating, for NR, the generation expression included in the column descriptor of GC. The General Rules of Subclause 9.2, "Store assignment", are applied with GCS as $TARGET$ and GCR as $VALUE$.

3) The General Rules of Subclause 14.25, "Execution of BEFORE triggers", are applied with the following set of state changes $BTSS$:

A) For every F, for k ranging from 1 (one) to NSS, let SS_k be the k-th set of SS.

B) $BTSS$ contains a state change SC_k as follows:

I) The trigger event of SC_k is UPDATE.

II) The subject table of SC_k is F.

III) The column list of SC_k is SS_k.

IV) The set of transitions of SC_k is the set of transitions for F.

V) The set of statement-level triggers for which SC_k is considered as executed is empty.

VI) The set of row-level triggers consists of each row-level trigger that is activated by SC_k , paired with the empty set (of rows considered as executed).

4) For every matching row *MR* in every *F*, *F* is identified for replacement processing and *MR* is identified for replacement in *F*. The General Rules of Subclause Subclause 14.22, "Effect of replacing rows in base tables", are applied.

5) For every *F*, for *k* ranging from 1 (one) to *NSS*, let SS_k be the *k*-th set of *SS*.

Case:

A) If no SC_i has subject table *F*, trigger event UPDATE, and a column list that is SS_k, then a state change SC_j is added to *SSC* as follows:

I) The trigger event of SC_j is UPDATE.

II) The subject table of SC_j is *F*.

III) The column list of SC_j is SS_k.

IV) The set of transitions of SC_j is a the set of transitions for *F*.

V) The set of statement-level triggers for which SC_j is considered as executed is empty.

VI) The set of row-level triggers consists of each row-level trigger that is activated by SC_j, paired with the empty set (of rows considered as executed).

B) Otherwise, let SC_j be the state change in *SSC* that has subject table *F*, event UPDATE, and column list that is SS_k. The set of transitions of SC_j is the set of transitions for *F*.

iv) If the <delete rule> specifies RESTRICT and there exists some matching row, then an exception condition is raised: *integrity constraint violation — restrict violation.*

b) If PARTIAL is specified, then

Case:

i) If the <delete rule> specifies CASCADE, then for every *F*:

1) Every unique matching row in *F* is marked for deletion.

2) If no SC_i has subject table *F*, event DELETE, and an empty column list, then a state change SC_j is added to *SSC* as follows:

A) The trigger event of SC_j is DELETE.

B) The subject table of SC_j is *F*.

C) The column list of SC_j is empty.

D) The set of transitions of SC_j is empty.

NOTE 258 — The set of transitions will have been replaced by a nonempty set by the time any triggers activated by this state change are executed.

E) The set of statement-level triggers for which SC_j is considered as executed is empty.

F) The set of row-level triggers consists of each row-level trigger that is activated by SC_j, paired with the empty set (of rows considered as executed).

ii) If the <delete rule> specifies SET NULL, then:

1) For every *F*, for each unique matching row *UMR* in *F*, a transition is formed by pairing *UMR* with the value formed by copying *UMR* and setting each referencing column in the copy to the null value.

2) For every *F*, for every generated column *GC* in *F* that depends on some referencing column, for every site *GCS* corresponding to *GC* in the new row *NR* of a transition for *F*, let *GCR* be the result of evaluating, for *NR*, the generation expression included in the column descriptor of *GC*. The General Rules of Subclause 9.2, "Store assignment", are applied with *GCS* as *TARGET* and *GCR* as *VALUE*.

3) The General Rules of Subclause 14.25, "Execution of BEFORE triggers", are applied with the following set of state changes *BTSS*:

A) For every *F*, for *k* ranging from 1 (one) to *NSS*, let SS_k be the *k*-th set of *SS*.

B) *BTSS* contains a state change SC_k as follows:

I) The trigger event of SC_k is UPDATE.

II) The subject table of SC_k is *F*.

III) The column list of SC_k is SS_k.

IV) The set of transitions of SC_k is the set of transitions for *F*.

V) The set of statement-level triggers for which SC_k is considered as executed is empty.

VI) The set of row-level triggers consists of each row-level trigger that is activated by SC_k, paired with the empty set (of rows considered as executed).

4) For every unique matching row *UMR* in every *F*, *F* is identified for replacement processing and *UMR* is identified for replacement in *F*. The General Rules of Subclause 14.22, "Effect of replacing rows in base tables", are applied.

5) For every *F*, for *k* ranging from 1 (one) to *NSS*, let SS_k be the *k*-th set of *SS*.

Case:

A) If no SC_i has subject table *F*, trigger event UPDATE, and a column list that is SS_k, then a state change SC_j is added to *SSC* as follows:

I) The trigger event of SC_j is UPDATE.

II) The subject table of SC_j is F.

III) The column list of SC_j is SS_k.

IV) The set of transitions of SC_j is a the set of transitions for F.

V) The set of statement-level triggers for which SC_j is considered as executed is empty.

VI) The set of row-level triggers consists of each row-level trigger that is activated by SC_j, paired with the empty set (of rows considered as executed).

B) Otherwise, let SC_j be the state change in *SSC* that has subject table F, trigger event UPDATE, and column list that is SS_k. The set of transitions of SC_j is the set of transitions of F.

iii) If the <delete rule> specifies SET DEFAULT, then:

1) For every F, for each unique matching row *UMR* in F, a transition is formed pairing *UMR* with the value formed by copying *UMR* and setting each referencing column in the copy to the default value specified in the General Rules of Subclause 11.5, "<default clause>".

2) For every F, for every generated column *GC* in F that depends on some referencing column, for every site *GCS* corresponding to *GC* in the new row *NR* of a transition for F, let *GCR* be the result of evaluating, for *NR*, the generation expression included in the column descriptor of *GC*. The General Rules of Subclause 9.2, "Store assignment", are applied with *GCS* as *TARGET* and *GCR* as *VALUE*.

3) The General Rules of Subclause 14.25, "Execution of BEFORE triggers", are applied with the following set of state changes *BTSS*:

A) For every F, for k ranging from 1 (one) to *NSS*, let SS_k be the k-th set of *SS*.

B) *BTSS* contains a state change SC_k as follows:

I) The trigger event of SC_k is UPDATE.

II) The subject table of SC_k is F.

III) The column list of SC_k is SS_k.

IV) The set of transitions of SC_k is the set of transitions for F.

V) The set of statement-level triggers for which SC_k is considered as executed is empty.

VI) The set of row-level triggers consists of each row-level trigger that is activated by SC_k, paired with the empty set (of rows considered as executed).

4) For every unique matching row *UMR* in every F, F is identified for replacement processing and *UMR* is identified for replacement in F. The General Rules of Subclause 14.22, "Effect of replacing rows in base tables", are applied.

5) For every *F*, for *k* ranging from 1 (one) to *NSS*, let SS_k be the *k*-th set of *SS*.

   Case:

   A) If no SC_i has subject table *F*, trigger event UPDATE, and a column list that is SS_k, then a new state change SC_j is added to *SSC* as follows:

      I) The trigger event of SC_j is UPDATE.

      II) The subject table of SC_j is *F*.

      III) The column list of SC_j is SS_k.

      IV) The set of transitions of SC_j is the set of transitions for *F*.

      V) The set of statement-level triggers for which SC_k is considered as executed is empty.

      VI) The set of row-level triggers consists of each row-level trigger that is activated by SC_k, paired with the empty set (of rows considered as executed).

   B) Otherwise, let SC_j be the state change in *SSC* that has subject table *F*, trigger event UPDATE, and a column list that is SS_k. The set of transitions of SC_j is the set of transitions for *F*.

iv) If the <delete rule> specifies RESTRICT and there exists some unique matching row, then an exception condition is raised: *integrity constraint violation — restrict violation.*

NOTE 259 — Otherwise, the <referential action> is not performed.

8) If a non-null value of a referenced column *RC* in the referenced table is updated to a value that is distinct from the current value of *RC*, then for every member *F* of the subtable family of the referencing table:

Case:

a) If SIMPLE is specified or implicit, or if FULL is specified, then

   Case:

   i) If the <update rule> specifies CASCADE, then:

      1) For every *F*, for each matching row *MR* in *F*, a transition is formed by pairing *MR* with the value formed by copying *MR* and setting each referencing column in the copy that corresponds with a referenced column to the new value of that referenced column.

      2) For every *F*, for every generated column *GC* in *F* that depends on some referencing column, for every site *GCS* corresponding to *GC* in the new row *NR* of a transition for *F*, let *GCR* be the result of evaluating, for *NR*, the generation expression included in the column descriptor of *GC*. The General Rules of Subclause 9.2, "Store assignment", are applied with *GCS* as *TARGET* and *GCR* as *VALUE.*

      3) The General Rules of Subclause 14.25, "Execution of BEFORE triggers", are applied with the following set of state changes *BTSS*:

         A) For every *F*, for *k* ranging from 1 (one) to *PNSS*, let SS_k be the *k*-th set of *PSS*.

B) *BTSS* contains a state change SC_k as follows:

I) The trigger event of SC_k is UPDATE.

II) The subject table of SC_k is *F*.

III) The column list of SC_k is SS_k.

IV) The set of transitions of SC_k is the set of transitions for *F*.

V) The set of statement-level triggers for which SC_k is considered as executed is empty.

VI) The set of row-level triggers consists of each row-level trigger that is activated by SC_k, paired with the empty set (of rows considered as executed).

4) For every matching row *MR* in every *F*, *F* is identified for replacement processing and *MR* is identified for replacement in *F*.

5) For every *F*, for *k* ranging from 1 (one) to *PNSS*, let SS_k be the *k*-th set of *PSS*.

Case:

A) If no SC_i has subject table *F*, trigger event UPDATE, and a column list that is SS_k, then a state change SC_j is added to *SSC* as follows:

I) The trigger event of SC_j is UPDATE.

II) The subject table of SC_j is *F*.

III) The column list of SC_j is SS_k.

IV) The set of transitions of SC_j is the set of transitions for *F*.

V) The set of statement-level triggers for which SC_j is considered as executed is empty.

VI) The set of row-level triggers consists of each row-level trigger that is activated by SC_j, paired with the empty set (of rows considered as executed).

B) Otherwise, let SC_j be the state change in *SSC* that has subject table *F*, trigger event UPDATE, and a column list that is SS_k. The set of transitions of SC_j is the set of transitions for *F*.

ii) If the <update rule> specifies SET NULL, then

Case:

1) If SIMPLE is specified or implicit, then:

A) For every *F*, for each matching row *MR* in *F*, a transition is formed by pairing *MR* with the value formed by copying *MR* and setting each referencing column in the copy to the null value.

B) For every *F*, for every generated column *GC* in *F* that depends on some referencing column, for every site *GCS* corresponding to *GC* in the new row *NR* of a transition for *F*, let *GCR* be the result of evaluating, for *NR*, the generation expression included in the column descriptor of *GC*. The General Rules of Subclause 9.2, "Store assignment", are applied with *GCS* as *TARGET* and *GCR* as *VALUE*.

C) The General Rules of Subclause 14.25, "Execution of BEFORE triggers", are applied with the following set of state changes *BTSS*:

I) For every *F*, for *k* ranging from 1 (one) to *PNSS*, let SS_k be the *k*-th set of *PSS*.

II) *BTSS* contains a state change SC_k as follows:

1) The trigger event of SC_k is UPDATE.

2) The subject table of SC_k is *F*.

3) The column list of SC_k is SS_k.

4) The set of transitions of SC_k is the set of transitions for *F*.

5) The set of statement-level triggers for which SC_k is considered as executed is empty.

6) The set of row-level triggers consists of each row-level trigger that is activated by SC_k, paired with the empty set (of rows considered as executed).

D) For every matching row *MR* in every *F*, *F* is identified for replacement processing and *MR* is identified for replacement in *F*.

E) For every *F*, for *k* ranging from 1 (one) to *NSS*, let SS_k be the *k*-th set of *SS*.

Case:

I) If no SC_i has subject table *F*, trigger event UPDATE, and a column list that is SS_k, then a state change SC_j is added to *SSC* as follows:

1) The trigger event of SC_j is UPDATE.

2) The subject table of SC_j is *F*.

3) The column list of SC_j is SS_k.

4) The set of transitions of SC_j is the set of transitions for *F*.

5) The set of statement-level triggers for which SC_j is considered as executed is empty.

6) The set of row-level triggers consists of each row-level trigger that is activated by SC_j, paired with the empty set (of rows considered as executed).

II) Otherwise, let SC_j be the state change in *SSC* that has subject table *F*, trigger event UPDATE, and a column list that is SS_k. The set of transitions of SC_j is the set of transitions of *F*.

2) If <match type> specifies FULL, then:

A) For every *F*, for each matching row *MR* in *F*, a transition is formed by pairing *MR* with the value formed by copying *MR* and setting each referencing column in the copy that corresponds with a referenced column to the null value.

B) For every *F*, for every generated column *GC* in *F* that depends on some referencing column, for every site *GCS* corresponding to *GC* in the new row *NR* of a transition for *F*, let *GCR* be the result of evaluating, for *NR*, the generation expression included in the column descriptor of *GC*. The General Rules of Subclause 9.2, "Store assignment", are applied with *GCS* as *TARGET* and *GCR* as *VALUE*.

C) The General Rules of Subclause 14.25, "Execution of BEFORE triggers", are applied with the following set of state changes *BTSS*:

I) For every *F*, for *k* ranging from 1 (one) to *NSS*, let SS_k be the *k*-th set of *SS*.

II) *BTSS* contains a state change SC_k as follows:

1) The trigger event of SC_k is UPDATE.

2) The subject table of SC_k is *F*.

3) The column list of SC_k is SS_k.

4) The set of transitions of SC_k is the set of transitions for *F*.

5) The set of statement-level triggers for which SC_k is considered as executed is empty.

6) The set of row-level triggers consists of each row-level trigger that is activated by SC_k, paired with the empty set (of rows considered as executed).

D) For every matching row *MR* in every *F*, *F* is identified for replacement processing and *MR* is identified for replacement in *F*.

E) For every *F*, for *k* ranging from 1 (one) to *NSS*, let SS_k be the *k*-th set of *SS*.

Case:

I) If no SC_i has subject table *F*, trigger event UPDATE, and a column list that is SS_k, then a state change SC_j is added to *SSC* as follows:

1) The trigger event of SC_j is UPDATE.

2) The subject table of SC_j is *F*.

3) The column list of SC_j is SS_k.

4) The set of transitions of SC_j is the set of transitions for F.

5) The set of statement-level triggers for which SC_j is considered as executed is empty.

6) The set of row-level triggers consists of each row-level trigger that is activated by SC_j, paired with the empty set (of rows considered as executed).

II) Otherwise, let SC_j be the state change in *SSC* that has subject table F, trigger event UPDATE, and a column list that is SS_k. The set of transitions of SC_j is the set of transitions for F.

iii) If the <update rule> specifies SET DEFAULT, then:

1) For every *F*, for each matching row *MR* in *F*, a transition is formed by pairing *MR* with the value formed by copying *MR* and setting each referencing column in the copy that corresponds with a referenced column to the default value specified in the General Rules of Subclause 11.5, "<default clause>".

2) For every *F*, for every generated column *GC* in *F* that depends on some referencing column, for every site *GCS* corresponding to *GC* in the new row *NR* of a transition for *F*, let *GCR* be the result of evaluating, for *NR*, the generation expression included in the column descriptor of *GC*. The General Rules of Subclause 9.2, "Store assignment", are applied with *GCS* as *TARGET* and *GCR* as *VALUE*.

3) The General Rules of Subclause 14.25, "Execution of BEFORE triggers", are applied with the following set of state changes *BTSS*:

A) For every *F*, for *k* ranging from 1 (one) to *PNSS*, let SS_k be the *k*-th set of *PSS*.

B) *BTSS* contains a state change SC_k as follows:

I) The trigger event of SC_k is UPDATE.

II) The subject table of SC_k is F.

III) The column list of SC_k is SS_k.

IV) The set of transitions of SC_k is the set of transitions for F.

V) The set of statement-level triggers for which SC_k is considered as executed is empty.

VI) The set of row-level triggers consists of each row-level trigger that is activated by SC_k, paired with the empty set (of rows considered as executed).

4) For every matching row *MR* in every *F*, *F* is identified for replacement processing and *MR* is identified for replacement in *F*.

5) For every *F*, for *k* ranging from 1 (one) to *PNSS*, let SS_k be the *k*-th set of *PSS*.

Case:

A) If no SC_i has subject table F, trigger event UPDATE, and a column list that is SS_k, then a state change SC_j is added to SSC as follows:

I) The trigger event of SC_j is UPDATE.

II) The subject table of SC_j is F.

III) The column list of SC_j is SS_k.

IV) The set of transitions of SC_j is the set of transitions for F.

V) The set of statement-level triggers for which SC_j is considered as executed is empty.

VI) The set of row-level triggers consists of each row-level trigger that is activated by SC_j, paired with the empty set (of rows considered as executed).

B) Otherwise, let SC_j be the state change in SSC that has subject table F, event UPDATE, and a column list that is SS_k. The set of transitions of SC_j is the set of transitions for F.

iv) If the <update rule> specifies RESTRICT and there exists some matching row, then an exception condition is raised: *integrity constraint violation — restrict violation.*

b) If PARTIAL is specified, then

Case:

i) If the <update rule> specifies CASCADE, then:

1) For every F, for each unique matching row UMR in F that contains a non-null value in the referencing column $C1$ in F that corresponds to the updated referenced column $C2$, a transition is formed by pairing UMR with the value formed by copying UMR and setting $C1$ in the copy to the new value V of $C2$, provided that, in all updated rows in the referenced table that formerly had, during the same execution of the same innermost SQL-statement, that unique matching row as a matching row, the values in $C2$ have all been updated to a value that is not distinct from V. If this last condition is not satisfied, then an exception condition is raised: *triggered data change violation.*

NOTE 260 — Because of the Rules of Subclause 8.2, "<comparison predicate>", on which the definition of "distinct" relies, the values in $C2$ may have been updated to values that are not distinct, yet are not identical. Which of these non-distinct values is used for the cascade operation is implementation-dependent.

2) For every F, for every generated column GC in F that depends on some referencing column, for every site GCS corresponding to GC in the new row NR of a transition for F, let GCR be the result of evaluating, for NR, the generation expression included in the column descriptor of GC. The General Rules of Subclause 9.2, "Store assignment", are applied with GCS as $TARGET$ and GCR as $VALUE$.

3) The General Rules of Subclause 14.25, "Execution of BEFORE triggers", are applied with the following set of state changes $BTSS$:

A) For every F, for k ranging from 1 (one) to $UNSS$, let SS_k be the k-th set of USS.

B) $BTSS$ contains a state change SC_k as follows:

I) The trigger event of SC_k is UPDATE.

II) The subject table of SC_k is F.

III) The column list of SC_k is SS_k.

IV) The set of transitions of SC_k is the set of transitions for F.

V) The set of statement-level triggers for which SC_k is considered as executed is empty.

VI) The set of row-level triggers consists of each row-level trigger that is activated by SC_k, paired with the empty set (of rows considered as executed).

4) For every unique matching row *UMR* in every *F*, *F* is identified for replacement processing and *UMR* is identified for replacement in *F*.

5) For every *F*, for *k* ranging from 1 (one) to *UNSS*, let SS_k be the *k*-th set of *USS*.

Case:

A) If no SC_i has subject table *F*, trigger event UPDATE, and a column list that is SS_k, then a state change SC_j is added to *SSC* as follows:

I) The trigger event of SC_j is UPDATE.

II) The subject table of SC_j is F.

III) The column list of SC_j is SS_k.

IV) The set of transitions of SC_j is the set of transitions for F.

V) The set of statement-level triggers for which SC_j is considered as executed is empty.

VI) The set of row-level triggers consists of each row-level trigger that is activated by SC_j, paired with the empty set (of rows considered as executed).

B) Otherwise, let SC_j be the state change in *SSC* that has subject table *F*, trigger event UPDATE, and a column list that is SS_k. The set of transitions of SC_j is the set of transitions for *F*.

ii) If the <update rule> specifies SET NULL, then:

1) For every *F*, for each unique matching row *UMR* in *F* that contains a non-null value in the referencing column in *F* that corresponds with the updated referenced column, a transition is formed by pairing *UMR* with the value formed by copying *UMR* and setting that referencing column in the copy to the null value.

2) For every *F*, for every generated column *GC* in *F* that depends on some referencing column, for every site *GCS* corresponding to *GC* in the new row *NR* of a transition for *F*, let *GCR* be the result of evaluating, for *NR*, the generation expression included in the column

descriptor of *GC*. The General Rules of Subclause 9.2, "Store assignment", are applied with *GCS* as *TARGET* and *GCR* as *VALUE*.

3) The General Rules of Subclause 14.25, "Execution of BEFORE triggers", are applied with the following set of state changes *BTSS*:

   A) For every *F*, for *k* ranging from 1 (one) to *UNSS*, let SS_k be the *k*-th set of *USS*.

   B) *BTSS* contains a state change SC_k as follows:

      I) The trigger event of SC_k is UPDATE.

      II) The subject table of SC_k is *F*.

      III) The column list of SC_k is SS_k.

      IV) The set of transitions of SC_k is the set of transitions for *F*.

      V) The set of statement-level triggers for which SC_k is considered as executed is empty.

      VI) The set of row-level triggers consists of each row-level trigger that is activated by SC_k, paired with the empty set (of rows considered as executed).

4) For every unique matching row *UMR* in every *F*, *F* is identified for replacement processing and *UMR* is identified for replacement in *F*.

5) For every *F*, for *k* ranging from 1 (one) to *NSS*, let SS_k be the *k*-th set of *SS*.

   Case:

   A) If no SC_i has subject table *F*, trigger event UPDATE, and a column list that is SS_k, then a state change SC_j is added to *SSC* as follows:

      I) The trigger event of SC_j is UPDATE.

      II) The subject table of SC_j is *F*.

      III) The column list of SC_j is SS_k.

      IV) The set of transitions of SC_j is the set of transitions for *F*.

      V) The set of statement-level triggers for which SC_j is considered as executed is empty.

      VI) The set of row-level triggers consists of each row-level trigger that is activated by SC_j, paired with the empty set (of rows considered as executed).

   B) Otherwise, let SC_j be the state change in *SSC* that has subject table *F*, trigger event UPDATE, and a column list that is SS_k. The set of transitions of SC_j is the set of transitions for *F*.

iii) If the <update rule> specifies SET DEFAULT, then:

1) For every *F*, for each unique matching row *UMR* in *F* that contains a non-null value in the referencing column in *F* that corresponds with the updated referenced column, a transition is formed by pairing *UMR* with the value formed by copying *UMR* and setting that referencing column in the copy to the default value specified in the General Rules of Subclause 11.5, "<default clause>".

2) For every *F*, for every generated column *GC* in *F* that depends on some referencing column, for every site *GCS* corresponding to *GC* in the new row *NR* of a transition for *F*, let *GCR* be the result of evaluating, for *NR*, the generation expression included in the column descriptor of *GC*. The General Rules of Subclause 9.2, "Store assignment", are applied with *GCS* as *TARGET* and *GCR* as *VALUE*.

3) The General Rules of Subclause 14.25, "Execution of BEFORE triggers", are applied with the following set of state changes *BTSS*:

   A) For every *F*, for *k* ranging from 1 (one) to *UNSS*, let SS_k be the *k*-th set of *USS*.

   B) *BTSS* contains a state change SC_k as follows:

      I) The trigger event of SC_k is UPDATE.

      II) The subject table of SC_k is *F*.

      III) The column list of SC_k is SS_k.

      IV) The set of transitions of SC_k is the set of transitions for *F*.

      V) The set of statement-level triggers for which SC_k is considered as executed is empty.

      VI) The set of row-level triggers consists of each row-level trigger that is activated by SC_k, paired with the empty set (of rows considered as executed).

4) For every unique matching row *UMR* in every *F*, *F* is identified for replacement processing and *UMR* is identified for replacement in *F*.

5) For every *F*, for *k* ranging from 1 (one) to *UNSS*, let SS_k be the *k*-th set of *USS*.

   Case:

   A) If no SC_i has subject table *F*, trigger event UPDATE, and a column list that is SS_k, then a state change SC_j is added to *SSC* as follows:

      I) The trigger event of SC_j is UPDATE.

      II) The subject table of SC_j is *F*.

      III) The column list of SC_j is SS_k.

      IV) The set of transitions of SC_j is the set of transitions for *F*.

      V) The set of statement-level triggers for which SC_j is considered as executed is empty.

VI) The set of row-level triggers consists of each row-level trigger that is activated by SC_j, paired with the empty set (of rows considered as executed).

B) Otherwise, let SC_j be the state change in *SSC* that has subject table *F*, trigger event UPDATE, and a column list that is SS_k. The set of transitions of SC_j is the set of transitions for *F*.

iv) If the <update rule> specifies RESTRICT and there exists some unique matching row, then an exception condition is raised: *integrity constraint violation — restrict violation.*

NOTE 261 — Otherwise, the <referential action> is not performed.

9) Let *ISS* be the innermost SQL-statement being executed.

10) If evaluation of these General Rules during the execution of *ISS* would cause an update of some site to a value that is distinct from the value to which that site was previously updated during the execution of *ISS*, then an exception condition is raised: *triggered data change violation.*

11) If evaluation of these General Rules during the execution of *ISS* would cause deletion of a row containing a site that is identified for replacement in that row, then an exception condition is raised: *triggered data change violation.*

12) If evaluation of these General Rules during the execution of *ISS* would cause either an attempt to update a row that has been deleted by any <delete statement: positioned> or <dynamic delete statement: positioned> that identifies some cursor *CR* that is still open or has been updated by any <update statement: positioned> or <dynamic delete statement: positioned> that identifies some cursor *CR* that is still open, or an attempt to mark for deletion such a row, then a completion condition is raised: *warning — cursor operation conflict.*

13) For every row *RMD* that is marked for deletion, every subrow of *RMD* and every superrow of *RMD* is marked for deletion.

14) If any table *T* is the subject table of a state change in *SSC* that has been created or modified during evaluation of the preceding General Rules of this subclause, then, for every referential constraint descriptor, the preceding General Rules of this subclause are applied.

NOTE 262 — Thus these rules are repeatedly evaluated until no further transitions are generated.

15) The General Rules of Subclause 14.25, "Execution of BEFORE triggers", are applied with the following set of state changes *BTSS*.

For each table F_i that contains a row that is marked for deletion, *BTSS* contains a state change SC_i as follows:

a) The trigger event of SC_i is DELETE.

b) The subject table of SC_i is F_i.

c) The column list of SC_i is empty.

d) The set of transitions of SC_i is a copy of the set of rows in F_i that are marked for deletion.

e) The set of statement-level triggers for which SC_i is considered as executed is empty.

f) The set of row-level triggers consists of each row-level trigger that is activated by SC_i, paired with the empty set (of rows considered as executed).

16) For each table F_i that contains a row that is marked for deletion, let SC_j be the state change in *SSC* that has subject table F_i, trigger event DELETE, and an empty column list. A copy of the rows in F_i that are marked for deletion constitutes the set of transitions of SC_j.

Conformance Rules

1) Without Feature T191, "Referential action RESTRICT", conforming SQL language shall not contain a <referential action> that contains RESTRICT.
2) Without Feature F741, "Referential MATCH types", conforming SQL language shall not contain a <references specification> that contains MATCH.
3) Without Feature F191, "Referential delete actions", conforming SQL language shall not contain a <delete rule>.
4) Without Feature F701, "Referential update actions", conforming SQL language shall not contain an <update rule>.
5) Without Feature T201, "Comparable data types for referential constraints", conforming SQL language shall not contain a <referencing columns> in which the data type of each referencing column is not the same as the data type of the corresponding referenced column.

   NOTE 263 — The Conformance Rules of Subclause 9.10, "Grouping operations", also apply.

11.9 <check constraint definition>

Function

Specify a condition for the SQL-data.

Format

```
<check constraint definition> ::= CHECK <left paren> <search condition> <right paren>
```

Syntax Rules

1) The <search condition> shall not contain a <target specification>.

2) The <search condition> shall not contain a <set function specification> that is not contained in a <subquery>.

3) If <check constraint definition> is contained in a <table definition> or <alter table statement>, then let *T* be the table identified by the containing <table definition> or <alter table statement>.

   Case:

   a) If *T* is a persistent base table, or if the <check constraint definition> is contained in a <domain definition> or <alter domain statement>, then no <table reference> generally contained in the <search condition> shall reference a temporary table.

   b) If *T* is a global temporary table, then no <table reference> generally contained in the <search condition> shall reference a table other than a global temporary table.

   c) If *T* is a created local temporary table, then no <table reference> generally contained in the <search condition> shall reference a table other than either a global temporary table or a created local temporary table.

   d) If *T* is a declared local temporary table, then no <table reference> generally contained in the <search condition> shall reference a persistent base table.

4) If the <check constraint definition> is contained in a <table definition> that defines a temporary table and specifies ON COMMIT PRESERVE ROWS or a <temporary table declaration> that specifies ON COMMIT PRESERVE ROWS, then no <subquery> in the <search condition> shall reference a temporary table defined by a <table definition> or a <temporary table declaration> that specifies ON COMMIT DELETE ROWS.

5) The <search condition> shall simply contain a <boolean value expression> that is retrospectively deterministic.

   NOTE 264 — "retrospectively deterministic" is defined in Subclause 6.34, "<boolean value expression>".

6) The <search condition> shall not generally contain a <routine invocation> whose subject routine is an SQL-invoked routine that possibly modifies SQL-data.

7) Let *A* be the <authorization identifier> that owns *T*.

Access Rules

None.

General Rules

1) A <check constraint definition> defines a check constraint.

   NOTE 265 — Subclause 10.8, "<constraint name definition> and <constraint characteristics>", specifies when a constraint is effectively checked. The General Rules that control the evaluation of a check constraint can be found in either Subclause 11.6, "<table constraint definition>", or Subclause 11.24, "<domain definition>", depending on whether it forms part of a table constraint or a domain constraint.

2) If the character representation of the <search condition> cannot be represented in the Information Schema without truncation, then a completion condition is raised: *warning — search condition too long for information schema*.

   NOTE 266 — The Information Schema is defined in ISO/IEC 9075-11.

Conformance Rules

1) Without Feature F671, "Subqueries in CHECK constraints", conforming SQL language shall not contain a <search condition> contained in a <check constraint definition> that contains a <subquery>.

2) Without Feature F672, "Retrospective check constraints", conforming SQL language shall not contain a <check constraint definition> that generally contains CURRENT_DATE, CURRENT_TIMESTAMP, or LOCALTIMESTAMP.

11.10 <alter table statement>

Function

Change the definition of a table.

Format

```
<alter table statement> ::= ALTER TABLE <table name> <alter table action>

<alter table action> ::=
    <add column definition>
  | <alter column definition>
  | <drop column definition>
  | <add table constraint definition>
  | <drop table constraint definition>
```

Syntax Rules

1) Let *T* be the table identified by the <table name>.
2) The schema identified by the explicit or implicit schema name of the <table name> shall include the descriptor of *T*.
3) The scope of the <table name> is the entire <alter table statement>.
4) *T* shall be a base table.
5) *T* shall not be a declared local temporary table.

Access Rules

1) The enabled authorization identifiers shall include the <authorization identifier> that owns the schema identified by the <schema name> of the table identified by <table name>.

General Rules

1) The base table descriptor of *T* is modified as specified by <alter table action>.
2) If <add column definition> or <drop column definition> is specified, then the row type *RT* of *T* is the set of pairs (<field name>, <data type>) where <field name> is the name of a column *C* of *T* and <data type> is the declared type of *C*. This set of pairs contains one pair for each column of *T* in the order of their ordinal position in *T*.

Conformance Rules

None.

11.11 <add column definition>

Function

Add a column to a table.

Format

```
<add column definition> ::= ADD [ COLUMN ] <column definition>
```

Syntax Rules

1) Let *T* be the table identified by the <table name> immediately contained in the containing <alter table statement>.

2) *T* shall not be a referenceable table.

3) If <column definition> contains <identity column specification>, then the table descriptor of *T* shall not include a column descriptor of an identity column.

Access Rules

None.

General Rules

1) The column defined by the <column definition> is added to *T*.

2) Let *C* be the column added to *T*.

   Case:

   a) If *C* is a generated column, then let *TN* be the <table name> immediately contained in the containing <alter table statement>, let *CN* be the <column name> immediately contained in <column definition>, and let *GE* be the generation expression included in the column descriptor of *C*. The following <update statement: searched> is executed without further Syntax Rule or Access Rule checking:

      ```
      UPDATE TN SET CN = GE
      ```

   b) Otherwise, *C* is a base column.

      Case:

      i) If *C* is an identity column, then for each row in *T* let *CS* be the site corresponding to *C* and let *NV* be the result of applying the General Rules of Subclause 9.21, "Generation of the next value of a sequence generator", with the sequence descriptor included in the column descriptor of *C* as *SEQUENCE*.

         Case:

1) If the declared type of *C* is a distinct type *DIST*, then let *CNV* be *DIST*(*NV*).

2) Otherwise, let *CNV* be *NV*.

The General Rules of Subclause 9.2, "Store assignment", are applied with *CS* as *TARGET* and *CNV* as *VALUE*.

ii) Otherwise, every value in *C* is the default value for *C*.

NOTE 267 — The default value of a column is defined in Subclause 11.5, "<default clause>".

NOTE 268 — The addition of a column to a table has no effect on any existing <query expression> included in a view descriptor, <triggered action> included in a trigger descriptor, or <search condition> included in a constraint descriptor because any implicit column references in these descriptor elements are syntactically substituted by explicit column references under the Syntax Rules of Subclause 7.12, "<query specification>". Furthermore, by implication (from the lack of any General Rules to the contrary), the meaning of a column reference is never retroactively changed by the addition of a column subsequent to the invocation of the <SQL schema statement> containing that column reference.

3) For every table privilege descriptor that specifies *T* and a privilege of SELECT, UPDATE, INSERT or REFERENCES, a new column privilege descriptor is created that specifies *T*, the same action, grantor, and grantee, and the same grantability, and specifies the <column name> of the <column definition>.

4) In all other respects, the specification of a <column definition> in an <alter table statement> has the same effect as specification of the <column definition> in the <table definition> for *T* would have had. In particular, the degree of *T* is increased by 1 (one) and the ordinal position of that column is equal to the new degree of *T* as specified in the General Rules of Subclause 11.4, "<column definition>".

Conformance Rules

None.

11.12 <alter column definition>

Function

Change a column and its definition.

Format

```
<alter column definition> ::=
    ALTER [ COLUMN ] <column name> <alter column action>

<alter column action> ::=
    <set column default clause>
  | <drop column default clause>
  | <add column scope clause>
  | <drop column scope clause>
  | <alter identity column specification>
```

Syntax Rules

1) Let *T* be the table identified in the containing <alter table statement>.

2) Let *C* be the column identified by the <column name>.

3) *C* shall be a column of *T*.

4) If *C* is the self-referencing column of *T* or *C* is a generated column of *T*, then <alter column action> shall not contain <add column scope clause> or <drop column scope clause>.

5) If *C* is an identity column, then <alter column action> shall contain <alter identity column specification>.

6) If <alter identity column specification> is specified, then *C* shall be an identity column.

Access Rules

None.

General Rules

1) The column descriptor of *C* is modified as specified by <alter column action>.

Conformance Rules

1) Without Feature F381, "Extended schema manipulation", conforming SQL language shall not contain an <alter column definition>.

11.13 <set column default clause>

Function

Set the default clause for a column.

Format

```
<set column default clause> ::= SET <default clause>
```

Syntax Rules

None.

Access Rules

None.

General Rules

1) Let *C* be the column identified by the <column name> in the containing <alter column definition>.
2) The default value specified by the <default clause> is placed in the column descriptor of *C*.

Conformance Rules

1) Without Feature F381, "Extended schema manipulation", conforming SQL language shall not contain a <set column default clause>.

11.14 <drop column default clause>

Function

Drop the default clause from a column.

Format

```
<drop column default clause> ::= DROP DEFAULT
```

Syntax Rules

1) Let *C* be the column identified by the <column name> in the containing <alter column definition>.
2) The descriptor of *C* shall include a default value.

Access Rules

None.

General Rules

1) The default value is removed from the column descriptor of *C*.

Conformance Rules

1) Without Feature F381, "Extended schema manipulation", conforming SQL language shall not contain a <drop column default clause>.

11.15 <add column scope clause>

Function

Add a non-empty scope for an existing column of data type REF in a base table.

Format

```
<add column scope clause> ::= ADD <scope clause>
```

Syntax Rules

1) Let *C* be the column identified by the <column name> in the containing <alter column definition>. The declared type of *C* shall be some reference type. Let *RTD* be the reference type descriptor included in the descriptor of *C*.

2) Let *T* be the table identified by the <table name> in the containing <alter table statement>. If *T* is a referenceable table, then *C* shall be an originally-defined column of *T*.

3) *RTD* shall not include a scope.

4) Let *UDTN* be the name of the referenced type included in *RTD*.

5) The <table name> *STN* contained in the <scope clause> shall identify a referenceable table whose structured type is *UDTN*.

Access Rules

None.

General Rules

1) *STN* is included as the scope in the reference type descriptor included in the column descriptor of *C*.

2) For any proper subtable *PST* of *T*, let *PSC* be the column whose corresponding column in *T* is *C*. *STN* is included as the scope in the reference type descriptor included in the column descriptor of *PSC*.

Conformance Rules

1) Without Feature F381, "Extended schema manipulation", conforming SQL language shall not contain an <add column scope clause>.

2) Without Feature S043, "Enhanced reference types", conforming SQL language shall not contain an <add column scope clause>.

11.16 <drop column scope clause>

Function

Drop the scope from an existing column of data type REF in a base table.

Format

```
<drop column scope clause> ::= DROP SCOPE <drop behavior>
```

Syntax Rules

1) Let *C* be the column identified by the <column name> in the containing <alter column definition>. The declared type of *C* shall be some reference type whose reference type descriptor includes a scope.

2) Let *T* be the table identified by the <table name> in the containing <alter table statement>. If *T* is a referenceable table, then *C* shall be an originally-defined column of *T*.

3) Let *SC* be the set of columns consisting of *C* and, for every proper subtable of *T*, the column whose super-column is *C*.

4) An *impacted dereference operation* is a <dereference operation> whose <reference value expression> is a column reference that identifies a column in *SC*, a <method reference> whose <value expression primary> is a column reference that identifies a column in *SC*, or a <reference resolution> whose <reference value expression> is a column reference that identifies a column in *SC*.

5) If RESTRICT is specified, then no impacted dereference operation shall be contained in any of the following:

   a) The SQL routine body of any routine descriptor.

   b) The <query expression> of any view descriptor.

   c) The <search condition> of any constraint descriptor.

   d) The triggered action of any trigger descriptor.

   NOTE 269 — If CASCADE is specified, then such referencing objects will be dropped by the execution of the <SQL procedure statement>s specified in the General Rules of this Subclause.

Access Rules

None.

General Rules

1) For every SQL-invoked routine *R* whose routine descriptor includes an SQL routine body that contains an impacted dereference operation, let *SN* be the <specific name> of *R*. The following <drop routine statement> is effectively executed for every *R* without further Access Rule checking:

```
DROP SPECIFIC ROUTINE SN CASCADE
```

2) For every view *V* whose view descriptor includes a <query expression> that contains an impacted dereference operation, let *VN* be the <table name> of *V*. The following <drop view statement> is effectively executed for every *V* without further Access Rule checking:

```
DROP VIEW VN CASCADE
```

3) For every assertion *A* whose assertion descriptor includes a <search condition> that contains an impacted dereference operation, let *AN* be the <constraint name> of *A*. The following <drop assertion statement> is effectively executed for every *A* without further Access Rule checking:

```
DROP ASSERTION AN CASCADE
```

4) For every table check constraint *CC* whose table check constraint descriptor includes a <search condition> that contains an impacted dereference operation, let *CN* be the <constraint name> of *CC* and let *TN* be the <table name> of the table whose descriptor includes descriptor of *CC*. The following <alter table statement> is effectively executed for every *CC* without further Access Rule checking:

```
ALTER TABLE TN DROP CONSTRAINT CN CASCADE
```

5) The scope included in the reference type descriptor included in the column descriptor of every column in *SC* is made empty.

Conformance Rules

1) Without Feature F381, "Extended schema manipulation", conforming SQL language shall not contain a <drop column scope clause>.

2) Without Feature S043, "Enhanced reference types", conforming SQL language shall not contain a <drop column scope clause>.

11.17 <alter identity column specification>

Function

Change the options specified for an identity column.

Format

```
<alter identity column specification> ::= <alter identity column option>...

<alter identity column option> ::=
    <alter sequence generator restart option>
  | SET <basic sequence generator option>
```

Syntax Rules

1) Let *SEQ* be the sequence generator descriptor included in the column descriptor identified by the <column name> in the containing <alter column definition>.

2) Let *OPT* be the character string formulated from <alter identity column specification> that conforms to the Format of <alter sequence generator options>.

   NOTE 270 — *OPT* is formulated by removing all instances of the keyword SET from the string corresponding to from <alter identity column specification>.

3) The Syntax Rules of Subclause 9.23, "Altering a sequence generator", are applied with *OPT* as *OPTIONS* and *SEQ* as *SEQUENCE*.

Access Rules

None.

General Rules

1) The General Rules of Subclause 9.23, "Altering a sequence generator", are applied with *OPT* as *OPTIONS* and *SEQ* as *SEQUENCE*.

Conformance Rules

1) Without Feature T174, "Identity columns", an <alter column definition> shall not contain an <alter identity column specification>.

11.18 <drop column definition>

Function

Destroy a column of a base table.

Format

```
<drop column definition> ::= DROP [ COLUMN ] <column name> <drop behavior>
```

Syntax Rules

1) Let *T* be the table identified by the <table name> in the containing <alter table statement> and let *TN* be the name of *T*.
2) Let *C* be the column identified by the <column name> *CN*.
3) *T* shall not be a referenceable table.
4) *C* shall be a column of *T* and *C* shall not be the only column of *T*.
5) If RESTRICT is specified, then *C* shall not be referenced in any of the following:
   a) The <query expression> of any view descriptor.
   b) The <search condition> of any constraint descriptor other than a table constraint descriptor that contains references to no other column and that is included in the table descriptor of *T*.
   c) The SQL routine body of any routine descriptor.
   d) Either an explicit trigger column list or a triggered action column set of any trigger descriptor.
   e) The generation expression of any column descriptor.

   NOTE 271 — A <drop column definition> that does not specify CASCADE will fail if there are any references to that column resulting from the use of CORRESPONDING, NATURAL, SELECT * (except where contained in an exists predicate>), or REFERENCES without a <reference column list> in its <referenced table and columns>.

   NOTE 272 — If CASCADE is specified, then any such dependent object will be dropped by the execution of the <revoke statement> specified in the General Rules of this Subclause.

Access Rules

None.

General Rules

1) Let *TR* be the trigger name of any trigger descriptor having an explicit trigger column list or a triggered action column set that contains *CN*. The following <drop trigger statement> is effectively executed without further Access Rule checking:

```
DROP TRIGGER TR
```

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

```
REVOKE INSERT(CN), UPDATE(CN), SELECT(CN), REFERENCES(CN) ON TABLE TN
FROM A CASCADE
```

3) Let *GC* be any generated column of *T* in whose descriptor the generation expression contains a <column reference> that references *C*. The following <alter table statement> is effectively executed without further Access Rule checking:

```
ALTER TABLE T DROP COLUMN GC CASCADE
```

4) If the column is not based on a domain, then its data type descriptor is destroyed.

5) The data associated with *C* is destroyed.

6) The descriptor of *C* is removed from the descriptor of *T*.

7) The descriptor of *C* is destroyed.

8) The degree of *T* is reduced by 1 (one). The ordinal position of all columns having an ordinal position greater than the ordinal position of *C* is reduced by 1 (one).

Conformance Rules

1) Without Feature F033, "ALTER TABLE statement: DROP COLUMN clause", conforming SQL language shall not contain a <drop column definition>.

11.19 <add table constraint definition>

Function

Add a constraint to a table.

Format

```
<add table constraint definition> ::= ADD <table constraint definition>
```

Syntax Rules

1) If PRIMARY KEY is specified, then *T* shall not have any proper supertable.

Access Rules

None.

General Rules

1) Let *T* be the table identified by the <table name> in the containing <alter table statement>.
2) The table constraint descriptor for the <table constraint definition> is included in the table descriptor for *T*.
3) Let *TC* be the table constraint added to *T*. If *TC* causes some column *CN* to be known not nullable and no other constraint causes *CN* to be known not nullable, then the nullability characteristic of the column descriptor of *CN* is changed to known not nullable.

NOTE 273 — The nullability characteristic of a column is defined in Subclause 4.13, "Columns, fields, and attributes".

Conformance Rules

1) Without Feature F381, "Extended schema manipulation", conforming SQL language shall not contain an <add table constraint definition>.

11.20 <drop table constraint definition>

Function

Destroy a constraint on a table.

Format

```
<drop table constraint definition> ::= DROP CONSTRAINT <constraint name> <drop behavior>
```

Syntax Rules

1) Let *T* be the table identified by the <table name> in the containing <alter table statement>.

2) The <constraint name> shall identify a table constraint *TC* of *T*.

3) If *TC* is a unique constraint and *RC* is a referential constraint whose referenced table is *T* and whose referenced columns are the unique columns of *TC*, then *RC* is said to be *dependent on TC*.

4) If *QS* is a <query specification> that contains an implicit or explicit <group by clause> and that contains a column reference to a column *C* in its <select list> that is not contained in an aggregated argument of a <set function specification>, and if *G* is the set of grouping columns of *QS*, and if the table constraint *TC* is needed to conclude that $G \mapsto C$ is a known functional dependency in *QS*, then *QS* is said to be *dependent on TC*.

5) If *V* is a view that contains a <query specification> that is dependent on a table constraint *TC*, then *V* is said to be *dependent on TC*.

6) If *R* is an SQL routine whose <SQL routine body> contains a <query specification> that is dependent on a table constraint *TC*, then *R* is said to be *dependent on TC*.

7) If *C* is a constraint or assertion whose <search condition> contains a <query specification> that is dependent on a table constraint *TC*, then *C* is said to be *dependent on TC*.

8) If *T* is a trigger whose triggered action contains a <query specification> that is dependent on a table constraint *TC*, then *T* is said to be *dependent on TC*.

9) If *T* is a referenceable table with a derived self-referencing column, then:

   a) *TC* shall not be a unique constraint whose unique columns correspond to the attributes in the list of attributes of the derived representation of the reference type whose referenced type is the structured type of *T*.

   b) *TC* shall not be a unique constraint whose unique column is the self-referencing column of *T*.

10) If RESTRICT is specified, then:

   a) No table constraint shall be dependent on *TC*.

   b) The <constraint name> of *TC* shall not be generally contained in the SQL routine body of any routine descriptor.

c) No view shall be dependent on *TC*.

d) No SQL routine shall be dependent on *TC*.

e) No constraint or assertion shall be dependent on *TC*.

f) No trigger shall be dependent on *TC*.

NOTE 274 — If CASCADE is specified, then any such dependent object will be dropped by the effective execution of the General Rules of this Subclause.

Access Rules

None.

General Rules

1) Let *TCN2* be the <constraint name> of any table constraint that is dependent on *TC* and let *T2* be the <table name> of the table descriptor that includes *TCN2*. The following <alter table statement> is effectively executed without further Access Rule checking:

```
ALTER TABLE T2 DROP CONSTRAINT TCN2 CASCADE
```

2) Let *R* be any SQL-invoked routine whose routine descriptor contains the <constraint name> of *TC* in the SQL routine body. Let *SN* be the <specific name> of *R*. The following <drop routine statement> is effectively executed without further Access Rule checking:

```
DROP SPECIFIC ROUTINE SN CASCADE
```

3) Let *VN* be the table name of any view *V* that is dependent on *TC*. The following <drop view statement> is effectively executed for every *V*:

```
DROP VIEW VN CASCADE
```

4) Let *SN* be the specific name of any SQL routine *R* that is dependent on *TC*. The following <drop routine statement> is effectively executed for every *SR*:

```
DROP SPECIFIC ROUTINE SN CASCADE
```

5) Let *CN* be the constraint name of any constraint *C* that is dependent on *TC*. Let *TN* be the name of the table constrainted by *C*. The following <alter table statement> is effectively executed for every *C*:

```
ALTER TABLE TN DROP CONSTRAINT CN CASCADE
```

6) Let *AN* be the assertion name of any assertion *A* that is dependent on *TC*. The following <drop assertion statement> is effectively executed for every *A*:

```
DROP ASSERTION AN CASCADE
```

7) Let *TN* be the trigger name of any trigger *T* that is dependent on *TC*. The following <drop trigger statement> is effectively executed for every *T*:

```
DROP TRIGGER T CASCADE
```

8) The descriptor of *TC* is removed from the descriptor of *T*.

9) If *TC* causes some column *CN* to be known not nullable and no other constraint causes *CN* to be known not nullable, then the nullability characteristic of the column descriptor of *CN* is changed to possibly nullable.

   NOTE 275 — The nullability characteristic of a column is defined in Subclause 4.13, "Columns, fields, and attributes".

10) The descriptor of *TC* is destroyed.

Conformance Rules

1) Without Feature F381, "Extended schema manipulation", conforming SQL language shall not contain a <drop table constraint definition>.

11.21 <drop table statement>

Function

Destroy a table.

Format

```
<drop table statement> ::= DROP TABLE <table name> <drop behavior>
```

Syntax Rules

1) Let *T* be the table identified by the <table name> and let *TN* be that <table name>.

2) The schema identified by the explicit or implicit schema name of the <table name> shall include the descriptor of *T*.

3) *T* shall be a base table.

4) *T* shall not be a declared local temporary table.

5) An *impacted dereference operation* is any of the following:

   a) A <dereference operation> *DO*, where *T* is the scope of the reference type of the <reference value expression> immediately contained in *DO*.

   b) A <method reference> *MR*, where *T* is the scope of the reference type of the <value expression primary> immediately contained in *MR*.

   c) A <reference resolution> *RR*, where *T* is the scope of the reference type of the <reference value expression> immediately contained in *RR*.

6) If RESTRICT is specified, then *T* shall not have any proper subtables.

7) If RESTRICT is specified, then *T* shall not be referenced and no impacted dereference operation shall be contained in any of the following:

   a) The <query expression> of any view descriptor.

   b) The <search condition> of any constraint descriptor that is not a table check constraint descriptor included in the base table descriptor of *T*.

   c) The <search condition> of any assertion descriptor.

   d) The table descriptor of the referenced table of any referential constraint descriptor of any table other than *T*.

   e) The SQL routine body of any routine descriptor.

   f) The <triggered action> of any trigger descriptor.

   NOTE 276 — If CASCADE is specified, then such objects will be dropped by the execution of the <revoke statement> specified in the General Rules of this Subclause.

8) If RESTRICT is specified and *T* is a referenceable table, then *TN* shall not be the scope included in a reference type descriptor generally included in any of the following:

   a) The attribute descriptor of an attribute of a user-defined type.

   b) The column descriptor of a column of a table other than *T*.

   c) The descriptor of an SQL parameter or the result type included in the routine descriptor of any <SQL-invoked routine>.

   d) The descriptor of an SQL parameter or the result type included in a method specification descriptor included in the user-defined type descriptor of any user-defined type.

   e) The descriptor of any user-defined cast.

   NOTE 277 — A descriptor that "generally includes" another descriptor is defined in Subclause 6.3.4, "Descriptors", in ISO/IEC 9075-1.

9) Let *A* be the <authorization identifier> that owns the schema identified by the <schema name> of the table identified by *TN*.

Access Rules

1) The enabled authorization identifiers shall include *A*.

General Rules

1) Let *STN* be the <table name> of any direct subtable of *T*. The following <drop table statement> is effectively executed without further Access Rule checking:

```
DROP TABLE STN CASCADE
```

2) For every proper supertable of *T*, every superrow of every row of *T* is effectively deleted at the end of the SQL-statement, prior to the checking of any integrity constraints.

   NOTE 278 — This deletion creates neither a new trigger execution context nor the definition of a new state change in the current trigger execution context.

3) The following <revoke statement> is effectively executed with a current authorization identifier of "_SYSTEM" and without further Access Rule checking:

```
REVOKE ALL PRIVILEGES ON TN FROM A CASCADE
```

4) If *T* is a referenceable table, then:

   a) For every reference type descriptor *RTD* that includes a scope of *TN* and is generally included in any of the following:

      i) The attribute descriptor of an attribute of a user-defined type.

      ii) The column descriptor of a column of a table other than *T*.

      iii) The descriptor of an SQL parameter or the result type included in the routine descriptor of any <SQL-invoked routine>.

iv) The descriptor of an SQL parameter or the result type in a method specification descriptor included in the user-defined type descriptor of any user-defined type.

v) The descriptor of any user-defined cast.

the scope of *RTD* is made empty.

b) Let *SOD* be the descriptor of a schema object dependent on the table descriptor of *T*.

Case:

i) If *SOD* is a view descriptor, then let *SON* be the name of the view included in *SOD*. The following <drop view statement> is effectively executed without further Access Rule checking:

```
DROP VIEW SON CASCADE
```

ii) If *SOD* is an assertion descriptor, then let *SON* be the name of the assertion included in *SOD*. The following <drop assertion statement> is effectively executed without further Access Rule checking:

```
DROP ASSERTION SON CASCADE
```

iii) If *SOD* is a table constraint descriptor, then let *SON* be the name of the constraint included in *SOD*. Let *CTN* be the <table name> included in the table descriptor that includes *SOD*. The following <alter table statement> is effectively executed without further Access Rule checking:

```
ALTER TABLE CTN DROP CONSTRAINT SON CASCADE
```

iv) If *SOD* is a routine descriptor, then let *SON* be the specific name included in *SOD*. The following <drop routine statement> is effectively executed without further Access Rule checking:

```
DROP SPECIFIC ROUTINE SON CASCADE
```

v) If *SOD* is a trigger descriptor, then let *SON* be the trigger name included in *SOD*. The following <drop trigger statement> is effectively executed without further Access Rule checking:

```
DROP TRIGGER SON CASCADE
```

NOTE 279 — A descriptor that "depends on" another descriptor is defined in Subclause 6.3.4, "Descriptors", in ISO/IEC 9075-1.

5) For each direct supertable *DST* of *T*, the table name of *T* is removed from the list of table names of direct subtables of *DST* that is included in the table descriptor of *DST*.

6) The descriptor of *T* is destroyed.

Conformance Rules

1) Without Feature F032, "CASCADE drop behavior", conforming SQL language shall not contain a <drop table statement> that contains <drop behavior> that contains CASCADE.

11.22 <view definition>

Function

Define a viewed table.

Format

```
<view definition> ::=
    CREATE [ RECURSIVE ] VIEW <table name> <view specification>
    AS <query expression> [ WITH [ <levels clause> ] CHECK OPTION ]

<view specification> ::=
    <regular view specification>
  | <referenceable view specification>

<regular view specification> ::=
    [ <left paren> <view column list> <right paren> ]

<referenceable view specification> ::=
    OF <path-resolved user-defined type name> [ <subview clause> ]
    [ <view element list> ]

<subview clause> ::= UNDER <table name>

<view element list> ::=
    <left paren> <view element> [ { <comma> <view element> }... ] <right paren>

<view element> ::=
    <self-referencing column specification>
  | <view column option>

<view column option> ::= <column name> WITH OPTIONS <scope clause>

<levels clause> ::=
    CASCADED
  | LOCAL

<view column list> ::= <column name list>
```

Syntax Rules

1) The <query expression> shall have an element type that is a row type.

2) The <query expression> shall not contain a <target specification>.

3) The <view definition> shall not contain an <embedded variable specification> or a <dynamic parameter specification>.

4) If a <view definition> is contained in a <schema definition> and the <table name> contains a <schema name>, then that <schema name> shall be equivalent to the specified or implicit <schema name> of the containing <schema definition>.

5) The schema identified by the explicit or implicit schema name of the <table name> shall not include a table descriptor whose table name is <table name>.

6) No <table reference> generally contained in the <query expression> shall identify any declared local temporary table.

7) If a <table reference> generally contained in the <query expression> identifies the viewed table *VT* defined by <view definition> *VD*, then *VD* and *VT* are said to be *recursive*.

8) If *VD* is recursive, then:

   a) <view column list> shall be specified.

   b) RECURSIVE shall be specified.

   c) CHECK OPTION shall not be specified.

   d) <referenceable view specification> shall not be specified.

   e) *VD* is equivalent to

```
CREATE VIEW <table name> AS
    WITH RECURSIVE <table name> (<view column list>)
      AS (<query expression>)
    SELECT <view column list> FROM <table name>
```

9) The viewed table is updatable if and only if the <query expression> is updatable.

10) The viewed table is insertable-into if and only if the <query expression> is insertable-into.

11) If the <query expression> is a <query specification> that contains a <group by clause> or a <having clause> that is not contained in a <subquery>, then the viewed table defined by the <view definition> is a *grouped view*.

12) If any two columns in the table specified by the <query expression> have equivalent <column name>s, or if any column of that table has an implementation-dependent name, then a <view column list> shall be specified.

13) Equivalent <column name>s shall not be specified more than once in the <view column list>.

14) The number of <column name>s in the <view column list> shall be the same as the degree of the table specified by the <query expression>.

15) Every column in the table specified by <query expression> whose declared type is a character string type shall have a declared type collation.

16) If WITH CHECK OPTION is specified, then the viewed table shall be updatable.

17) If WITH CHECK OPTION is specified and <levels clause> is not specified, then a <levels clause> of CASCADED is implicit.

18) If WITH LOCAL CHECK OPTION is specified, then the <query expression> shall not generally contain a <query expression> *QE* or a <query specification> *QS* that is *possibly non-deterministic* unless *QE* or *QS* is generally contained in a viewed table that is a leaf underlying table of the <query expression>.

If WITH CASCADED CHECK OPTION is specified, then the <query expression> shall not generally contain a <query expression> or <query specification> that is *possibly non-deterministic*.

19) Let *V* be the view defined by the <view definition>. The underlying columns of every *i*-th column of *V* are the underlying columns of the *i*-th column of the <query expression> and the underlying columns of *V* are the underlying columns of the <query expression>.

20) <subview clause>, if present, identifies the *direct superview SV* of *V* and *V* is said to be a *direct subview* of *SV*. View *V1* is a *superview* of view *V2* if and only if one of the following is true:

   a) *V1* and *V2* are the same view.

   b) *V1* is a direct superview of *V2*.

   c) There exists a view *V3* such that *V1* is a direct superview of *V3* and *V3* is a superview of *V2*. If *V1* is a superview of *V2*, then *V2* is a subview of *V1*.

   If *V1* is a superview of *V2* and *V1* and *V2* are not the same view, then *V2* is a *proper subview* of *V1* and *V1* is a *proper superview* of *V2*.

   If *V2* is a direct subview of *V1*, then *V2* is a direct subtable of *V1*.

   NOTE 280 — It follows that the subviews of the superviews of *V* together constitute the subtable family of *V*, every implication of which applies.

21) If <referenceable view specification> is specified, then:

   a) *V* is a *referenceable view*.

   b) RECURSIVE shall not be specified.

   c) The <user-defined type name> simply contained in <path-resolved user-defined type name> shall identify a structured type *ST*.

   d) The subtable family of *V* shall not include a member, other than *V* itself, whose associated structured type is *ST*.

   e) If <subview clause> is not specified, then <self-referencing column specification> shall be specified.

   f) Let *QE* be the <query expression>.

   g) Let *n* be the number of attributes of *ST*. Let A_i, 1 (one) $\leq i \leq n$ be the attributes of *ST*.

   h) Let *RT* be the row type of *QE*.

   i) If <self-referencing column specification> is specified, then:

      i) Exactly one <self-referencing column specification> shall be specified.

      ii) <subview clause> shall not be specified.

      iii) SYSTEM GENERATED shall not be specified.

      iv) Let *RST* be the reference type REF(*ST*).

      Case:

         1) If USER GENERATED is specified, then:

A) *RST* shall have a user-defined representation.

B) Let *m* be 1 (one).

2) If DERIVED is specified, then:

A) *RST* shall have a derived representation.

B) Let *m* be 0 (zero).

j) If <subview clause> is specified, then:

i) The <table name> contained in the <subview clause> shall identify a referenceable table *SV* that is a view.

ii) *ST* shall be a direct subtype of the structured type of the direct supertable of *V*.

iii) The SQL-schema identified by the explicit or implicit <schema name> of the <table name> of *V* shall include the descriptor of *SV*.

iv) Let *MSV* be the maximum superview of the subtable family of *V*. Let *RMSV* be the reference type REF(*MSV*).

Case:

1) If *RMSV* has a user-defined representation, then let *m* be 1 (one).

2) Otherwise, *RMSV* has a derived representation. Let *m* be 0 (zero).

k) The degree of *RT* shall be $n+m$.

l) Let F_i, 1 (one) $\leq i \leq n$, be the fields of *RT*.

m) For *i* varying from 1 (one) to *n*:

i) The declared data type $DDTF_{i+m}$ of F_{i+m} shall be compatible with the declared data type $DDTA_i$ of A_i.

ii) The Syntax Rules of Subclause 9.16, "Data type identity", are applied with $DDTF_{i+m}$ and $DDTA_i$.

n) *QE* shall consist of a single <query specification> *QS*.

o) The <from clause> of *QS* shall simply contain a single <table reference> *TR*.

p) *TR* shall immediately contain a <table or query name>. Let *TQN* be the table identified by the <table or query name>. *TQN* is the *basis table* of *V*.

q) If *TQN* is a referenceable base table or a referenceable view, then *TR* shall simply contain ONLY.

r) *QS* shall not simply contain a <group by clause> or a <having clause>.

s) If <self-referencing column specification> is specified, then

Case:

i) If *RST* has a user-defined representation, then:

1) *TQN* shall have a candidate key consisting of a single column *RC*.
2) Let *SS* be the first <select sublist> in the <select list> of *QS*.
3) *SS* shall consist of a single <cast specification> *CS* whose leaf column is *RC*.

   NOTE 281 — "Leaf column of a <cast specification>" is defined in Subclause 6.12, "<cast specification>".
4) The declared type of F_1 shall be REF(*ST*).

ii) Otherwise, *RST* has a derived representation.

1) Let C_i, 1 (one) ≤ i ≤ n, be the columns of *V* that correspond to the attributes of the derived representation of *RST*.
2) *TQN* shall have a candidate key consisting of some subset of the underlying columns of C_i, 1 (one) ≤ i ≤ n.

t) If <subview clause> is specified, then *TQN* shall be a proper subtable or proper subview of the basis table of *SV*.

u) Let <view element list>, if specified, be *TEL1*.

v) Let *r* be the number of <view column option>s. For every <view column option> VCO_j, 1 (one) ≤ j ≤ r, <column name> shall be equivalent to the <attribute name> of some attribute of *ST*.

w) Distinct <view column option>s contained in *TEL1* shall specify distinct <column name>s.

x) Let CN_j, 1 (one) ≤ j ≤ r, be the <column name> contained in VCO_j and let SCL_j be the <scope clause> contained in VCO_j.

   i) CN_j shall be equivalent to some <attribute name> of *ST*, whose declared type is some reference type $CORT_j$.

   ii) The <table name> contained in SCL_j shall identify a referenceable table *SRT*.

   iii) *SRT* shall be based on the referenced type of $CORT_j$.

22) Let the *originally-defined columns* of *V* be the columns of the table defined by *QE*.

23) A column of *V* is called an *updatable column* of *V* if its underlying column is updatable.

24) If the <view definition> is contained in a <schema definition>, then let *A* be the explicit or implicit <authorization identifier> of the <schema definition>; otherwise, let *A* be the <authorization identifier> that owns the schema identified by the explicit or implicit <schema name> of the <table name>.

Access Rules

1) If a <view definition> is contained in an <SQL-client module definition>, then the enabled authorization identifiers shall include the <authorization identifier> that owns the schema identified by the implicit or explicit <schema name> of the <table name>.

2) If <referenceable view specification> is specified, then the applicable privileges for *A* shall include USAGE on *ST*.

3) If <subview clause> is specified, then

   Case:

   a) If <view definition> is contained, without an intervening <SQL routine spec> that specifies SQL SECURITY INVOKER, in an <SQL schema statement>, then the applicable privileges of the <authorization identifier> that owns the schema shall include UNDER for *SV*.

   b) Otherwise, the current privileges shall include UNDER for *SV*.

   NOTE 282 — "current privileges" and "applicable privileges" are defined in Subclause 12.3, "<privileges>".

General Rules

1) A view descriptor *VD* is created that describes *V*. *VD* includes:

   a) The <table name> *TN*.

   b) *QE*, as both the <query expression> of the descriptor and the original <query expression> of the descriptor.

   c) Case:

      i) If <regular view specification> is specified, then the column descriptors taken from the table specified by the <query expression>.

         Case:

         1) If a <view column list> is specified, then the <column name> of the *i*-th column of the view is the *i*-th <column name> in that <view column list>.

         2) Otherwise, the <column name>s of the view are the <column name>s of the table specified by the <query expression>.

      ii) Otherwise:

         1) A column descriptor in which:

            A) The name of the column is <self-referencing column name>.

            B) The data type descriptor is that generated by the <data type> "REF(*ST*) SCOPE(*TN*)".

            C) The nullability characteristic is *known not nullable*.

            D) The ordinal position is 1 (one).

            E) The column is indicated to be self-referencing.

         2) The column descriptor *ODCD* of each originally-defined column *ODC* of *V* in which:

            A) The <column name> included in *ODCD* is replaced by the <attribute name> of its corresponding attribute of *ST*.

B) If the declared type of the column is a reference type and some VCO_i contains the <attribute name> of *ST* that corresponds to the column, then the (possibly empty) scope contained in the reference type descriptor immediately included in the column descriptor is replaced by SCO_i.

3) If DERIVED is specified, then an indication that the self-referencing column is a derived self-referencing column.

4) If USER GENERATED is specified, then an indication that the self-referencing column is a user-generated self-referencing column.

d) An indication as to whether WITH CHECK OPTION was omitted, specified with LOCAL, or specified with CASCADED.

2) Let *VN* be the <table name>. Let *QE* be the <query expression> included in the view descriptor *VD* of the view identified by *VN*. Let *OQE* be the original <query expression> included in *VD*. If a <view column list> is specified, then let *VCL* be the <view column list> preceded by a <left paren> and followed by a <right paren>; otherwise, let *VCL* be the zero-length string.

Case:

a) If *VN* is immediately contained in some SQL-schema statement, then *VN* identifies the view descriptor *VD*.

b) If *VN* is immediately contained in a <table reference> that specifies ONLY, then *VN* references the same table as the <table reference>:

```
( OQE ) AS VN VCL
```

c) Otherwise, *VN* references the same table as the <table reference>:

```
( QE ) AS VN VCL
```

3) For *i* ranging from 1 (one) to the number of distinct leaf underlying tables of the <query expression> *QE* of *V*, let RT_i be the <table name>s of those tables. For every column *CV* of *V*:

a) Let $CRT_{i,j}$, for *j* ranging from 1 (one) to the number of columns of RT_i that are underlying columns of *CV*, be the <column name>s of those columns.

b) A set of privilege descriptors with the grantor for each set to the special grantor value "_SYSTEM" is created as follows:

i) For every column *CV* of *V*, a privilege descriptor is created that defines the privilege SELECT(*CV*) on *V* to *A*. That privilege is grantable if and only if all the following are true:

1) The applicable privileges for *A* include grantable SELECT privileges on all of the columns $CRT_{i,j}$.

2) The applicable privileges for *A* include grantable EXECUTE privileges on all SQL-invoked routines that are subject routines of <routine invocation>s contained in *QE*.

3) The applicable privileges for *A* include grantable SELECT privilege on every table *T1* and every method *M* such that there is a <method reference> *MR* contained in *QE* such that *T1*

is in the scope of the <value expression primary> of *MR* and *M* is the method identified by the <method name> of *MR*.

4) The applicable privileges for *A* include grantable SELECT privilege WITH HIERARCHY OPTION on at least one supertable of the scoped table of every <reference resolution> that is contained in *QE*.

ii) For every column *CV* of *V*, if the applicable privileges for *A* include REFERENCES($CRT_{i,j}$) for all *i* and for all *j*, and the applicable privileges for *A* include REFERENCES on some column of RT_i for all *i*, then a privilege descriptor is created that defines the privilege REFERENCES(*CV*) on *V* to *A*. That privilege is grantable if and only if all the following are true:

1) The applicable privileges for *A* include grantable REFERENCES privileges on all of the columns $CRT_{i,j}$.

2) The applicable privileges for *A* include grantable EXECUTE privileges on all SQL-invoked routines that are subject routines of <routine invocation>s contained in *QE*.

3) The applicable privileges for *A* include grantable SELECT privilege on every table *T1* and every method *M* such that there is a <method reference> *MR* contained in *QE* such that *T1* is in the scope of the <value expression primary> of *MR* and *M* is the method identified by the <method name> of *MR*.

4) The applicable privileges for *A* include grantable SELECT privilege WITH HIERARCHY OPTION on at least one supertable of the scoped table of every <reference resolution> that is contained in *QE*.

4) A privilege descriptor is created that defines the privilege SELECT on *V* to *A*. That privilege is grantable if and only if the applicable privileges for *A* include grantable SELECT privilege on every column of *V*. The grantor of that privilege descriptor is set to the special grantor value "_SYSTEM".

5) If the applicable privileges for *A* include REFERENCES privilege on every column of *V*, then a privilege descriptor is created that defines the privilege REFERENCES on *V* to *A*. That privilege is grantable if and only if the applicable privileges for *A* include grantable REFERENCES privilege on every column of *V*. The grantor of that privilege descriptor is set to the special grantor value "_SYSTEM".

6) If *V* is updatable, then a set of privilege descriptors with the grantor for each set to the special grantor value "_SYSTEM" is created as follows:

a) For each leaf underlying table *LUT* of *QE*, if *QE* is one-to-one with respect to *LUT*, and the applicable privileges for *A* include INSERT privilege on *LUT*, then a privilege descriptor is created that defines the INSERT privilege on *V*. That privilege is grantable if and only if the applicable privileges for *A* include grantable INSERT privilege on *LUT*.

b) For each leaf underlying table *LUT* of *QE*, if *QE* is one-to-one with respect to *LUT*, and the applicable privileges for *A* include UPDATE privilege on *LUT*, then a privilege descriptor is created that defines the UPDATE privilege on *V*. That privilege is grantable if and only if the applicable privileges for *A* include grantable UPDATE privilege on *LUT*.

c) For each leaf underlying table *LUT* of *QE*, if *QE* is one-to-one with respect to *LUT*, and the applicable privileges for *A* include DELETE privilege on *LUT*, then a privilege descriptor is created that defines the DELETE privilege on *V*. That privilege is grantable if and only if the applicable privileges for *A* include grantable DELETE privilege on *LUT*.

d) For each column *CV* of *V* that has a counterpart *CLUT* in *LUT*, if *QE* is one-to-one with respect to *LUT*, and the applicable privileges for *A* include INSERT(*CLUT*) privilege on *LUT*, then a privilege descriptor is created that defines the INSERT(*CV*) privilege on *V*. That privilege is grantable if and only if the applicable privileges for *A* include grantable INSERT(*CLUT*) privilege on *LUT*.

e) For each column *CV* of *V* that has a counterpart *CLUT* in *LUT*, if *QE* is one-to-one with respect to *LUT*, and the applicable privileges for *A* include UPDATE(*CLUT*) privilege on *LUT*, then a privilege descriptor is created that defines the UPDATE(*CV*) privilege on *V*. That privilege is grantable if and only if the applicable privileges for *A* include grantable UPDATE(*CLUT*) privilege on *LUT*.

7) If *V* is a referenceable view, then a set of privilege descriptors with the grantor for each set to the special grantor value "_SYSTEM" are created as follows:

a) A privilege descriptor is created that defines the SELECT privilege WITH HIERARCHY OPTION on *V* to *A*. That privilege is grantable.

b) For every method *M* of the structured type identified by <path-resolved user-defined type name>, a privilege descriptor is created that defines the privilege SELECT(*M*) on *V* to *A*. That privilege is grantable.

c) Case:

i) If <subview clause> is not specified, then a privilege descriptor is created that defines the UNDER privilege on *V* to *A*. That privilege is grantable.

ii) Otherwise, a privilege descriptor is created that defines the UNDER privilege on *V* to *A*. That privilege is grantable if and only if the applicable privileges for *A* include grantable UNDER privilege on the direct supertable of *V*.

8) If <subview clause> is specified, then let *ST* be the set of supertables of *V*. Let *PDS* be the set of privilege descriptors that define SELECT WITH HIERARCHY OPTION privilege on a table in *ST*.

9) For every privilege descriptor in *PDS*, with grantee *G* and grantor *A*,

Case:

a) If the privilege is grantable, then let *WGO* be "WITH GRANT OPTION".

b) Otherwise, let *WGO* be a zero-length string.

The following <grant statement> is effectively executed without further Access Rule checking:

```
GRANT SELECT ON V
TO G WGO FROM A
```

10) If <subview clause> is specified, then let *SVQE* be the <query expression> included in the view descriptor of *SV*.

a) The <query expression> included in the descriptor of *SV* is replaced by the following <query expression>:

```
( SVQE ) UNION ALL CORRESPONDING SELECT * FROM TN
```

b) The General Rules of this subclause are reevaluated for *SV* in the light of the new <query expression> in its descriptor.

11) If the character representation of the <query expression> cannot be represented in the Information Schema without truncation, then a completion condition is raised: *warning — query expression too long for information schema.*

NOTE 283 — The Information Schema is defined in ISO/IEC 9075-11.

Conformance Rules

1) Without Feature T131, "Recursive query", conforming SQL language shall not contain a <view definition> that immediately contains RECURSIVE.
2) Without Feature F751, "View CHECK enhancements", conforming SQL language shall not contain a <levels clause>.
3) Without Feature S043, "Enhanced reference types", conforming SQL language shall not contain a <referenceable view specification>.
4) Without Feature F751, "View CHECK enhancements", conforming SQL language shall not contain <view definition> that contains a <subquery> and contains CHECK OPTION.

11.23 <drop view statement>

Function

Destroy a view.

Format

```
<drop view statement> ::= DROP VIEW <table name> <drop behavior>
```

Syntax Rules

1) Let *V* be the table identified by the <table name> and let *VN* be that <table name>. The schema identified by the explicit or implicit schema name of *VN* shall include the descriptor of *V*.

2) *V* shall be a viewed table.

3) An *impacted dereference operation* is any of the following:

   a) A <dereference operation> *DO*, where *V* is the scope of the reference type of the <reference value expression> immediately contained in *DO*.

   b) A <method reference> *MR*, where *V* is the scope of the reference type of the <value expression primary> immediately contained in *MR*.

   c) A <reference resolution> *RR*, where *V* is the scope of the reference type of the <reference value expression> immediately contained in *RR*.

4) If RESTRICT is specified, then *V* shall not have any proper subtables.

5) If RESTRICT is specified, then *V* shall not be referenced and no impacted dereference operation shall be contained in any of the following:

   a) The <query expression> of the view descriptor of any view other than *V*.

   b) The <search condition> of any constraint descriptor or assertion descriptor.

   c) The <triggered action> of any trigger descriptor.

   d) The SQL routine body of any routine descriptor.

   NOTE 284 — If CASCADE is specified, then any such dependent object will be dropped by the execution of the <revoke statement> specified in the General Rules of this Subclause.

6) If RESTRICT is specified and *V* is a referenceable view, then *VN* shall not be the scope included in a reference type descriptor generally included in any of the following:

   a) The attribute descriptor of an attribute of a user-defined type.

   b) The column descriptor of a column of a table other than *V*.

   c) The descriptor of an SQL parameter or the result type included in the routine descriptor of any <SQL-invoked routine>.

  d) The descriptor of an SQL parameter or the result type included in a method specification descriptor included in the user-defined type descriptor of any user-defined type.

  e) The descriptor of any user-defined cast.

  NOTE 285 — A descriptor that "generally includes" another descriptor is defined in Subclause 6.3.4, "Descriptors", in ISO/IEC 9075-1.

7) Let *A* be the <authorization identifier> that owns the schema identified by the <schema name> of the table identified by *VN*.

Access Rules

1) The enabled authorization identifier shall include *A*.

General Rules

1) Let *SVN* be the <table name> of any direct subview of *V*. The following <drop view statement> is effectively executed without further Access Rule checking:

```
DROP VIEW SVN CASCADE
```

2) The following <revoke statement> is effectively executed with a current authorization identifier of "_SYSTEM" and without further Access Rule checking:

```
REVOKE ALL PRIVILEGES ON VN FROM A CASCADE
```

3) If *V* is a referenceable view, then:

  a) For every reference type descriptor *RTD* that includes a scope of *VN* and is generally included in any of the following:

    i) The attribute descriptor of an attribute of a user-defined type.

    ii) The column descriptor of a column of a table other than *V*.

    iii) The descriptor of an SQL parameter or the result type included in the routine descriptor of any <SQL-invoked routine>.

    iv) The descriptor of an SQL parameter or the result type included in a method specification descriptor included in the user-defined type descriptor of any user-defined type.

    v) The descriptor of any user-defined cast.

  the scope of *RTD* is made empty.

  b) Let *SOD* be the descriptor of a schema object dependent on the view descriptor of *V*.

  Case:

    i) If *SOD* is a view descriptor, then let *SON* be the name of the view included in *SOD*. The following <drop view statement> is effectively executed without further Access Rule checking:

```
    DROP VIEW SON CASCADE
```

ii) If *SOD* is an assertion descriptor, then let *SON* be the name of the assertion included in *SOD*. The following <drop assertion statement> is effectively executed without further Access Rule checking:

```
DROP ASSERTION SON CASCADE
```

iii) If *SOD* is a table constraint descriptor, then let *SON* be the name of the constraint included in *SOD*. Let *CTN* be the <table name> included in the table descriptor that includes *SOD*. The following <alter table statement> is effectively executed without further Access Rule checking:

```
ALTER TABLE CTN DROP CONSTRAINT SON CASCADE
```

iv) If *SOD* is a routine descriptor, then let *SON* be the specific name included in *SOD*. The following <drop routine statement> is effectively executed without further Access Rule checking:

```
DROP SPECIFIC ROUTINE SON CASCADE
```

v) If *SOD* is a trigger descriptor, then let *SON* be the trigger name included in *SOD*. The following <drop trigger statement> is effectively executed without further Access Rule checking:

```
DROP TRIGGER SON
```

NOTE 286 — A descriptor that "depends on" another descriptor is defined in Subclause 6.3.4, "Descriptors", in ISO/IEC 9075-1.

4) For each direct supertable *DST* of *V*, the table name of *V* is removed from the list of table names of direct subtables of *DST* that is included in the table descriptor of *DST*.

5) The descriptor of *V* is destroyed.

Conformance Rules

1) Without Feature F032, "CASCADE drop behavior", conforming SQL language shall not contain a <drop view statement> that contains a <drop behavior> that contains CASCADE.

11.24 <domain definition>

Function

Define a domain.

Format

```
<domain definition> ::=
    CREATE DOMAIN <domain name> [ AS ] <predefined type>
    [ <default clause> ]
    [ <domain constraint>... ]
    [ <collate clause> ]

<domain constraint> ::=
    [ <constraint name definition> ] <check constraint definition> [
    <constraint characteristics> ]
```

Syntax Rules

1) If a <domain definition> is contained in a <schema definition>, and if the <domain name> contains a <schema name>, then that <schema name> shall be equivalent to the specified or implicit <schema name> of the containing <schema definition>.

2) If <constraint name definition> is specified and its <constraint name> contains a <schema name>, then that <schema name> shall be equivalent to the explicit or implicit <schema name> of the <domain name> of the domain identified by the containing <domain definition> or <alter domain statement>.

3) The schema identified by the explicit or implicit schema name of the <domain name> shall not include a domain descriptor whose domain name is equivalent to <domain name> nor a user-defined type descriptor whose user-defined type name is equivalent to <domain name>.

4) If <predefined type> specifies a <character string type> and does not specify <character set specification>, then the character set name of the default character set of the schema identified by the implicit or explicit <schema name> of <domain name> is implicit.

5) <collate clause> shall not be both specified in <predefined type> and immediately contained in <domain definition>. If <collate clause> is immediately contained in <domain definition>, then it is equivalent to specifying an equivalent <collate clause> in <predefined type>.

6) Let *D1* be some domain. *D1* is *in usage by* a domain constraint *DC* if and only if the <search condition> of *DC* generally contains the <domain name> either of *D1* or of some domain *D2* such that *D1* is in usage by some domain constraint of *D2*. No domain shall be in usage by any of its own constraints.

7) If <collate clause> is specified, then <predefined type> shall be a character string type.

8) For every <domain constraint> specified:

   a) If <constraint characteristics> is not specified, then INITIALLY IMMEDIATE NOT DEFERRABLE is implicit.

b) If <constraint name definition> is not specified, then a <constraint name definition> that contains an implementation-dependent <constraint name> is implicit. The assigned <constraint name> shall obey the Syntax Rules of an explicit <constraint name>.

Access Rules

1) If a <domain definition> is contained in an <SQL-client module definition>, then the enabled authorization identifiers shall include the <authorization identifier> that owns the schema identified by the implicit or explicit <schema name> of the <domain name>.

General Rules

1) A <domain definition> defines a domain.

2) A data type descriptor is created that describes the declared type of the domain being created.

3) A domain descriptor is created that describes the domain being created. The domain descriptor contains the name of the domain, the data type descriptor of the declared type, the value of the <default clause> if the <domain definition> immediately contains <default clause>, and a domain constraint descriptor for every immediately contained <domain constraint>.

4) A privilege descriptor is created that defines the USAGE privilege on this domain to the <authorization identifier> *A* of the schema or SQL-client module in which the <domain definition> appears. This privilege is grantable if and only if the applicable privileges for *A* include a grantable REFERENCES privilege for each column reference included in the domain descriptor and a grantable USAGE privilege for each <domain name>, <collation name>, <character set name>, and <transliteration name> contained in the <search condition> of any domain constraint descriptor included in the domain descriptor. The grantor of the privilege descriptor is set to the special grantor value "_SYSTEM".

5) Let *DSC* be the <search condition> included in some domain constraint descriptor *DCD*. Let *D* be the name of the domain whose descriptor includes *DCD*. Let *T* be the name of some table whose descriptor includes some column descriptor with column name *C* whose domain name is *D*. Let *CSC* be a copy of *DSC* in which every instance of the <general value specification> VALUE is replaced by *C*.

6) The domain constraint specified by *DCD* for *C* is not satisfied if and only if

```
EXISTS ( SELECT * FROM T WHERE NOT ( CSC ) )
```

is *True*.

NOTE 287 — Subclause 10.8, "<constraint name definition> and <constraint characteristics>", specifies when a constraint is effectively checked.

Conformance Rules

1) Without Feature F251, "Domain support", conforming SQL language shall not contain a <domain definition>.

2) Without Feature F692, "Extended collation support", conforming SQL language shall not contain a <domain definition> that immediately contains a <collate clause>.

11.25 <alter domain statement>

Function

Change a domain and its definition.

Format

```
<alter domain statement> ::= ALTER DOMAIN <domain name> <alter domain action>

<alter domain action> ::=
    <set domain default clause>
  | <drop domain default clause>
  | <add domain constraint definition>
  | <drop domain constraint definition>
```

Syntax Rules

1) Let *D* be the domain identified by <domain name>. The schema identified by the explicit or implicit schema name of the <domain name> shall include the descriptor of *D*.

Access Rules

1) The enabled authorization identifiers shall include the <authorization identifier> that owns the schema identified by the implicit or explicit <schema name> of <domain name>.

General Rules

1) The domain descriptor of *D* is modified as specified by <alter domain action>.

   NOTE 288 — The changed domain descriptor of *D* is applicable to every column that is dependent on *D*.

Conformance Rules

1) Without Feature F711, "ALTER domain", conforming SQL language shall not contain an <alter domain statement>.

11.26 <set domain default clause>

Function

Set the default value in a domain.

Format

```
<set domain default clause> ::= SET <default clause>
```

Syntax Rules

None.

Access Rules

None.

General Rules

1) Let *D* be the domain identified by the <domain name> in the containing <alter domain statement>.
2) The default value specified by the <default clause> is placed in the domain descriptor of *D*.

Conformance Rules

1) Without Feature F711, "ALTER domain", conforming SQL language shall not contain a <set domain default clause>.

11.27 <drop domain default clause>

Function

Remove the default clause of a domain.

Format

```
<drop domain default clause> ::= DROP DEFAULT
```

Syntax Rules

1) Let *D* be the domain identified by the <domain name> in the containing <alter domain statement>.
2) The descriptor of *D* shall contain a default value.

Access Rules

None.

General Rules

1) Let *C* be the set of columns whose column descriptors contain the domain descriptor of *D*.
2) For every column belonging to *C*, if the column descriptor does not already contain a default value, then the default value from the domain descriptor of *D* is placed in that column descriptor.
3) The default value is removed from the domain descriptor of *D*.

Conformance Rules

1) Without Feature F711, "ALTER domain", conforming SQL language shall not contain a <drop domain default clause>.

11.28 <add domain constraint definition>

Function

Add a constraint to a domain.

Format

```
<add domain constraint definition> ::= ADD <domain constraint>
```

Syntax Rules

1) Let *D* be the domain identified by the <domain name> in the <alter domain statement>.

2) Let *D1* be some domain. *D1* is *in usage by* a domain constraint *DC* if and only if the <search condition> of *DC* generally contains the <domain name> either of *D1* or of some domain *D2* such that *D1* is in usage by some domain constraint of *D2*. No domain shall be in usage by any of its own constraints.

Access Rules

None.

General Rules

1) The constraint descriptor of the <domain constraint> is added to the domain descriptor of *D*.

2) If *DC* causes some column *CN* to be known not nullable and no other constraint causes *CN* to be known not nullable, then the nullability characteristic of the column descriptor of *CN* is changed to known not nullable.

   NOTE 289 — The nullability characteristic of a column is defined in Subclause 4.13, "Columns, fields, and attributes".

Conformance Rules

1) Without Feature F711, "ALTER domain", conforming SQL language shall not contain an <add domain constraint definition>.

11.29 <drop domain constraint definition>

Function

Destroy a constraint on a domain.

Format

```
<drop domain constraint definition> ::= DROP CONSTRAINT <constraint name>
```

Syntax Rules

1) Let *D* be the domain identified by the <domain name> *DN* in the containing <alter domain statement>.
2) Let *CD* be any column descriptor that includes *DN*, let *T* be the table described by the table descriptor that includes *CD*, and let *TN* be the <table name> of *T*.
3) Let *DC* be the descriptor of the constraint identified by <constraint name>.
4) *DC* shall be included in the domain descriptor of *D*.

Access Rules

None.

General Rules

1) The constraint descriptor *DC* is removed from the domain descriptor of *D*.
2) If *DC* causes some column *CN* to be known not nullable and no other constraint causes *CN* to be known not nullable, then the nullability characteristic of the column descriptor of *CN* is changed to possibly nullable.

   NOTE 290 — The nullability characteristic of a column is defined in Subclause 4.13, "Columns, fields, and attributes".
3) The descriptor of *DC* is destroyed.

Conformance Rules

1) Without Feature F711, "ALTER domain", conforming SQL language shall not contain a <drop domain constraint definition>.
2) Without Feature F491, "Constraint management", conforming SQL language shall not contain a <drop domain constraint definition>.

11.30 <drop domain statement>

Function

Destroy a domain.

Format

```
<drop domain statement> ::= DROP DOMAIN <domain name> <drop behavior>
```

Syntax Rules

1) Let *D* be the domain identified by <domain name> and let *DN* be that <domain name>. The schema identified by the explicit or implicit schema name of *DN* shall include the descriptor of *D*.

2) If RESTRICT is specified, then *D* shall not be referenced in any of the following:

   a) A column descriptor.

   b) The <query expression> of any view descriptor.

   c) The <search condition> of any constraint descriptor.

   d) The SQL routine body of any routine descriptor.

3) Let *UA* be the <authorization identifier> that owns the schema identified by the <schema name> of the domain identified by *DN*.

Access Rules

1) The enabled authorization identifiers shall include *UA*.

General Rules

1) Let *C* be any column descriptor that includes *DN*, let *T* be the table described by the table descriptor that includes *C*, and let *TN* be the table name of *T*. *C* is modified as follows:

   a) *DN* is removed from *C*. A copy of the data type descriptor of *D* is included in *C*.

   b) If *C* does not include a <default clause> and the domain descriptor of *D* includes a <default clause>, then a copy of the <default clause> of *D* is included in *C*.

   c) Let the *excluded constraint list* be the <constraint name> of each domain constraint descriptor included in the domain descriptor of *D* that does not occur in the implicit or explicit <constraint name list>.

   d) For every domain constraint descriptor included in the domain descriptor of *D* whose <constraint name> is not contained in the excluded constraint list:

i) Let *TCD* be a <table constraint definition> consisting of a <constraint name definition> whose <constraint name> is implementation-dependent, whose <table constraint> is derived from the <check constraint definition> of the domain constraint descriptor by replacing every instance of VALUE by the <column name> of *C*, and whose <constraint characteristics> are the <constraint characteristics> of the domain constraint descriptor.

ii) If the applicable privileges for *UA* include all of the privileges necessary for *UA* to successfully execute the <add table constraint definition>

```
ALTER TABLE TN ADD TCD
```

then the following <table constraint definition> is effectively executed with a current authorization identifier of *UA*:

```
ALTER TABLE TN ADD TCD
```

2) The following <revoke statement> is effectively executed with a current authorization identifier of "_SYSTEM" and without further Access Rule checking:

```
REVOKE USAGE ON DOMAIN DN FROM UA CASCADE
```

3) The descriptor of *D* is destroyed.

Conformance Rules

1) Without Feature F251, "Domain support", conforming SQL language shall not contain a <drop domain statement>.

11.31 <character set definition>

Function

Define a character set.

Format

```
<character set definition> ::=
    CREATE CHARACTER SET <character set name> [ AS ]
    <character set source> [ <collate clause> ]

<character set source> ::= GET <character set specification>
```

Syntax Rules

1) If a <character set definition> is contained in a <schema definition> and if the <character set name> immediately contained in the <character set definition> contains a <schema name>, then that <schema name> shall be equivalent to the specified or implicit <schema name> of the <schema definition>.

2) The schema identified by the explicit or implicit schema name of the <character set name> shall not include a character set descriptor whose character set name is <character set name>.

3) The character set *CS* identified by the <character set specification> contained in <character set source> shall have associated with it a privilege descriptor that was effectively defined by the <grant statement>

   ```
   GRANT USAGE ON CHARACTER SET CSN TO PUBLIC
   ```

   where *CSN* is a <character set name> that identifies *CS*.

4) If <collate clause> is specified, then the <collation name> contained in <collate clause> shall identify a collation descriptor *CD* included in the schema identified by the explicit or implicit <schema name> contained in the <collation name>. The collation shall be applicable to the character repertoire of the character set identified by <character set source>. The list of applicable character set names included in *CD* shall include one that identifies *CS*.

5) Let *A* be the <authorization identifier> that owns the schema identified by the implicit or explicit <schema name> of <character set name>.

Access Rules

1) If a <character set definition> is contained in an <SQL-client module definition>, then the enabled authorization identifiers shall include *A*.

2) The applicable privileges for *A* shall include USAGE on the character set identified by the <character set specification>.

General Rules

1) A <character set definition> defines a character set.
2) A character set descriptor is created for the defined character set.
3) The descriptor created for the character set being defined is identical to the descriptor for the character set identified by <character set specification>, except that the included character set name is <character set name> and, if <collate clause> is specified, then the included name of the default collation is the <collation name> contained in <collate clause>.
4) A privilege descriptor is created that defines the USAGE privilege on this character set to be the <authorization identifier> of the <schema definition> or <SQL-client module definition> in which the <character set definition> appears. The grantor of the privilege descriptor is set to the special grantor value "_SYSTEM". This privilege is grantable.

Conformance Rules

1) Without Feature F451, "Character set definition", conforming SQL language shall not contain a <character set definition>.

11.32 <drop character set statement>

Function

Destroy a character set.

Format

```
<drop character set statement> ::= DROP CHARACTER SET <character set name>
```

Syntax Rules

1) Let *C* be the character set identified by the <character set name> and let *CN* be the name of *C*.
2) The schema identified by the explicit or implicit schema name of *CN* shall include the descriptor of *C*.
3) The explicit or implicit <schema name> contained in *CN* shall not be equivalent to INFORMA-TION_SCHEMA.
4) *C* shall not be referenced in any of the following:
   a) The data type descriptor included in any column descriptor.
   b) The data type descriptor included in any domain descriptor.
   c) The data type descriptor generally included in any user-defined type descriptor.
   d) The data type descriptor included in any field descriptor.
   e) The <query expression> of any view descriptor.
   f) The <search condition> of any constraint descriptor.
   g) The collation descriptor of any collation.
   h) The transliteration descriptor of any transliteration.
   i) The SQL routine body, the <SQL parameter declaration>s, or the <returns data type> of any routine descriptor.
   j) The <SQL parameter declaration>s or <returns data type> of any method specification descriptor.
5) Let the containing schema be the schema identified by the <schema name> explicitly or implicitly contained in <character set name>.

Access Rules

1) Let *A* be the <authorization identifier> that owns the schema identified by the <schema name> of the character set identified by *C*. The enabled authorization identifiers shall include *A*.

General Rules

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

```
REVOKE USAGE ON CHARACTER SET CN FROM A CASCADE
```

2) The descriptor of *C* is destroyed.

Conformance Rules

1) Without Feature F451, "Character set definition", conforming SQL language shall not contain a <drop character set statement>.

11.33 <collation definition>

Function

Define a collation.

Format

```
<collation definition> ::=
    CREATE COLLATION <collation name> FOR <character set specification>
    FROM <existing collation name> [ <pad characteristic> ]

<existing collation name> ::= <collation name>

<pad characteristic> ::=
    NO PAD
  | PAD SPACE
```

Syntax Rules

1) Let *A* be the <authorization identifier> that owns the schema identified by the implicit or explicit <schema name> of the <collation name>.

2) If a <collation definition> is contained in a <schema definition> and if the <collation name> immediately contained in the <collation definition> contains a <schema name>, then that <schema name> shall be equivalent to the specified or implicit <schema name> of the <schema definition>.

3) The schema identified by the explicit or implicit schema name of the <collation name> *CN* immediately contained in <collation definition> shall not include a collation descriptor whose collation name is *CN*.

4) The schema identified by the explicit or implicit schema name of the <collation name> *ECN* immediately contained in <existing collation name> shall include a collation descriptor whose collation name is *ECN*.

5) The collation identified by *ECN* shall be a collation whose descriptor includes a character repertoire name that is equivalent to that included in the descriptor of the character set identified by <character set specification>.

6) If <pad characteristic> is not specified, then the <pad characteristic> of the collation identified by *ECN* is implicit.

7) If NO PAD is specified, then the collation is said to have the NO PAD characteristic. If PAD SPACE is specified, then the collation is said to have the PAD SPACE characteristic.

Access Rules

1) If a <collation definition> is contained in an <SQL-client module definition>, then the enabled authorization identifiers shall include *A*.

2) The applicable privileges for *A* shall include USAGE on *ECN*.

General Rules

1) A <collation definition> defines a collation.
2) A privilege descriptor is created that defines the USAGE privilege on this collation for *A*. The grantor of the privilege descriptor is set to the special grantor value "_SYSTEM".
3) This privilege descriptor is grantable if and only if the USAGE privilege for *A* on the collation identified by *ECN* is grantable.
4) A collation descriptor is created for the defined collation.
5) The collation descriptor *CD* created is identical to the collation descriptor for *ECN*, except that the collation name included in *CD* is *CN* and, if <pad characteristic> is specified, then the pad characteristic included in *CD* is <pad characteristic>.

Conformance Rules

1) Without Feature F690, "Collation support", conforming SQL language shall not contain a <collation definition>.

11.34 <drop collation statement>

Function

Destroy a collation.

Format

```
<drop collation statement> ::= DROP COLLATION <collation name> <drop behavior>
```

Syntax Rules

1) Let *C* be the collation identified by the <collation name> and let *CN* be the name of *C*.
2) The schema identified by the explicit or implicit schema name of *CN* shall include the descriptor of *C*.
3) The explicit or implicit <schema name> contained in *CN* shall not be equivalent to INFORMATION_SCHEMA.
4) If RESTRICT is specified, then *C* shall not be referenced in any of the following:
   a) Any character set descriptor.
   b) The triggered action of any trigger descriptor.
   c) The <query expression> of any view descriptor.
   d) The <search condition> of any constraint descriptor.
   e) The SQL routine body, the <SQL parameter declaration>s, or the <returns data type> of any routine descriptor.
   f) The <SQL parameter declaration>s or the <returns data type> of any method specification descriptor.
5) Let *A* be the <authorization identifier> that owns the schema identified by the <schema name> of the collation identified by *C*.
6) Let the containing schema be the schema identified by the <schema name> explicitly or implicitly contained in <collation name>.

Access Rules

1) The enabled authorization identifiers shall include *A*.

General Rules

1) For every character set descriptor *CSD* that includes *CN*, *CSD* is modified such that it does not include *CN*. If *CSD* does not include any collation name, then *CSD* is modified to indicate that it utilizes the default collation for its character repertoire.

2) For every data type descriptor *DD* that includes *CN*, *DD* is modified such that it includes the collation name of the character set collation of the character set of *DD*.

   NOTE 291 — This causes the column, domain, attribute, or field described by *DD* to revert to the default collation for its character set.

3) The following <revoke statement> is effectively executed with a current authorization identifier of "_SYSTEM" and without further Access Rule checking:

```
REVOKE USAGE ON COLLATION CN FROM A CASCADE
```

4) The descriptor of *C* is destroyed.

Conformance Rules

1) Without Feature F690, "Collation support", conforming SQL language shall not contain a <drop collation statement>.

11.35 <transliteration definition>

Function

Define a character transliteration.

Format

```
<transliteration definition> ::=
    CREATE TRANSLATION <transliteration name> FOR <source character set specification>
    TO <target character set specification> FROM <transliteration source>

<source character set specification> ::= <character set specification>

<target character set specification> ::= <character set specification>

<transliteration source> ::=
    <existing transliteration name>
  | <transliteration routine>

<existing transliteration name> ::= <transliteration name>

<transliteration routine> ::= <specific routine designator>
```

Syntax Rules

1) If a <transliteration definition> is contained in a <schema definition> and if the <transliteration name> immediately contained in the <transliteration definition> contains a <schema name>, then that <schema name> shall be equivalent to the specified or implicit <schema name> of the <schema definition>.

2) The schema identified by the explicit or implicit schema name of the <transliteration name> *TN* immediately contained in <transliteration definition> shall not include a transliteration descriptor whose transliteration name is *TN*.

3) The schema identified by the explicit or implicit schema name of the <character set name> *SCSN* contained in the <character set specification> contained in <source character set specification> shall include a character set descriptor whose character set name is *SCSN*.

4) The schema identified by the explicit or implicit schema name of the <character set name> *TCSN* contained in the <character set specification> contained in <source character set specification> shall include a character set descriptor whose character set name is *TCSN*.

5) If <existing transliteration name> is specified, then:

   a) The schema identified by the explicit or implicit schema name of the <transliteration name> *TN* contained in <transliteration source> shall include a transliteration descriptor whose transliteration name is *TN*.

   b) The character set identified by *SCSN* shall have the same character repertoire and character encoding form as the source character set of the transliteration identified by *TN*.

   c) The character set identified by *TCSN* shall have the same character repertoire and character encoding form as the target character set of the transliteration identified by *TN*.

6) If <transliteration routine> is specified, then:

   a) The schema identified by the explicit or implicit schema name of the <specific routine designator> *SRD* contained in <transliteration routine> shall include a routine descriptor that identifies a routine having a <specific routine designator> *SRD*.

   b) The routine identified by *SRD* shall be an SQL-invoked function that has one parameter whose data type is character string and whose character set is the character set specified by *SCSN*; the <returns type> of the routine shall be character string whose character set is the character set specified by *TCSN*.

Access Rules

1) Let *A* be the <authorization identifier> that owns the schema identified by the implicit or explicit <schema name> of <transliteration name>. If a <transliteration definition> is contained in an <SQL-client module definition>, then the enabled authorization identifiers shall include *A*.

2) If <transliteration source> is specified, then the applicable privileges for *A* shall include USAGE on the transliteration identified by *TN*.

3) If <transliteration routine> is specified, then the applicable privileges for *A* shall include EXECUTE on the routine identified by *RN*.

General Rules

1) A <transliteration definition> defines a transliteration.

2) If <transliteration source> contains <existing transliteration name>, then let *SRDN* be the specific name included in the transliteration descriptor whose transliteration name is *TN*; otherwise, let *SRDN* be the specific name of the SQL-invoked routine identified by <transliteration routine>.

3) A transliteration descriptor is created that includes:

   a) The name of the transliteration *TN*.

   b) The name of the character set *SCSN* from which it translates.

   c) The name of the character set *TCSN* to which it translates.

   d) *SRDN*, the specific name of the SQL-invoked routine that performs the transliteration.

4) A privilege descriptor *PD* is created that defines the USAGE privilege on this transliteration to the <authorization identifier> of the <schema definition> or <SQL-client module definition> in which the <transliteration definition> appears. The grantor of the privilege descriptor is set to the special grantor value "_SYSTEM".

5) *PD* is grantable if and only if the USAGE privilege for the <authorization identifier> of the <schema definition> or <SQL-client module definition> in which the <transliteration definition> appears is also grantable on every character set identified by a <character set name> contained in the <transliteration definition>.

Conformance Rules

1) Without Feature F695, "Translation support", conforming SQL language shall not contain a <transliteration definition>.

11.36 <drop transliteration statement>

Function

Destroy a character transliteration.

Format

```
<drop transliteration statement> ::= DROP TRANSLATION <transliteration name>
```

Syntax Rules

1) Let *T* be the transliteration identified by the <transliteration name> and let *TN* be the name of *T*.
2) Let *A* be the <authorization identifier> that owns the schema identified by the <schema name> of the transliteration identified by *T*.
3) The schema identified by the explicit or implicit schema name of *TN* shall include the descriptor of *T*.
4) *T* shall not be referenced in any of the following:
   a) The triggered action of any trigger descriptor.
   b) The <query expression> of any view descriptor.
   c) The <search condition> of any constraint descriptor.
   d) The collation descriptor of any collation.
   e) The transliteration descriptor of any translation.
   f) The SQL routine body of any routine descriptor.

Access Rules

1) The enabled authorization identifiers shall include *A*.

General Rules

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

   ```
   REVOKE USAGE ON TRANSLATION TN FROM A CASCADE
   ```

2) The descriptor of *T* is destroyed.

Conformance Rules

1) Without Feature F695, "Translation support", conforming SQL language shall not contain a <drop transliteration statement>.

11.37 <assertion definition>

Function

Specify an integrity constraint.

Format

```
<assertion definition> ::=
    CREATE ASSERTION <constraint name>
    CHECK <left paren> <search condition> <right paren>
    [ <constraint characteristics> ]
```

Syntax Rules

1) If an <assertion definition> is contained in a <schema definition> and if the <constraint name> contains a <schema name>, then that <schema name> shall be equivalent to the explicit or implicit <schema name> of the containing <schema definition>.

2) The schema identified by the explicit or implicit schema name of the <constraint name> shall not include a constraint descriptor whose constraint name is <constraint name>.

3) If <constraint characteristics> is not specified, then INITIALLY IMMEDIATE NOT DEFERRABLE is implicit.

4) The <search condition> shall not contain a <host parameter name>, an <SQL parameter name>, an <embedded variable specification> or a <dynamic parameter specification>.

   NOTE 292 — <SQL parameter name> is excluded because of the scoping rules for <SQL parameter name>.

5) No <query expression> in the <search condition> shall reference a temporary table.

6) The <search condition> shall simply contain a <boolean value expression> that is retrospectively deterministic.

   NOTE 293 — "retrospectively deterministic" is defined in Subclause 6.34, "<boolean value expression>".

7) The <search condition> shall not generally contain a <routine invocation> whose subject routine is an SQL-invoked routine that possibly modifies SQL-data.

8) The <qualified identifier> of <constraint name> shall not be equivalent to the <qualified identifier> of the <constraint name> of any other constraint defined in the same schema.

Access Rules

1) Let *A* be the <authorization identifier> that owns the schema identified by the <schema name> of the <assertion definition>. If an <assertion definition> is contained in an <SQL-client module definition>, then the enabled authorization identifier shall include *A*.

General Rules

1) An <assertion definition> defines an assertion. An assertion is a constraint.

   NOTE 294 — Subclause 10.8, "<constraint name definition> and <constraint characteristics>", specifies when a constraint is effectively checked.

2) Let *SC* be the <search condition> simply contained in the <assertion definition>.

3) The assertion is not satisfied if and only if the result of evaluating *SC* is *False*.

4) An assertion descriptor is created that describes the assertion being defined. The name included in the assertion descriptor is <constraint name>.

   The assertion descriptor includes an indication of whether the constraint is deferrable or not deferrable and whether the initial constraint mode is *deferred* or *immediate*.

   The assertion descriptor includes *SC*.

5) If the character representation of *SC* cannot be represented in the Information Schema without truncation, then a completion condition is raised: *warning — search condition too long for information schema*.

   NOTE 295 — The Information Schema is defined in ISO/IEC 9075-11.

6) If *SC* causes some column *CN* be to known not nullable and no other constraint causes *CN* to be known not nullable, then the nullability characteristic of *CN* is changed to known not nullable.

   NOTE 296 — The nullability characteristic of a column is defined in Subclause 4.13, "Columns, fields, and attributes".

Conformance Rules

1) Without Feature F521, "Assertions", conforming SQL language shall not contain an <assertion definition>.

2) Without Feature F672, "Retrospective check constraints", conforming SQL language shall not contain an <assertion definition> that generally contains CURRENT_DATE, CURRENT_TIMESTAMP, or LOCALTIMESTAMP.

11.38 <drop assertion statement>

Function

Destroy an assertion.

Format

```
<drop assertion statement> ::= DROP ASSERTION <constraint name> [ <drop behavior> ]
```

Syntax Rules

1) Let *A* be the assertion identified by <constraint name> and let *AN* be the name of *A*.

2) The schema identified by the explicit or implicit schema name of *AN* shall include the descriptor of *A*.

3) If <drop behavior> is not specified, then RESTRICT is implicit.

4) If RESTRICT is specified or implied, then *AN* shall not be referenced in the SQL routine body of any routine descriptor.

5) If *QS* is a <query specification> that contains a column reference to a column *C* in its <select list> that is not contained in a <set function specification>, and if *G* is the set of columns defined by the <grouping column reference list> of *QS*, and if the assertion A is needed to conclude that $G \mapsto C$ is a known functional dependency in QS, then *QS* is said to be *dependent on A*.

6) If *V* is a view that contains a <query specification> that is dependent on *A*, then *V* is said to be *dependent on A*.

7) If *R* is an SQL routine whose <SQL routine body> contains a <query specification> that is dependent on *A*, then *R* is said to be *dependent on A*.

8) If *C* is a constraint or assertion whose <search condition> contains a <query specification> that is dependent on *A*, then *C* is said to be *dependent on A*.

9) If *T* is a trigger whose triggered action contains a <query specification> that is dependent on *A*, then *T* is said to be *dependent on A*.

10) If RESTRICT is specified or implicit, or <drop behavior> is not specified, then:

   a) No table constraint shall be dependent on *A*.

   b) No view shall be dependent on *TC*.

   c) No SQL routine shall be dependent on *TC*.

   d) No constraint or assertion shall be dependent on *TC*.

   e) No trigger shall be dependent on *TC*.

   NOTE 297 — If CASCADE is specified, then any such dependent object will be dropped by the execution of the <revoke statement> specified in the General Rules of this Subclause.

Access Rules

1) The enabled authorization identifiers shall include the <authorization identifier> that owns the schema identified by the <schema name> of the assertion identified by *AN*.

General Rules

1) Let *R* be any SQL-invoked routine whose routine descriptor contains the <constraint name> of *A* in the SQL routine body. Let *SN* be the <specific name> of *R*. The following <drop routine statement> is effectively executed without further Access Rule checking:

```
DROP SPECIFIC ROUTINE SN CASCADE
```

2) Let *VN* be the table name of any view *V* that is dependent on *A*. The following <drop view statement> is effectively executed for every *V*:

```
DROP VIEW VN CASCADE
```

3) Let *SN* be the specific name of any SQL routine *SR* that is dependent on *A*, or that contains a reference to *A*. The following <drop routine statement> is effectively executed for every *SR*:

```
DROP SPECIFIC ROUTINE SN CASCADE
```

4) Let *CN* be the constraint name of any constraint *C* that is dependent on *A*. Let *TN* be the name of the table constrained by *C*. The following <alter table statement> is effectively executed for every *C*:

```
ALTER TABLE TN DROP CONSTRAINT CN CASCADE
```

5) Let *AN2* be the assertion name of any assertion *A* that is dependent on *A*. The following <drop assertion statement> is effectively executed for every *A*:

```
DROP ASSERTION AN2 CASCADE
```

6) Let *TN* be the trigger name of any trigger *T* that is dependent on *A*. The following <drop trigger statement> is effectively executed for every *T*:

```
DROP TRIGGER TN
```

7) Let *SC* be the <search condition> included in the descriptor of *A*. If *SC* causes some column *CN* be to known not nullable and no other constraint causes *CN* to be known not nullable, then the nullability characteristic of *CN* is changed to possibly nullable.

   NOTE 298 — The nullability characteristic of a column is defined in Subclause 4.13, "Columns, fields, and attributes".

8) The descriptor of *A* is destroyed.

Conformance Rules

1) Without Feature F521, "Assertions", conforming SQL language shall not contain a <drop assertion statement>.

11.39 <trigger definition>

Function

Define triggered SQL-statements.

Format

```
<trigger definition> ::=
    CREATE TRIGGER <trigger name> <trigger action time> <trigger event>
    ON <table name> [ REFERENCING <transition table or variable list> ]
    <triggered action>

<trigger action time> ::=
    BEFORE
  | AFTER

<trigger event> ::=
    INSERT
  | DELETE
  | UPDATE [ OF <trigger column list> ]

<trigger column list> ::= <column name list>

<triggered action> ::=
    [ FOR EACH { ROW | STATEMENT } ]
    [ WHEN <left paren> <search condition> <right paren> ]
    <triggered SQL statement>

<triggered SQL statement> ::=
    <SQL procedure statement>
  | BEGIN ATOMIC { <SQL procedure statement> <semicolon> }... END

<transition table or variable list> ::= <transition table or variable>...

<transition table or variable> ::=
    OLD [ ROW ] [ AS ] <old transition variable name>
  | NEW [ ROW ] [ AS ] <new transition variable name>
  | OLD TABLE [ AS ] <old transition table name>
  | NEW TABLE [ AS ] <new transition table name>

<old transition table name> ::= <transition table name>

<new transition table name> ::= <transition table name>

<transition table name> ::= <identifier>

<old transition variable name> ::= <correlation name>

<new transition variable name> ::= <correlation name>
```

Syntax Rules

1) Case:

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

   b) If a <trigger definition> is contained in an <SQL-client module definition> and if the <trigger name> contains a <schema name>, then that <schema name> shall be equivalent to the specified or implicit <schema name> of the <SQL-client module definition>.

2) Let *TN* be the <table name> of a <trigger definition>. The table *T* identified by *TN* is the *subject table* of the <trigger definition>.

3) The schema identified by the explicit or implicit <schema name> of *TN* shall include the descriptor of *T*.

4) The schema identified by the explicit or implicit <schema name> of a <trigger name> *TRN* shall not include a trigger descriptor whose trigger name is *TRN*.

5) *T* shall be a base table that is not a declared local temporary table.

6) If a <trigger column list> is specified, then:

   a) No <column name> shall appear more than once in the <trigger column list>.

   b) The <column name>s of the <trigger column list> shall identify columns of *T*.

7) If REFERENCING is specified, then:

   a) Let *OR*, *OT*, *NR*, and *NT* be the <old transition variable name>, <old transition table name>, <new transition variable name>, and <new transition table name>, respectively.

   b) OLD or OLD ROW, NEW or NEW ROW, OLD TABLE, and NEW TABLE shall be specified at most once each within the <transition table or variable list>.

   c) Case:

      i) If <trigger event> specifies INSERT, then neither OLD ROW nor OLD TABLE shall be specified.

      ii) If <trigger event> specifies DELETE, then neither NEW ROW nor NEW TABLE shall be specified.

   d) No two of *OR*, *OT*, *NR*, and *NT* shall be equivalent.

   e) Both *OR* and *NR* are range variables.

      NOTE 299 — "range variable" is defined in Subclause 4.14.6, "Operations involving tables".

   f) The scope of *OR*, *OT*, *NR*, and *NT* is the <triggered action>, excluding any <SQL schema statement>s that are contained in the <triggered action>.

8) If neither FOR EACH ROW nor FOR EACH STATEMENT is specified, then FOR EACH STATEMENT is implicit.

9) If *OR* or *NR* is specified, then FOR EACH ROW shall be specified.

10) The <triggered action> shall not contain an <SQL parameter reference>, a <host parameter name>, a <dynamic parameter specification>, or an <embedded variable name>.

11) It is implementation-defined whether the <triggered SQL statement> shall not generally contain an <SQL transaction statement>, an <SQL connection statement>, an <SQL schema statement>, an <SQL dynamic statement>, or an <SQL session statement>.

12) If BEFORE is specified, then:

   a) It is implementation-defined whether the <triggered action> shall not generally contain an <SQL data change statement> or a <routine invocation> whose subject routine is an SQL-invoked routine that possibly modifies SQL-data.

   b) Neither OLD TABLE nor NEW TABLE shall be specified.

   c) The <triggered action> shall not contain a <field reference> that references a field in the new transition variable corresponding to a generated column of *T*.

Access Rules

1) Let *A* be the <authorization identifier> that owns the schema identified by the implicit or explicit <schema name> of the <trigger name> of the <trigger definition>. If a <trigger definition> is contained in an <SQL-client module definition>, then the enabled authorization identifiers shall include *A*.

2) The applicable privileges for *A* for *T* shall include TRIGGER.

3) If the <triggered action> *TA* of a <trigger definition> contains an <old transition table name> *OTTN*, an <old transition variable name> *OTVN*, a <new transition table name> *NTTN*, or a <new transition variable name> *NTVN*, and *TA* contains *OTTN*, *OTVN*, or *NTTN*, or if *TA* contains *NTVN*, then the applicable privileges for *TA* shall include SELECT.

General Rules

1) A <trigger definition> defines a trigger.

2) *OT* identifies the old transition table. *NT* identifies the new transition table. *OR* identifies the old transition variable. *NR* identifies the new transition variable.

   NOTE 300 — "old transition table", "new transition table", "old transition variable", and "new transition variable" are defined in Subclause 4.38.1, "General description of triggers".

3) The transition table identified by *OT* is the table associated with *OR*. The transition table identified by *NT* is the table associated with *NR*.

4) If the character representation of the <triggered SQL statement> cannot be represented in the Information Schema without truncation, then a completion condition is raised: *warning — statement too long for information schema*.

   NOTE 301 — The Information Schema is defined in ISO/IEC 9075-11.

5) A trigger descriptor is created for <trigger definition>s as follows:

   a) The trigger name included in the trigger descriptor is <trigger name>.

b) The subject table included in the trigger descriptor is <table name>.

c) The trigger action time included in the trigger descriptor is <trigger action time>.

d) If FOR EACH STATEMENT is specified or implicit, then an indication that the trigger is a statement-level trigger; otherwise, an indication that the trigger is a row-level trigger.

e) The trigger event included in the trigger descriptor is <trigger event>.

f) Any <old transition variable name>, <new transition variable name>, <old transition table name>, or <new transition table name> specified in the <trigger definition> is included in the trigger descriptor as the old transition variable name, new transition variable name, old transition table name, or new transition table name, respectively.

g) The trigger action included in the trigger descriptor is the specified <triggered action>.

h) If a <trigger column list> *TCL* is specified, then *TCL* is the trigger column list included in the trigger descriptor; otherwise, that trigger column list is empty.

i) The *triggered action column set* included in the trigger descriptor is the set of all distinct, fully qualified names of columns contained in the <triggered action>.

j) The timestamp of creation included in the trigger descriptor is the timestamp of creation of the trigger.

Conformance Rules

1) Without Feature T211, "Basic trigger capability", conforming SQL language shall not contain a <trigger definition>.

2) Without Feature T212, "Enhanced trigger capability", in conforming SQL language, a <triggered action> shall contain FOR EACH ROW.

11.40 <drop trigger statement>

Function

Destroy a trigger.

Format

```
<drop trigger statement> ::= DROP TRIGGER <trigger name>
```

Syntax Rules

None.

Access Rules

1) Let *TR* be the trigger identified by the <trigger name>. The enabled authorization identifiers shall include the <authorization identifier> that owns the schema identified by the <schema name> of *TR*.

General Rules

1) The descriptor of *TR* is destroyed.

Conformance Rules

1) Without Feature T211, "Basic trigger capability", conforming SQL language shall not contain a <drop trigger statement>.

11.41 <user-defined type definition>

Function

Define a user-defined type.

Format

```
<user-defined type definition> ::= CREATE TYPE <user-defined type body>

<user-defined type body> ::=
    <schema-resolved user-defined type name>
    [ <subtype clause> ]
    [ AS <representation> ]
    [ <user-defined type option list> ]
    [ <method specification list> ]

<user-defined type option list> ::=
    <user-defined type option> [ <user-defined type option>... ]

<user-defined type option> ::=
    <instantiable clause>
  | <finality>
  | <reference type specification>
  | <cast to ref>
  | <cast to type>
  | <cast to distinct>
  | <cast to source>

<subtype clause> ::= UNDER <supertype name>

<supertype name> ::= <path-resolved user-defined type name>

<representation> ::=
    <predefined type>
  | <member list>

<member list> ::= <left paren> <member> [ { <comma> <member> }... ] <right paren>

<member> ::= <attribute definition>

<instantiable clause> ::=
    INSTANTIABLE
  | NOT INSTANTIABLE

<finality> ::=
    FINAL
  | NOT FINAL

<reference type specification> ::=
    <user-defined representation>
  | <derived representation>
  | <system-generated representation>

<user-defined representation> ::= REF USING <predefined type>
```

```
<derived representation> ::= REF FROM <list of attributes>

<system-generated representation> ::= REF IS SYSTEM GENERATED

<cast to ref> ::=
    CAST <left paren> SOURCE AS REF <right paren> WITH <cast to ref identifier>

<cast to ref identifier> ::= <identifier>

<cast to type> ::=
    CAST <left paren> REF AS SOURCE <right paren> WITH <cast to type identifier>

<cast to type identifier> ::= <identifier>

<list of attributes> ::=
    <left paren> <attribute name> [ { <comma> <attribute name> }... ] <right paren>

<cast to distinct> ::=
    CAST <left paren> SOURCE AS DISTINCT <right paren>
    WITH <cast to distinct identifier>

<cast to distinct identifier> ::= <identifier>

<cast to source> ::=
    CAST <left paren> DISTINCT AS SOURCE <right paren>
    WITH <cast to source identifier>

<cast to source identifier> ::= <identifier>

<method specification list> ::=
    <method specification> [ { <comma> <method specification> }... ]

<method specification> ::=
    <original method specification>
  | <overriding method specification>

<original method specification> ::=
    <partial method specification> [ SELF AS RESULT ] [ SELF AS LOCATOR ]
    [ <method characteristics> ]

<overriding method specification> ::= OVERRIDING <partial method specification>

<partial method specification> ::=
    [ INSTANCE | STATIC | CONSTRUCTOR ]
    METHOD <method name> <SQL parameter declaration list>
    <returns clause>
    [ SPECIFIC <specific method name> ]

<specific method name> ::= [ <schema name> <period> ]<qualified identifier>

<method characteristics> ::= <method characteristic>...

<method characteristic> ::=
    <language clause>
  | <parameter style clause>
  | <deterministic characteristic>
  | <SQL-data access indication>
  | <null-call clause>
```

Syntax Rules

1) Let *UDTD* be the <user-defined type definition>, let *UDTB* be the <user-defined type body> immediately contained in *UDTD*, let *UDTN* be the <schema-resolved user-defined type name> immediately contained in *UDTB*, let *SN* be the specified or implicit <schema name> of *UDTN*, let *SS* be the SQL-schema identified by *SN*, and let *UDT* be the data type defined by *UDTD*.

2) If *UDTD* is contained in a <schema definition> and *UDTN* contains a <schema name>, then that <schema name> shall be equivalent to the specified or implicit <schema name> of the containing <schema definition>.

3) *SS* shall not include a user-defined type descriptor or a domain descriptor whose name is equivalent to *UDTN*.

4) None of <instantiable clause>, <finality>, <reference type specification>, <cast to ref>, <cast to type>, <cast to distinct>, or <cast to source> shall be specified more than once.

5) Case:

   a) If <representation> specifies <predefined type>, then *UDTD* defines a *distinct type*.

   b) Otherwise, *UDTD* defines a *structured type*.

6) If <finality> specifies FINAL, then <instantiable clause> shall not specify NOT INSTANTIABLE.

7) If *UDTD* defines a distinct type, then:

   a) Let *PSDT* be the data type identified by <predefined type>.

      Case:

      i) If *PSDT* is an exact numeric type, then let *SDT* be an implementation-defined exact numeric type whose precision is equal to the precision of *PSDT* and whose scale is equal to the scale of *PSDT*.

      ii) If *PSDT* is an approximate numeric type, then let *SDT* be an implementation-defined approximate numeric type whose precision is equal to the precision of *PSDT*.

      iii) Otherwise, let *SDT* be *PSDT*.

   b) <instantiable clause> shall not be specified.

   c) If <finality> is not specified, then FINAL is implicit; otherwise, FINAL shall be specified.

   d) <subtype clause> shall not be specified.

   e) <reference type specification> shall not be specified.

   f) <cast to distinct> and <cast to source> shall not be specified.

   g) If <cast to distinct> is specified, then let *FNUDT* be <cast to distinct identifier>; otherwise, let *FNUDT* be the <qualified identifier> of *UDTN*.

   h) If <cast to source> is specified, then let *FNSDT* be <cast to source identifier>; otherwise, the Syntax Rules of Subclause 9.7, "Type name determination", are applied to *SDT*, yielding an <identifier> *FNSDT*.

8) If *UDTD* specifies a structured type, then:

   a) <cast to distinct> and <cast to type> shall not be specified.

   b) If <subtype clause> is specified, then <reference type specification> shall not be specified.

   c) If <subtype clause> and <reference type specification> are not specified, then <system-generated representation> is implicit.

   d) If <instantiable clause> is not specified, then INSTANTIABLE is implicit.

   e) <finality> shall be specified.

   f) The *originally-defined attributes* of *UDT* are those defined by <attribute definition>s contained in <member list>. No two originally-defined attributes of *UDT* shall have equivalent <attribute name>s.

   g) For each <attribute definition> *ATD* contained in <member list>, let *AN* be the <attribute name> contained in *ATD* and let *DT* be the <data type> contained in *ATD*. The following <original method specification>s are implicit:

```
METHOD AN ( )
    RETURNS DT
    LANGUAGE SQL
    DETERMINISTIC
    CONTAINS SQL
  RETURNS NULL ON NULL INPUT
```

   This is the *original method specification* of the observer function of attribute *AN*.

```
METHOD AN ( ATTR DT )
    RETURNS UDTN
    SELF AS RESULT
    LANGUAGE SQL
    DETERMINISTIC
    CONTAINS SQL
  CALLED ON NULL INPUT
```

   This is the *original method specification* of the mutator function of attribute *AN*.

   h) If <user-defined representation> is specified, then:

      i) Let *BT* be <predefined type>. *BT* is the *representation type of the referencing type* of *UDT*.

      ii) *BT* shall be exact numeric or a character string type that is not a large object string type.

      iii) If <cast to ref> is specified, then let *FNREF* be <cast to ref identifier>; otherwise, let *FNREF* be the <qualified identifier> of *UDTN*.

      iv) Case:

         1) If <cast to type> is specified, then let *FNTYP* be <cast to type identifier>.

         2) Otherwise, the Syntax Rules of Subclause 9.7, "Type name determination", are applied to *BT*, yielding an <identifier> *FNTYP*.

   i) If <derived representation> is specified, then no two <attribute name>s in <list of attributes> shall be equivalent.

**

j) If <subtype clause> is specified, then:

   i) <supertype name> shall not be equivalent to *UDTN*.

   ii) The <supertype name> immediately contained in the <subtype clause> shall identify the descriptor of some structured type *SST*. *UDT* is a direct subtype of *SST*, and *SST* is a direct supertype of *UDT*.

   iii) The descriptor of *SST* shall not include an indication that *SST* is final.

   iv) The inherited attributes of *UDT* are the attributes described by the attribute descriptors included in the descriptor of *SST*.

   v) If <member list> is specified, then no <attribute name> contained in <member list> shall have an attribute name that is equivalent to the attribute name of an inherited attribute.

   vi) If the user-defined type descriptor of *SST* indicates that the referencing type of *SST* has a user-defined representation, then let *BT* be the data type described by the data type descriptor of the representation type of the referencing type of *SST* included in the user-defined type descriptor of *SST*.

      1) If <cast to ref> is specified, then let *FNREF* be <cast to ref identifier>; otherwise, let *FNREF* be the <qualified identifier> of *UDTN*.

      2) Case:

         A) If <cast to type> is specified, then let *FNTYP* be <cast to type identifier>.

         B) Otherwise, the Syntax Rules of Subclause 9.7, "Type name determination", are applied to *BT*, yielding an <identifier> *FNTYP*.

k) If <cast to distinct> or <cast to source> is specified, then exactly one of the following shall be true:

   i) <user-defined representation> is specified.

   ii) <subtype clause> is specified and the user-defined type descriptor of the direct supertype of *UDT* indicates that the referencing type of the direct supertype of *UDT* has a user-defined representation.

9) If <method specification list> is specified, then:

   a) Let *M* be the number of <method specification>s MS_i, 1 (one) $\leq i \leq M$, contained in <method specification list>. Let MN_i be the <method name> of MS_i.

   b) For *i* ranging from 1 (one) to *M*:

      i) If MS_i does not specify INSTANCE, CONSTRUCTOR, or STATIC, then INSTANCE is implicit.

      ii) If MS_i specifies STATIC, then:

         1) None of SELF AS RESULT, SELF AS LOCATOR, and OVERRIDING shall be specified.

         2) MS_i specifies a *static method*.

      iii) If MS_i specifies CONSTRUCTOR, then:

1) SELF AS RESULT shall be specified.
2) OVERRIDING shall not be specified.
3) MN_i shall be equivalent to the <qualified identifier> of *UDTN*.
4) The <returns data type> shall specify *UDTN*.
5) *UDTD* shall define a structured type.
6) MS_i specifies an *SQL-invoked constructor method*.

iv) Let RN_i be $SN.MN_i$.

v) If <specific method name> is not specified, then an implementation-dependent <specific method name> whose <schema name> is equivalent to *SN* is implicit.

vi) If <specific method name> contains a <schema name>, then that <schema name> shall be equivalent to *SN*. If <specific method name> does not contain a <schema name>, then the <schema name> of *SN* is implicit.

vii) The schema identified by the explicit or implicit <schema name> of the <specific method name> shall not include a routine descriptor whose specific name is equivalent to <specific method name> or a user-defined type descriptor that includes a method specification descriptor whose specific method name is equivalent to <specific method name>.

viii) Let PDL_i be the <SQL parameter declaration list> contained in MS_i.

1) No two <SQL parameter name>s contained in PDL_i shall be equivalent.
2) No <SQL parameter name> contained in PDL_i shall be equivalent to SELF.

ix) Let N_i be the number of <SQL parameter declaration>s contained in MS_i. For every <SQL parameter declaration> $PD_{i,j}$, 1 (one) $\leq j \leq N_i$:

1) $PD_{i,j}$ shall not contain <parameter mode>. A <parameter mode> of IN is implicit.
2) $PD_{i,j}$ shall not specify RESULT.
3) <parameter type> $PT_{i,j}$ immediately contained in $PD_{i,j}$ shall not specify ROW.
4) If $PT_{i,j}$ simply contains <locator indication>, then:
   A) MS_i shall not specify or imply LANGUAGE SQL.
   B) $PT_{i,j}$ shall specify either binary large object type, character large object type, array type, multiset type, or user-defined type.

x) If <returns data type> *RT* simply contains <locator indication>, then:

1) LANGUAGE SQL shall not be specified or implied.
2) *RT* shall be either binary large object type, character large object type, array type, multiset type, or user-defined type.

3) <result cast> shall not be specified.

xi) If SELF AS RESULT is specified, then the <returns data type> shall specify *UDTN*.

xii) For k ranging from $(i+1)$ to M, at least one of the following conditions shall be false:

1) MN_i and the <method name> of MS_k are equivalent.

2) Both MS_i and MS_k either specify CONSTRUCTOR or neither specifies CONSTRUCTOR.

3) MS_k has N_i <SQL parameter declaration>s.

4) The data type of $PT_{i,j}$, 1 (one) $\leq j \leq N_i$, is compatible with $PT_{k,j}$.

xiii) The *unaugmented SQL parameter declaration list* of MS_i is the <SQL parameter declaration list> contained in MS_i.

xiv) If MS_i specifies <original method specification>, then:

1) The <method characteristics> of MS_i shall contain at most one <language clause>, at most one <parameter style clause>, at most one <deterministic characteristic>, at most one <SQL-data access indication>, and at most one <null-call clause>.

2) If <language clause> is not specified, then LANGUAGE SQL is implicit.

3) If <deterministic characteristic> is not specified, then NOT DETERMINISTIC is implicit.

4) <SQL-data access indication> shall be specified.

5) If <null-call clause> is not specified, then CALLED ON NULL INPUT is implicit.

6) Case:

A) If LANGUAGE SQL is specified or implied, then:

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

II) <SQL-data access indication> shall not specify NO SQL.

III) <parameter style clause> shall not be specified.

IV) Every <SQL parameter declaration> contained in <SQL parameter declaration list> shall contain an <SQL parameter name>.

B) Otherwise:

I) If <parameter style> is not specified, then PARAMETER STYLE SQL is implicit.

II) If a <result cast> is specified, then let *V* 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 6.12, "<cast specification>":

```
CAST ( V AS RT )
```

III) If <result cast from type> *RCT* simply contains <locator indication>, then *RCT* shall be either binary large object type, character large object type, array type, multiset type, or user-defined type.

7) Let a *conflicting method specification CMS* be a method specification that is included in the descriptor of a proper supertype of *UDT*, such that the following are all true:

A) The method names of *CMS* and MN_i are equivalent.

B) *CMS* and MS_i have the same number of SQL parameters N_i.

C) Let $PCMS_j$, 1 (one) $\leq j \leq N_i$, be the *j*-th SQL parameter in the unaugmented SQL parameter declaration list of *CMS*. Let $PMS_{i,j}$, 1 (one) $\leq j \leq N_i$, be the *j*-th SQL parameter in the unaugmented SQL parameter declaration list of MS_i.

D) For *j* varying from 1 (one) to N_i, the declared type of $PCMS_j$ and the declared type of $PMS_{i,j}$ are compatible.

E) MS_i does not specify CONSTRUCTOR.

F) *CMS* and MS_i either both are not static methods or one of *CMS* and MS_i is a static method and the other is not a static method.

8) There shall be no conflicting method specification.

9) The *augmented SQL parameter declaration list* NPL_i of MS_i is defined as follows:

Case:

A) If MS_i specifies STATIC, then let NPL_i be:

($PD_{i,1}$, ... , PD_{i,N_i})

B) If MS_i specifies SELF AS RESULT and SELF AS LOCATOR, then let NPL_i be:

(SELF *UDTN* RESULT AS LOCATOR, $PD_{i,1}$, ... , PD_{i,N_i})

C) If MS_i specifies SELF AS LOCATOR, then let NPL_i be:

(SELF *UDTN* AS LOCATOR, $PD_{i,1}$, ... , PD_{i,N_i})

D) If MS_i specifies SELF AS RESULT, then let NPL_i be:

(SELF *UDTN* RESULT, $PD_{i,1}$, ... , PD_{i,N_i})

E) Otherwise, let NPL_i be:

(SELF *UDTN*, $PD_{i,1}$, ... , PD_{i,N_i})

F) Let AN_i be the number of <SQL parameter declaration>s in NPL_i.

10) If MS_i does not specify STATIC or CONSTRUCTOR, then there shall be no SQL-invoked function *F* that satisfies all the following conditions:

A) The routine name of *F* and RN_i have equivalent <qualified identifier>s.

B) If *F* is not a static method, then *F* has AN_i SQL parameters; otherwise, *F* has (AN_i-1) SQL parameters.

C) The data type being defined is a proper subtype of

Case:

I) If *F* is not a static method, then the declared type of the first SQL parameter of *F*.

II) Otherwise, the user-defined type whose user-defined type descriptor includes the routine descriptor of *F*.

D) The declared type of the *i*-th SQL parameter in NPL_i, $2 \leq i \leq AN_i$ is compatible with

Case:

I) If *F* is not a static method, then the declared type of *i*-th SQL parameter of *F*.

II) Otherwise, the declared type of the (*i*-1)-th SQL parameter of *F*.

11) If MS_i specifies STATIC, then there shall be no SQL-invoked function *F* that is not a static method that satisfies all the following conditions:

A) The routine name of *F* and RN_i have equivalent <qualified identifier>s.

B) *F* has (AN_i+1) SQL parameters.

C) The data type being defined is a subtype of the declared type of the first SQL parameter of *F*.

D) The declared type of the *i*-th SQL parameter in *F*, $2 \leq i \leq (AN_i+1)$, is compatible with the declared type of the (*i*-1)-th SQL parameter of NPL_i.

xv) If MS_i specifies <overriding method specification>, then:

1) MS_i shall not specify STATIC or CONSTRUCTOR.

2) A <returns clause> contained in MS_i shall not specify a <result cast> or <locator indication>.

3) Let the candidate original method specification *COMS* be an original method specification whose descriptor is included in the descriptor of a proper supertype of the user-defined type being defined, such that the following are all true:

A) The <method name> of *COMS* and MN_i are equivalent.

B) *COMS* and MS_i have the same number of SQL parameters N_i.

C) Let *PCOMS*$_i$, 1 (one) $\leq i \leq N_i$, be the *i*-th SQL parameter in the unaugmented SQL parameter declaration list of *COMS*. Let *POVMS*$_i$, 1 (one) $\leq i \leq N_i$, be the *i*-th SQL parameter in the unaugmented SQL parameter declaration list of *MS*$_i$.

D) For *i* varying from 1 (one) to N_i, the Syntax Rules of Subclause 9.16, "Data type identity", are applied with the declared type of *PCOMS*$_i$ and the declared type of *POVMS*$_i$.

E) The descriptor of *COMS* shall not include an indication that STATIC or CONSTRUCTOR was specified.

4) There shall exist exactly one *COMS*.

5) *COMS* shall not be the corresponding method specification of a mutator or observer function.

NOTE 302 — "Corresponding method specification" is defined in Subclause 11.50, "<SQL-invoked routine>".

6) For *j* ranging from 1 (one) to N_i, all of the following shall be true:

A) If *POVMS*$_j$ contains an <SQL parameter name> *PNM1*, then *PCOMS*$_j$ contains an <SQL parameter name> that is equivalent to *PNM1*.

B) If *PCOMS*$_j$ contains an <SQL parameter name> *PNM2*, then *POVMS*$_j$ contains an <SQL parameter name> that is equivalent to *PNM2*.

C) If *POVMS*$_j$ contains a <locator indication>, then *PCOMS*$_j$ contains a <locator indication>.

D) If *PCOMS*$_j$ contains a <locator indication>, then *POVMS*$_j$ contains a <locator indication>.

7) Let *ROVMS* be the <returns data type> of *MS*$_i$. Let *RCOMS* be the <returns data type> of *COMS*.

Case:

A) If *RCOMS* is a user-defined type, then:

I) Let a *candidate overriding method specification COVRMS* be a method specification that is included in the descriptor of a proper supertype of *UDT*, such that all of the following are true:

1) The <method name> of *COVRMS* and *MN*$_i$ are equivalent.

2) *COVRMS* and *MS*$_i$ have the same number of SQL parameters N_i.

3) Let *PCOVRMS*$_i$, 1 (one) $\leq i \leq N_i$, be the *i*-th SQL parameter in the unaugmented SQL parameter declaration list of *COVRMS*. Let *POVMS*$_i$, 1 (one) $\leq i \leq N_i$, be the *i*-th SQL parameter in the unaugmented SQL parameter declaration list of *MS*$_i$.

4) For *i* varying from 1 (one) to N_i, the Syntax Rules of Subclause 9.16, "Data type identity", are applied with the declared type of *PCOVRMS*$_i$ and the declared type of *POVMS*$_i$.

II) Let *NOVMS* be the number of candidate overriding method specifications. For *i* varying from 1 (one) to *NOVMS*, *ROVMS* shall be a subtype of the <returns data type> of the *i*-th candidate overriding method specification.

B) Otherwise, the Syntax Rules of Subclause 9.16, “Data type identity”, are applied with *RCOMS* and *ROVMS*.

8) The augmented SQL parameter declaration list *ASPDL* of MS_i is formed from the augmented SQL parameter declaration list of *COMS* by replacing the <data type> of the first parameter (named SELF) with *UDTN*.

9) There shall be no SQL-invoked function *F* that satisfies all the following conditions:

A) The routine name of *F* and the RN_i have equivalent <qualified identifier>s.

B) *F* and *ASPDL* have the same number *N* of SQL parameters.

C) The data type being defined is a proper subtype of the declared type of the first SQL parameter of *F*.

D) The declared type of $POVMS_i$, 1 (one) $\leq i \leq N$, is compatible with the declared type of SQL parameter P_{i+1} of *F*.

E) *F* is not an SQL-invoked method.

Access Rules

1) Let *A* be the <authorization identifier> that owns *SS*. If a <user-defined type definition> is contained in an <SQL-client module definition>, then the enabled authorization identifiers shall include *A*.

2) The applicable privileges for *A* shall include UNDER on the <user-defined type name> specified in <subtype clause>.

General Rules

1) A user-defined type descriptor *UDTDS* that describes *UDT* is created. *UDTDS* includes:

a) The user-defined type name *UDTN*.

b) If *UDT* is a distinct type or INSTANTIABLE is specified or implicit, then an indication that *UDT* is instantiable; otherwise, an indication that *UDT* is not instantiable.

c) An indication of whether the user-defined type is final or not final.

d) An indication of whether *UDT* is a distinct type or a structured type.

e) If *UDT* is a distinct type, then the data type descriptor of *SDT*.

f) If *UDT* is a structured type, then:

i) For each inherited attribute *IA* of *UDT*, the attribute descriptor of *IA* and an indication that *IA* is an inherited attribute.

ii) For each originally-defined attribute *ODA* of *UDT*, the attribute descriptor of *ODA* and an indication that *ODA* is an originally-defined attribute.

iii) The name of the direct supertype of *UDT*.

iv) A transform descriptor with an empty list of groups.

v) Case:

1) If <user-defined representation> is specified, then an indication that the referencing type of *UDT* has a user-defined representation, along with the data type descriptor of the representation type of the referencing type of *UDT*.

2) If <derived representation> is specified, then an indication that the referencing type of *UDT* has a derived representation, along with the attributes specified by <list of attributes>.

3) Otherwise, an indication that the referencing type of *UDT* has a system-defined representation.

vi) If <subtype clause> is specified, then let *SUDT* be the direct supertype of *UDT* and let *DSUDT* be the user-defined type descriptor of *SUDT*. Let *RUDT* be the referencing type of *UDT* and let *RSUDT* be the referencing type of *SUDT*.

Case:

1) If *DSUDT* indicates that *RSUDT* has a user-defined representation, then an indication that *RUDT* has a user-defined representation and the data type descriptor of the representation type of *RSUDT* included in *DSUDT*.

2) If *DSUDT* indicates that *RSUDT* has a derived representation, then an indication that *RUDT* has a derived representation and the list of attributes included in *DSUDT*.

3) If *DSUDT* indicates that *RSUDT* has a system-defined representation, then an indication that *RUDT* has a system-defined representation.

vii) The ordering form NONE.

viii) The ordering category STATE.

g) If <method specification list> is specified, then for every <original method specification> *ORMS* contained in <method specification list>, a method specification descriptor that includes:

i) An indication that the method specification is original.

ii) An indication of whether STATIC or CONSTRUCTOR is specified.

iii) The <method name> of *ORMS*.

iv) The <specific method name> of *ORMS*.

v) The <SQL parameter declaration list> contained in *ORMS* (augmented, if STATIC is not specified in *ORMS*, to include the implicit first parameter with parameter name SELF).

vi) The <language name> contained in the explicit or implicit <language clause>.

vii) The explicit or implicit <parameter style> if the <language name> is SQL.

viii) The <returns data type>.

ix) The <result cast from type>, if any.

x) An indication of whether the method is deterministic.

xi) An indication of whether the method possibly modifies SQL-data, possibly reads SQL-data, possibly contains SQL, or does not possibly contain SQL.

xii) An indication of whether the method should not be invoked if any argument is the null value.

h) If <method specification list> is specified, then for every <overriding method specification> *OVMS* contained in <method specification list>, let *DCMS* be the descriptor of the corresponding original method specification. The method specification descriptor of *OVMS* includes:

i) An indication that the method specification is overriding.

ii) The <method name> of *OVMS*.

iii) The <specific method name> of *OVMS*.

iv) The <SQL parameter declaration list> contained in *OVMS* (augmented to include the implicit first parameter with parameter name SELF).

v) The <language name> included in *DCMS*.

vi) The <parameter style> included in *DCMS* (if any).

vii) The <returns data type> of *OVMS*.

viii) The <result cast from type> included in *DCMS* (if any).

ix) The determinism indication included in *DCMS*.

x) The SQL-data access indication included in *DCMS*.

xi) The indication included in *DCMS*, whether the method should not be invoked if any argument is the null value.

2) If *UDTD* specifies a distinct type, then:

a) The degree of *UDT* is 0 (zero).

b) The following SQL-statements are executed without further Access Rule checking:

```
CREATE FUNCTION SN.FNUDT ( SDTP SDT )
    RETURNS UDTN
    LANGUAGE SQL
    DETERMINISTIC
  RETURN RV1
CREATE FUNCTION SN.FNSDT ( UDTP UDTN )
    RETURNS SDT
    LANGUAGE SQL
    DETERMINISTIC
  RETURN RV2
CREATE CAST ( UDTN AS SDT )
    WITH FUNCTION FNSDT ( UDTN )
    AS ASSIGNMENT
CREATE CAST (SDT AS UDTN)
    WITH FUNCTION SN.FNUDT ( SDT )
    AS ASSIGNMENT
CREATE TRANSFORM FOR UDTN
    FNUDT ( FROM SQL WITH FUNCTION FNSDT ( UDTN ),
              TO SQL WITH FUNCTION SN.FNUDT(SDT) )
```

where: *SN* is the explicit or implicit <schema name> of *UDTN*; *RV1* is an implementation-dependent <value expression> such that for every invocation of *SN.FNUDT* with argument value *AV1*, *RV1* evaluates to the representation of *AV1* in the data type identified by *UDTN*; *RV2* is an implementation-dependent <value expression> such that for every invocation of *SN.FNSDT* with argument value *AV2*, *RV2* evaluates to the representation of *AV2* in the data type *SDT*, and *SDTP* and *UDTP* are <SQL parameter name>s arbitrarily chosen.

c) Case:

i) If *SDT* is not a large object type, then the following SQL-statement is executed without further Access Rule checking:

```
CREATE ORDERING FOR UDTN
ORDER FULL BY
MAP WITH FUNCTION FNSDT(UDTN)
FOR UDTN
```

ii) If *SDT* is a large object type, and the SQL implementation supports Feature T042, "Extended LOB data type support", then the following SQL-statement is executed without further Access Rule checking:

```
CREATE ORDERING FOR UDTN
ORDER EQUALS ONLY BY
MAP WITH FUNCTION FNSDT(UDTN)
FOR UDTN
```

NOTE 303 — If *SDT* is a large object type, and the SQL implementation does not support Feature T042, "Extended LOB data type support", then no ordering for *UDTN* is created.

3) If *UDTD* specifies a structured type, then:

a) The degree of *UDT* is the number of attributes of *UDT*, including inherited attributes. The ordinal position of an inherited attribute is its ordinal position in the direct supertype of *UDT*. The ordinal position of an attribute that is an originally-defined attribute is the ordinal position of its corresponding <attribute definition> in <member list> plus the number of inherited attributes.

b) If INSTANTIABLE is specified, then let *V* be a value of the most specific type *UDT* such that, for every attribute *ATT* of *UDT*, invocation of the corresponding observer function on *V* yields the default value for *ATT*. The following <SQL-invoked routine> is effectively executed:

```
CREATE FUNCTION UDTN () RETURNS UDTN
    RETURN V
```

This SQL-invoked function is the *constructor function* for *UDT*.

c) If <user-defined representation> is specified or if <subtype clause> is specified and the user-defined type descriptor of the direct supertype of *UDT* indicates that the referencing type of the direct supertype of *UDT* has a user-defined representation, then the following SQL-statements are executed without further Access Rule checking:

```
CREATE FUNCTION SN.FNREF ( BTP BT )
    RETURNS REF(UDTN)
    LANGUAGE SQL
    DETERMINISTIC
    STATIC DISPATCH
  RETURN RV1
CREATE FUNCTION SN.FNTYP ( UDTNP REF(UDTN) )
    RETURNS BT
    LANGUAGE SQL
    DETERMINISTIC
    STATIC DISPATCH
  RETURN RV2
CREATE CAST ( BT AS REF(UDTN) )
    WITH FUNCTION SN.FNREF(BT)
CREATE CAST ( REF(UDTN) AS BT )
    WITH FUNCTION SN.FNTYP(REF(UDTN) )
```

where: *SN* is the explicit or implicit <schema name> of *UDTN*; *RV1* is an implementation-dependent <value expression> such that for every invocation of *SN.FNREF* with argument value *AV1*, *RV1* evaluates to the representation of *AV1* in the data type identified by REF(*UDTN*); *RV2* is an implementation-dependent <value expression> such that for every invocation of *SN.FNTYP* with argument value *AV2*, *RV2* evaluates to the representation of *AV2* in the data type *BT*; and *UDTNP* is an <SQL parameter name> arbitrarily chosen.

4) A privilege descriptor is created that defines the USAGE privilege on *UDT* to *A*. This privilege is grantable. The grantor for this privilege descriptor is set to the special grantor value "_SYSTEM".

5) If *UDTD* specifies a structured type, then a privilege descriptor is created that defines the UNDER privilege on *UDT* to *A*. The grantor for the privilege descriptor is set to the special grantor value "_SYSTEM". This privilege is grantable if and only if *A* holds the UNDER privilege on the direct supertype of *UDT* WITH GRANT OPTION.

Conformance Rules

1) Without Feature S023, "Basic structured types", conforming SQL language shall not contain a <member list>.

2) Without Feature S024, "Enhanced structured types", conforming SQL language shall not contain an <instantiable clause> that contains NOT INSTANTIABLE.

3) Without Feature S024, "Enhanced structured types", conforming SQL language shall not contain an <original method specification> that immediately contains SELF AS RESULT.

4) Without Feature S024, "Enhanced structured types", conforming SQL language shall not contain a <method characteristics> that contains a <parameter style> that contains GENERAL.

5) Without Feature S024, "Enhanced structured types", conforming SQL language shall not contain an <original method specification> that contains an <SQL-data access indication> that immediately contains NO SQL.

6) Without Feature T571, "Array-returning external SQL-invoked functions", conforming SQL language shall not contain a <method specification> that contains a <returns clause> that satisfies either of the following conditions:

   a) A <result cast from type> is specified that simply contains an <array type> and does not contain a <locator indication>.

   b) A <result cast from type> is not specified and <returns data type> simply contains an <array type> and does not contain a <locator indication>.

7) Without Feature T572, "Multiset-returning external SQL-invoked functions", conforming SQL language shall not contain a <method specification> that contains a <returns clause> that satisfies either of the following conditions:

   a) A <result cast from type> is specified that simply contains a <multiset type> and does not contain a <locator indication>.

   b) A <result cast from type> is not specified and <returns data type> simply contains a <multiset type> and does not contain a <locator indication>.

8) Without Feature S043, "Enhanced reference types", conforming SQL language shall not contain a <reference type specification>.

9) Without Feature S024, "Enhanced structured types", conforming SQL language shall not contain a <partial method specification> that contains INSTANCE or STATIC.

10) Without Feature S023, "Basic structured types", conforming SQL language shall not contain a <method specification list>.

11) Without Feature S025, "Final structured types", in conforming SQL language, a <user-defined type definition> that defines a structured type shall contain a <finality> that is NOT FINAL.

12) Without Feature S028, "Permutable UDT options list", conforming SQL language shall not contain a <user-defined type option list> in which <instantiable clause>, if specified, <finality>, <reference type specification>, if specified, <cast to ref>, if specified, <cast to type>, if specified, <cast to distinct>, if specified, and <cast to source>, if specified, do not appear in that sequence.

11.42 <attribute definition>

Function

Define an attribute of a structured type.

Format

```
<attribute definition> ::=
    <attribute name> <data type>
    [ <attribute default> ]
    [ <collate clause> ]

<attribute default> ::= <default clause>
```

Syntax Rules

1) An <attribute definition> defines a certain component of some structured type. Let *UDT* be that structured type, let *UDTN* be its name, and let *SS* be the SQL-schema whose descriptor includes the descriptor of *UDT*.

2) Let *AN* be the <attribute name> contained in the <attribute definition>.

3) The declared type *DT* of the attribute is <data type>.

4) <collate clause> shall not be both specified in <data type> and immediately contained in <attribute definition>. If <collate clause> is immediately contained in <attribute definition>, then it is equivalent to specifying an equivalent <collate clause> in <data type>.

5) *DT* shall not be based on *UDT*.

   NOTE 304 — The notion of one data type being based on another data type is defined in Subclause 4.1, "Data types".

6) If *DT* is a <character string type> and does not contain a <character set specification>, then the default character set for *SS* is implicit.

Access Rules

None.

General Rules

1) A data type descriptor is created that describes *DT*.

2) Let *A* be the attribute defined by <attribute definition>.

3) An attribute descriptor is created that describes *A*. The attribute descriptor includes:

   a) *AN*, the name of the attribute.

b) The data type descriptor of *DT*.

c) The ordinal position of the attribute in *UDT*.

d) The implicit or explicit <attribute default>.

e) The name *UDTN* of the user-defined type *UDT*.

4) An SQL-invoked method *OF* is created whose signature and result data type are as given in the descriptor of the original method specification of the observer function of *A*. Let *V* be a value in *UDT*. If *V* is the null value, then the invocation *V.AN*() of *OF* returns the result of:

```
CAST (NULL AS DT)
```

Otherwise, *V.AN*() returns the value of *A* in *V*.

NOTE 305 — The original method specification of the observer function of *A* is defined in the Syntax Rules of Subclause 11.41, "<user-defined type definition>".

NOTE 306 — The descriptor of *OF* is created under the General Rules of Subclause 11.50, "<SQL-invoked routine>".

5) An SQL-invoked method *MF* is created whose signature and result data type are as given in the descriptor of the original method specification of the mutator function of *A*. Let *V* be a value in *UDT* and let *AV* be a value in *DT*. If *V* is the null value, then the invocation *V.AN*(*AV*) of *MF* raises an exception condition: *data exception — null value substituted for mutator subject parameter*; otherwise, the invocation *V.AN*(*AV*) returns *V2* such that *V2.AN*() = *AV* and for every other observer function *ANX* of *UDT*, *V2.ANX*() = *V.ANX*().

NOTE 307 — The original method specification of the mutator function of *A* is defined in the Syntax Rules of Subclause 11.41, "<user-defined type definition>".

NOTE 308 — The descriptor of *MF* is created under the General Rules of Subclause 11.50, "<SQL-invoked routine>".

Conformance Rules

1) Without Feature S023, "Basic structured types", conforming SQL language shall not contain an <attribute definition>.

2) Without Feature F692, "Extended collation support", conforming SQL language shall not contain an <attribute definition> that immediately contains a <collate clause>.

3) Without Feature S024, "Enhanced structured types", conforming SQL language shall not contain an <attribute default>.

4) Without Feature S026, "Self-referencing structured types", conforming SQL language shall not contain a <data type> simply contained in an <attribute definition> that is not be a <reference type> whose <referenced type> is equivalent to the <schema-resolved user-defined type name> simply contained in the <user-defined type definition> that contains <attribute definition>.

11.43 <alter type statement>

Function

Change the definition of a user-defined type.

Format

```
<alter type statement> ::=
    ALTER TYPE <schema-resolved user-defined type name> <alter type action>

<alter type action> ::=
    <add attribute definition>
  | <drop attribute definition>
  | <add original method specification>
  | <add overriding method specification>
  | <drop method specification>
```

Syntax Rules

1) Let *DN* be the <schema-resolved user-defined type name> and let *D* be the data type identified by *DN*.

2) The schema identified by the explicit or implicit schema name of the <schema-resolved user-defined type name> shall include the descriptor of *D*. Let *S* be that schema.

3) The scope of the <schema-resolved user-defined type name> is the entire <alter type statement>.

4) If <alter type action> contains <add attribute definition>, <drop attribute definition>, or <add overriding method specification>, then *D* shall be a structured type.

5) Let *A* be the <authorization identifier> that owns the schema *S*.

Access Rules

1) If an <alter type statement> is contained in an <SQL-client module definition>, then the enabled authorization identifiers shall include *A*.

2) The applicable privileges for *A* shall include UNDER on each proper supertype of *D*.

General Rules

1) The user-defined type descriptor of *D* is modified as specified by <alter type action>.

Conformance Rules

1) Without Feature S024, "Enhanced structured types", conforming SQL language shall not contain an <alter type statement>.

11.44 <add attribute definition>

Function

Add an attribute to a user-defined type.

Format

```
<add attribute definition> ::= ADD ATTRIBUTE <attribute definition>
```

Syntax Rules

1) Let *D* be the user-defined type identified by the <schema-resolved user-defined type name> immediately contained in the containing <alter type statement>. Let *SPD* be any supertype of *D*. Let *SBD* be any subtype of *D*.

2) Let *RD* be the reference type whose referenced type is *D*. Let *SPRD* be any supertype of *RD*. Let *SBRD* be any subtype of *RD*. Let *AD* be any collection type whose element type is *D*. Let *SPAD* be any collection type whose element type is *SPD* or *SPRD*. Let *SBAD* be any collection type whose element type is *SBD* or *SBRD*.

3) The declared type of a column of a base table shall not be *SPRD*, *SBRD*, *SPAD*, or *SBAD*.

4) The declared type of a column of a base table shall not be based on *D*.

   NOTE 309 — The notion of one data type type being based on another data type is defined in Subclause 4.1, "Data types".

5) *SBD* shall not be the structured type of a referenceable table.

6) Let *M* be the mutator function resulting from the <attribute definition>, had that <attribute definition> been simply contained in the <user-defined type definition> for *D*. There shall be no SQL-invoked routine *F* that satisfies all of the following conditions:

   a) The routine name included in the descriptor of *F* and the <schema qualified routine name> of *M* have equivalent <qualified identifier>s.

   b) *F* has 2 SQL parameters.

   c) The declared type of the first SQL parameter of *F* is a subtype or supertype of *D*.

   d) The declared type of the second SQL parameter of *F* is a compatible with the second SQL parameter of *M*.

7) Let *O* be the observer function resulting from the <attribute definition>, had that <attribute definition> been simply contained in the <user-defined type definition> for *D*. There shall be no SQL-invoked routine *F* that satisfies all of the following conditions:

   a) The <schema qualified routine name> of *O* and the routine name included in the descriptor of *F* have equivalent <qualified identifier>s.

   b) *F* has 1 (one) SQL parameter.

c) The declared type of the first SQL parameter of *F* is a subtype or supertype of *D*.

Access Rules

None.

General Rules

1) The attribute defined by the <attribute definition> is added to *D*.
2) In all other respects, the specification of an <attribute definition> in an <alter type statement> has the same effect as specification of the <attribute definition> simply contained in the <user-defined type definition> for *D* would have had. In particular, the degree of *D* is increased by 1 (one) and the ordinal position of that attribute is equal to the new degree of *D* as specified in the General Rules of Subclause 11.42, "<attribute definition>".
3) Let *A* be the attribute defined by <attribute definition>. Let *CPA* be a copy of the descriptor of *A*, modified to include an indication that the attribute is an inherited attribute.
4) For each proper subtype *PSBD* of *D*:
   a) Let *DPSBD* be the descriptor of *PSBD*, let *N* be the number of attribute descriptors included in *DPSBD*, and let DA_i, 1 (one) ≤ *i* ≤ *N*, be the attribute descriptors included in *DPSBD*.
   b) For every *i* between 1 (one) and *N*, if DA_i is the descriptor of an originally-defined attribute, then increase the ordinal position included in DA_i by 1 (one).
   c) Include *CPA* in *DPSBD*.

Conformance Rules

None.

11.45 <drop attribute definition>

Function

Destroy an attribute of a user-defined type.

Format

```
<drop attribute definition> ::= DROP ATTRIBUTE <attribute name> RESTRICT
```

Syntax Rules

1) Let *D* be the user-defined type identified by the <schema-resolved user-defined type name> immediately contained in the containing <alter type statement>.
2) Let *A* be the attribute identified by the <attribute name> *AN*.
3) *A* shall be an attribute of *D* that is not an inherited attribute, and *A* shall not be the only attribute of *D*.
4) Let *SPD* be any supertype of *D*. Let *SBD* be any subtype of *D*. Let *RD* be the reference type whose referenced type is *D*. Let *SPRD* be any supertype of *RD*. Let *SBRD* be any subtype of *RD*. Let *AD* be any collection type whose element type is *D*. Let *SPAD* be any collection type whose element type is *SPD* or *SPRD*. Let *SBAD* be any collection type whose element type is *SBD* or *SBRD*.
5) The declared type of any column of any base table shall not be *SPRD*, *SBRD*, *SPAD*, or *SBAD*.
6) The declared type of any column of any base table shall not be based on *D*.

   NOTE 310 — The notion of one data type type being based on another data type is defined in Subclause 4.1, "Data types".
7) *SBD* shall not be the structured type of a referenceable table.
8) Let *R1* be the mutator function and let *R2* be the observer function of *A*.
   a) *R1* and *R2* shall not be the subject routine of any <routine invocation>, <method invocation>, <static method invocation>, or <method reference> that is contained in any of the following:
      i) The SQL routine body of any routine descriptor.
      ii) The <query expression> of any view descriptor.
      iii) The <search condition> of any constraint descriptor.
      iv) The trigger action of any trigger descriptor.
   b) The specific names of *R1* and *R2* shall not be included in any user-defined cast descriptor.
   c) *R1* and *R2* shall not be the ordering function in the descriptor of any user-defined type.

Access Rules

None.

General Rules

1) The descriptor of *A* is removed from the descriptor of every *SBD*.
2) The descriptor of *A* is destroyed.
3) The descriptors of the mutator and observer functions of *A* are destroyed.
4) The degree of every *SBD* is reduced by 1 (one). The ordinal position of all attributes having an ordinal position greater than the ordinal position of *A* in *SBD* is reduced by 1 (one).

Conformance Rules

None.

11.46 <add original method specification>

Function

Add an original method specification to a user-defined type.

Format

```
<add original method specification> ::= ADD <original method specification>
```

Syntax Rules

1) Let *D* be the user-defined type identified by the <schema-resolved user-defined type name> *DN* immediately contained in the containing <alter type statement>. Let *SN* be the specified or implied <schema name> of *DN*. Let *SPD* be any supertype of *D*, if any. Let *SBD* be any subtype of *D*, if any.

2) Let *ORMS* and *PORMS* be the <original method specification> and its immediately contained <partial method specification>, respectively.

3) Let *MN*, *MPDL* and *MCH* be the <method name>, the <SQL parameter declaration list> and the <method characteristics>, respectively, that are simply contained in *ORMS*. *MPDL* is called the *unaugmented SQL parameter declaration list* of *ORMS*.

4) If *PORMS* does not specify INSTANCE, CONSTRUCTOR, or STATIC, then INSTANCE is implicit.

5) If *PORMS* specifies CONSTRUCTOR, then:

   a) SELF AS RESULT shall be specified.

   b) *MN* shall be equivalent to the <qualified identifier> of *DN*.

   c) The <returns data type> shall specify *DN*.

   d) *D* shall be a structured type.

   e) *PORMS* specifies an *SQL-invoked constructor method*.

6) If *PORMS* specifies STATIC, then:

   a) Neither SELF AS RESULT nor SELF AS LOCATOR shall be specified.

   b) *PORMS* specifies a static method.

7) Let *RN* be *SN.MN*.

8) Case:

   a) If *PORMS* does not specify <specific method name>, then an implementation-dependent <specific method name> is implicit whose <schema name> is equivalent to *SN*.

   b) Otherwise:

      Case:

i) If <specific method name> contains a <schema name>, then that <schema name> shall be equivalent to *SN*.

ii) Otherwise, the <schema name> *SN* is implicit.

The schema identified by the explicit or implicit <schema name> of the <specific method name> shall not include a routine descriptor whose specific name is equivalent to <specific method name> or a user-defined type descriptor that includes a method specification descriptor whose specific method name is equivalent to <specific method name>.

9) *MCH* shall contain at most one <language clause>, at most one <parameter style clause>, at most one <deterministic characteristic>, at most one <SQL-data access indication>, and at most one <null-call clause>.

a) If <language clause> is not specified in *MCH*, then LANGUAGE SQL is implicit.

b) Case:

i) If LANGUAGE SQL is specified or implied, then:

1) <parameter style clause> shall not be specified.

2) <SQL-data access indication> shall not specify NO SQL.

3) Every <SQL parameter declaration> contained in <SQL parameter declaration list> shall contain an <SQL parameter name>.

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

ii) Otherwise:

1) If <parameter style clause> is not specified, then PARAMETER STYLE SQL is implicit.

2) If a <result cast> is specified, then let *V* 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 6.12, "<cast specification>":

```
CAST ( V AS RT )
```

3) If <result cast from type> *RCT* simply contains <locator indication>, then *RCT* shall be either binary large object type, character large object type, array type, multiset type, or user-defined type.

c) If <deterministic characteristic> is not specified in *MCH*, then NOT DETERMINISTIC is implicit.

d) If <SQL-data access indication> is not specified, then CONTAINS SQL is implicit.

e) If <null-call clause> is not specified in *MCH*, then CALLED ON NULL INPUT is implicit.

10) No two <SQL parameter name>s contained in *MPDL* shall be equivalent.

11) No <SQL parameter name> contained in *MPDL* shall be equivalent to SELF.

12) Let *N* be the number of <SQL parameter declaration>s contained in *MPDL*. For every <SQL parameter declaration> PD_j, 1 (one) $\leq j \leq N$:

a) PD_j shall not contain <parameter mode>. A <parameter mode> of IN is implicit.

b) PD_j shall not specify RESULT.

c) <parameter type> PT_j immediately contained in PD_j shall not specify ROW.

d) If PT_j simply contains <locator indication>, then:

   i) *MCH* shall not specify LANGUAGE SQL, nor shall LANGUAGE SQL be implied.

   ii) PT_j shall specify either binary large object type, character large object type, array type, multiset type, or user-defined type.

13) If <returns data type> *RT* simply contains <locator indication>, then:

   a) *MCH* shall not be specify LANGUAGE SQL, nor shall LANGUAGE SQL be implied.

   b) *RT* shall be either binary large object type, character large object type, array type, multiset type, or user-defined type.

   c) <result cast> shall not be specified.

14) If SELF AS RESULT is specified, then the <returns data type> shall specify *DN*.

15) Case:

   a) If *ORMS* specifies CONSTRUCTOR, then let a *conflicting method specification CMS* be a method specification whose descriptor is included in the descriptor of *D*, such that the following are all true:

      i) *MPDL* and the unaugmented SQL parameter list of *CMS* have the same number *N* of SQL parameters.

      ii) Let $PCMS_j$, 1 (one) $\leq j \leq N$, be the *j*-th SQL parameter in the unaugmented SQL parameter declaration list of *CMS*. Let PMS_j, 1 (one) $\leq j \leq N$, be the *j*-th SQL parameter in the unaugmented SQL parameter declaration list *MPDL*.

      iii) For *j* varying from 1 (one) to *N*, the declared type of $PCMS_j$ and the declared type of PMS_j are compatible.

      iv) *CMS* is an SQL-invoked constructor method.

   b) Otherwise, let a *conflicting method specification CMS* be a method specification whose descriptor is included in the descriptor of some *SPD* or *SBD*, such that the following are all true:

      i) *MN* and the method name included in the descriptor of *CMS* are equivalent.

      ii) *MPDL* and the unaugmented SQL parameter list of *CMS* have the same number *N* of SQL parameters.

      iii) Let $PCMS_j$, 1 (one) $\leq j \leq N$, be the *j*-th SQL parameter in the unaugmented SQL parameter declaration list of *CMS*. Let PMS_j, 1 (one) $\leq j \leq N$, be the *j*-th SQL parameter in the unaugmented SQL parameter declaration list *MPDL*.

iv) For j varying from 1 (one) to N, the declared type of $PCMS_j$ and the declared type of PMS_j are compatible.

v) *CMS* and *ORMS* either both are not instance methods or one of *CMS* and *ORMS* is a static method and the other is an instance method.

16) There shall be no conflicting method specification.

17) Let MP_i, 1 (one) $\leq i \leq N$, be the i-th <SQL parameter declaration> contained in *MPDL*. The augmented SQL parameter declaration list *NPL* of *ORMS* is defined as follows:

Case:

a) If *PORMS* specifies STATIC, then let *NPL* be:

(MP_1, ..., MP_N)

b) If *ORMS* specifies SELF AS RESULT and SELF AS LOCATOR, then let *NPL* be:

(SELF *DN* RESULT AS LOCATOR, MP_1, ..., MP_N)

c) If *ORMS* specifies SELF AS LOCATOR , then let *NPL* be:

(SELF *DN* AS LOCATOR, MP_1, ..., MP_N)

d) If *ORMS* specifies SELF AS RESULT, then let *NPL* be:

(SELF *DN* RESULT, MP_1 , ..., MP_N)

e) Otherwise, let *NPL* be:

(SELF *DN*, MP_1, ..., MP_N)

Let *AN* be the number of <SQL parameter declaration>s in *NPL*.

18) If *PORMS* does not specify STATIC or CONSTRUCTOR, then there shall be no SQL-invoked function *F* that satisfies all the following conditions:

a) *F* is not an SQL-invoked method.

b) The <routine name> of *F* and *RN* have equivalent <qualified identifier>s.

c) *F* has *AN* SQL parameters.

d) *D* is a subtype or supertype of the declared type of the first SQL parameter of *F*.

e) The declared type of the i-th SQL parameter in *NPL*, $2 \leq i \leq AN$ is compatible with the declared type of i-th SQL parameter of *F*.

19) If *PORMS* specifies STATIC, then there shall be no SQL-invoked function *F* that is not a static method that satisfies all the following conditions:

a) The <routine name> of *F* and *RN* have equivalent <qualified identifier>s.

b) *F* has (*AN*+1) SQL parameters.

c) *D* is a subtype or supertype of the declared type of the first SQL parameter of *F*.

d) The declared type of the *i*-th SQL parameter of *F*, $2 \leq i \leq (AN+1)$, is compatible with the declared type of the (*i*-1)-th SQL parameter of *NPL*.

Access Rules

None.

General Rules

1) Let *STDS* be the descriptor of *D*. A method specification descriptor *DOMS* is created for *ORMS*. *DOMS* includes:

   a) An indication that the method specification is original.

   b) An indication of whether STATIC or CONSTRUCTOR is specified.

   c) The <method name> *MN*.

   d) The <specific method name> contained in *PORMS*.

   e) The augmented SQL parameter declaration list *NPL*.

   f) For every parameter descriptor of a parameter of *NPL*, a locator indication (if specified).

   g) The <returns data type> contained in *PORMS*.

   h) The <result cast from type> contained in *PORMS* (if any).

   i) The locator indication, if a <locator indication> is contained in the <returns clause> of *PORMS* (if any).

   j) The <language name> explicitly or implicitly contained in *MCH*.

   k) The explicit or implicit <parameter style> contained in *MCH*, if the <language name> is not SQL.

   l) The determinism indication contained in *MCH*.

   m) An indication of whether the method possibly modifies SQL-data, possibly reads SQL-data, possibly contains SQL, or does not possibly contain SQL.

   n) An indication of whether the method should not be invoked if any argument is the null value.

2) *DOMS* is added to *STDS*.

3) Let *N* be the number of table descriptors that include the user-defined type name of a subtype of *D*.

   For *i* varying from 1 (one) to *N*:

   a) Let TN_i be the <table name> included in the *i*-th such table descriptor.

b) For every table privilege descriptor that specifies TN_i and a privilege of SELECT, a new table/method privilege descriptor is created that specifies TN_i, the same action, grantor, and grantee, and the same grantability, and the <specific method name> contained in *ORMS*.

Conformance Rules

None.

11.47 <add overriding method specification>

Function

Add an overriding method specification to a user-defined type.

Format

```
<add overriding method specification> ::=
    ADD <overriding method specification>
```

Syntax Rules

1) Let *OVMS* be the <overriding method specification> immediately contained in <add overriding method specification>. Let *D* be the user-defined type identified by the <schema-resolved user-defined type name> *DN* immediately contained in the <alter type statement> containing *OVMS*. Let *SN* be the specified or implied <schema name> of *DN*. Let *SPD* be any supertype of *D*, if any. Let *SBD* be any subtype of *D*, if any.

2) Let *POVMS* be the <partial method specification> immediately contained in *OVMS*. *POVMS* shall not specify STATIC or CONSTRUCTOR.

3) Let *MN*, *RTC* and *MPDL* be <routine name>, the <returns clause> and the <SQL parameter declaration list> immediately contained in *POVMS*.

4) *MN* shall not be equivalent to the <qualified identifier> of the user-defined type name of any *SPD* or *SBD* other than *D*.

5) Let *RN* be *SN.MN*.

6) Case:

   a) If *POVMS* does not specify <specific method name>, then an implementation-dependent <specific method name> is implicit whose <schema name> is equivalent to *SN*.

   b) Otherwise,

      Case:

      i) If <specific method name> contains a <schema name>, then that <schema name> shall be equivalent to *SN*.

      ii) Otherwise, the <schema name> *SN* is implicit.

   The schema identified by the explicit or implicit <schema name> of the <specific method name> shall not include a routine descriptor whose specific name is equivalent to <specific method name> or a user-defined type descriptor that includes a method specification descriptor whose specific method name is equivalent to <specific method name>.

7) *RTC* shall not specify a <result cast> or <locator indication>.

8) Let the *candidate original method specification COMS* be an original method specification that is included in the descriptor of a proper supertype of the user-defined type of *D*, such that the following are all true:

   a) *MN* and the <method name> of *COMS* are equivalent.

   b) Let *N* be the number of elements of the augmented SQL parameter declaration list *UPCOMS* generally included in the descriptor of *COMS*. *MPDL* contains (*N*-1) SQL parameter declarations.

   c) For *i* varying from 2 to *N*, the Syntax Rules of Subclause 9.16, "Data type identity", are applied with the data types of the SQL parameters $PCOMS_i$ of *UPCOMS* and the data types of the SQL parameters $POVMS_{i-1}$ of *MPDL*, respectively.

   d) The descriptor of *COMS* shall not include an indication that STATIC or CONSTRUCTOR was specified.

9) There shall exist exactly one such *COMS*.

10) *COMS* shall not be the corresponding method specification of a mutator or observer function.

    NOTE 311 — "Corresponding method specification" is defined in Subclause 11.50, "<SQL-invoked routine>".

11) For *i* varying from 2 to *N*:

    a) If $POVMS_{i-1}$ contains an <SQL parameter name> *PNM1*, then the descriptor of the *i*-th parameter of the augmented <SQL parameter declaration list> of *UPCOMS* shall include a parameter name that is equivalent to *PNM1*.

    b) If the descriptor of the *i*-th parameter of the augmented <SQL parameter declaration list> of *UPCOMS* includes a parameter name *PNM2*, then $POVMS_{i-1}$ shall contain an <SQL parameter name> that is equivalent to *PNM2*.

    c) $POVMS_{i-1}$ shall not contain <parameter mode>. A <parameter mode> IN is implicit.

    d) $POVMS_{i-1}$ shall not specify RESULT.

    e) If the <parameter type> PT_{i-1} immediately contained in $POVMS_{i-1}$ contains a <locator indication>, then the descriptor of the *i*-th parameter of the augmented <SQL parameter declaration list> of *UPCOMS* shall include a <locator indication>.

    f) If the the descriptor of the *i*-th parameter of the augmented <SQL parameter declaration list> of *UPCOMS* includes a <locator indication>, then the <parameter type> PT_{i-1} immediately contained in $POVMS_{i-1}$ shall contain a <locator indication>.

12) Let *ROVMS* be the <returns data type> of *RTC*. Let *RCOMS* be the <returns data type> of *COMS*.

    Case:

    a) If *RCOMS* is a user-defined type, then:

       i) Let a *candidate overriding method specification COVRMS* be a method specification that is included in the descriptor of a proper supertype or a proper subtype of *UDT*, such that all of the following are true:

          1) The <method name> of *COVRMS* and *MN* are equivalent.

          2) *COVRMS* and *OVMS* have the same number of SQL parameters N_i.

3) Let *PCOVRMS*$_i$, 1 (one) ≤ *i* ≤ *N*$_i$, be the *i*-th SQL parameter in the unaugmented SQL parameter declaration list of *COVRMS*. Let *POVMS*$_i$, 1 (one) ≤ *i* ≤ *N*$_i$, be the *i*-th SQL parameter in the unaugmented SQL parameter declaration list of *OVMS*.

4) For *i* varying from 1 (one) to *N*$_i$, the Syntax Rules of Subclause 9.16, "Data type identity", are applied with the declared type of *PCOVRMS*$_i$ and the declared type of *POVMS*$_i$.

ii) Let *NOVMS* be the number of candidate overriding method specifications. For *i* varying from 1 (one) to *NOVMS*, let *COVRMS*$_i$ be the *i*-th candidate overriding method specification.

Case:

1) If *COVRMS*$_i$ is included in the descriptor of a proper supertype of *D*, then *ROVMS* shall be a subtype of the <returns data type> of *COVRMS*$_i$.

2) Otherwise, *ROVMS* shall be a supertype of the <returns data type> of *COVRMS*$_i$.

b) Otherwise, the Syntax Rules of Subclause 9.16, "Data type identity", are applied with *RCOMS* and *ROVMS* as the data types.

13) Let a *conflicting overriding method specification COVMS* be an overriding method specification that is included in the descriptor of *D*, such that all of the following are true:

a) *MN* and the method name of *COVMS* are equivalent.

b) The augmented SQL parameter declaration list of *COVMS* contains *N* elements.

c) For *i* varying from 2 to *N*, the data types of the SQL parameter *POVMS*$_{i-1}$ and the SQL parameter *PCOVMS*$_{i-1}$ of *COVMS* are compatible.

There shall be no conflicting overriding method specification *COVMS*.

14) The augmented SQL parameter declaration list *ASPDL* of *OVMS* is formed from the augmented SQL parameter declaration list of *COMS* by replacing the <data type> of the first parameter (named SELF) with the <schema-resolved user-defined type name> *DN*.

15) There shall be no SQL-invoked function *F* that satisfies all the following conditions:

a) *F* is not an SQL-invoked method.

b) The <routine name> of *F* and the <routine name> *MS* have equivalent <qualified identifier>s.

c) Let *NPF* be the number of SQL parameters in *ASPDL*. *F* has *NPF* SQL parameters.

d) *D* is a subtype or supertype of the declared type of the first SQL parameter of *F*.

e) The declared type of the *i*-th SQL parameter in *ASPDL*, 2 ≤ *i* ≤ *NPF* is compatible with the declared type of *i*-th SQL parameter of *F*.

16) If the descriptor of *D* includes any method specification descriptor, then:

a) Let *M* be the number of method specification descriptors *MSD*$_i$, 1 (one) ≤ *i* ≤ *M*, included in the descriptor of *D*.

b) For i ranging from 1 (one) to M:

i) Let N_i be the number of <SQL parameter declaration>s contained in the augmented SQL parameter declaration list included in MSD_i. Let $PT_{i,j}$, 1 (one) $\leq j \leq N_i$, be the j-th <parameter type> contained in MSD_i.

ii) At least one of the following conditions shall be false:

1) The <routine name> included in MSD_i is equivalent to *MN*.

2) *ASPDL* has N_i <SQL parameter declaration>s.

3) The data type of $PT_{i,j}$, 1 (one) $\leq j \leq N_i$, is compatible with the data type of the j-th <SQL parameter declaration> of *MPDL*.

4) MSD_i does not include an indication that CONSTRUCTOR was specified.

Access Rules

None.

General Rules

1) Let *STDS* be the descriptor of D, and *DCMS* the descriptor of the corresponding original method specification *COMS*. A method specification descriptor *DOMS* is created for *OVMS*. *DOMS* includes:

a) An indication that the method specification is overriding.

b) The <method name> *MN*.

c) The <specific method name> contained in *POVMS*.

d) The augmented SQL parameter declaration list *APDL*.

e) For every parameter descriptor of a parameter of *APDL*, the locator indication of the descriptor of the corresponding parameter included in *DCMS* (if any).

f) The <language name> included in *DCMS*.

g) The <parameter style> included in *DCMS* (if any).

h) The <returns data type> contained in *POVMS*.

i) The <result cast from type> included in *DCMS* (if any).

j) The locator indication contained in the <returns clause> included in the *DCMS*.

k) The determinism indication included in *DCMS*.

l) The SQL-data access indication included in *DCMS* (if any).

m) The indication included in *DCMS* (if at all), whether the method should be invoked if any argument is the null value.

2) *DOMS* is added to *STDS*.

3) Let *N* be the number of table descriptors that include the user-defined type name of a subtype of *D*.

   For *i* varying from 1 (one) to *N*:

   a) Let TN_i be the <table name> included in the *i*-th such table descriptor.

   b) Let *M* be the number of table/method privilege descriptors that specify TN_i and the <specific method name> contained in *COMS*. For *j* varying from 1 (one) to *M*:

      i) Let $TMPD_j$ be the *j*-th such table/method privilege descriptor.

      ii) A new table/method privilege descriptor is created that specifies TN_i, the same action, grantor, and grantee, and the same grantability, and the <specific method name> contained in *OVMS*.

      iii) $TMPD_j$ is deleted.

Conformance Rules

None.

11.48 <drop method specification>

Function

Remove a method specification from a user-defined type.

Format

```
<drop method specification> ::=
    DROP <specific method specification designator> RESTRICT

<specific method specification designator> ::=
    [ INSTANCE | STATIC | CONSTRUCTOR ]
    METHOD <method name> <data type list>
```

Syntax Rules

1) Let *D* be the user-defined type identified by the <schema-resolved user-defined type name> *DN* immediately contained in the <alter type statement> containing the <drop method specification> *DORMS*. Let *DSN* be the explicit or implicit <schema name> of *DN*. Let *SMSD* be the <specific method specification designator> immediately contained in *DORMS*.

2) If *SMSD* immediately contains a <specific method name> *SMN*, then:

   a) If *SMN* contains a <schema name>, then that <schema name> shall be equivalent to *DSN*. Otherwise, the <schema name> *DSN* is implicit.

   b) The descriptor of *D* shall include a method specification descriptor *DOOMS* whose specific method name is equivalent to *SMN*.

   c) Let *PDL* be the augmented parameter list included in *DOOMS*.

   d) Let *MN* be the <method name> included in *DOOMS*.

3) If *SMSD* immediately contains a <method name> *ME*, then:

   a) If none of INSTANCE, STATIC, or CONSTRUCTOR is immediately contained in *SMSD*, then INSTANCE is implicit.

   b) The descriptor of *D* shall include a method specification descriptor *DOOMS* whose method name *MN* is equivalent to *ME*.

   c) If *SMSD* immediately contains a <data type list> *DTL*, then

      Case:

      i) If STATIC is specified, then the descriptor of *D* shall include exactly one method specification descriptor *DOOMS* that includes:

         1) An indication that the method specification is STATIC.

         2) An indication that the method specification is original.

3) An augmented parameter list *PDL* such that the declared type of its *i*-th parameter, for all *i*, is identical to the *i*-th declared type in *DTL*.

ii) If CONSTRUCTOR is specified, then the descriptor of *D* shall include exactly one method specification descriptor *DOOMS* that includes:

1) An indication that the method specification is CONSTRUCTOR.

2) An indication that the method specification is original.

3) An augmented parameter list *PDL* such that the declared type of its *i*-th parameter, for all *i* > 1 (one), is identical to the (*i*–1)-th declared type in *DTL* and the declared type of the first parameter of *PDL* is identical to *DN*.

iii) Otherwise, the descriptor of *D* shall include exactly one method specification descriptor *DOOMS* for which:

1) If *DOOMS* includes an indication that the method specification is original, then *DOOMS* shall not include an indication that the method specification is either STATIC or CONSTRUCTOR.

2) *DOOMS* includes an augmented parameter list *PDL* such that the declared type of its *i*-th parameter, for all *i* > 1 (one), is identical to the (*i*–1)-th declared type in *DTL* and the declared type of the first parameter of *PDL* is identical to *DN*.

d) If *SMSD* does not immediately contain a <data type list>, then

Case:

i) If STATIC is specified, then the descriptor of *D* shall include exactly one method specification descriptor *DOOMS* that includes an indication that the method specification is both original and STATIC.

ii) If CONSTRUCTOR is specified, then the descriptor of *D* shall include exactly one method specification descriptor *DOOMS* that includes an indications that the method specification is both original and CONSTRUCTOR.

iii) Otherwise, the descriptor of *D* shall include exactly one method specification descriptor *DOOMS* for which if *DOOMS* includes an indication that the method specification is original, then *DOOMS* shall not include an indication that the method specification is either STATIC or CONSTRUCTOR.

4) Case:

a) If *DOOMS* includes an indication that the method specification is original, then

Case:

i) If *DOOMS* includes an indication that the method specification specified STATIC, then there shall be no SQL-invoked function *F* that satisfies all of the following conditions:

1) The <routine name> of *F* and *MN* have equivalent <qualified identifier>s.

2) If *N* is the number of elements in *PDL*, then *F* has *N* SQL parameters.

3) The declared type of the first SQL parameter of *F* is *D*.

4) The declared type of the *i*-th element of *PDL*, 1 (one) ≤ *i* ≤ *N*, is compatible with the declared type of SQL parameter P_i of *F*.

5) *F* is an SQL-invoked method.

6) *F* includes an indication that STATIC is specified.

ii) If *DOOMS* includes an indication that the method specification specified CONSTRUCTOR, then there shall be no SQL-invoked function *F* that satisfies all of the following conditions:

1) The <routine name> of *F* and *MN* have equivalent <qualified identifier>s.

2) If *N* is the number of elements in *PDL*, then *F* has *N* SQL parameters.

3) The declared type of the first SQL parameter of *F* is *D*.

4) The declared type of the *i*-th element of *PDL*, 2 ≤ *i* ≤ *N*, is compatible with the declared type of SQL parameter P_i of *F*.

5) *F* is an SQL-invoked method.

6) *F* includes an indication that CONSTRUCTOR is specified.

iii) Otherwise:

1) There shall be no proper subtype *PSBD* of *D* whose descriptor includes the descriptor *DOVMS* of an overriding method specification such that all of the following is true:

A) *MN* and the <method name> included in *DOVMS* have equivalent <qualified identifier>s.

B) If *N* is the number of elements in *PDL*, then the augmented SQL parameter declaration list *APDL* included in *DOVMS* has *N* SQL parameters.

C) *PSBD* is the declared type the first SQL parameter of *APDL*.

D) The declared type of the *i*-th element of *PDL*, 2 ≤ *i* ≤ *N*, is compatible with the declared type of SQL parameter P_i of *APDL*.

2) There shall be no SQL-invoked function *F* that satisfies all of the following conditions:

A) The <routine name> of *F* and *MN* have equivalent <qualified identifier>s.

B) If *N* is the number of elements in *PDL*, then *F* has *N* SQL parameters.

C) The declared type of the first SQL parameter of *F* is *D*.

D) The declared type of the *i*-th element of *PDL*, 2 ≤ *i* ≤ *N*, is compatible with the declared type of SQL parameter P_i of *F*.

E) *F* is an SQL-invoked method.

F) *F* does not include an indication that either STATIC or CONSTRUCTOR is specified.

b) Otherwise, there shall be no SQL-invoked function *F* that satisfies all of the following conditions:

i) The <routine name> of *F* and *MN* have equivalent <qualified identifier>s.

ii) If N is the number of elements in PDL, then F has N SQL parameters.

iii) The declared type of the first SQL parameter of F is D.

iv) The declared type of the i-th element of PDL, $2 \leq i \leq N$, is compatible with the declared type of SQL parameter P_i of F.

v) F is an SQL-invoked method.

vi) F does not include an indication that either STATIC or CONSTRUCTOR is specified.

Access Rules

None.

General Rules

1) Let *STDS* be the descriptor of *D*.
2) *DOOMS* is removed from *STDS*.
3) *DOOMS* is destroyed.

Conformance Rules

None.

11.49 <drop data type statement>

Function

Destroy a user-defined type.

Format

```
<drop data type statement> ::=
    DROP TYPE <schema-resolved user-defined type name> <drop behavior>
```

Syntax Rules

1) Let *DN* be the <schema-resolved user-defined type name> and let *D* be the data type identified by *DN*. Let *SD* be any supertype of *D*.

2) Let *RD* be the reference type whose referenced type is *D*. Let *SRD* be any supertype of *RD*. Let *AD* be any collection type whose element type is *D*. Let *SAD* be any collection type whose element type is a supertype of *D* or *RD*.

3) The schema identified by the explicit or implicit schema name of *DN* shall include the descriptor of *D*.

4) If RESTRICT is specified, then:

   a) The declared type of no column, field, or attribute whose descriptor is not included in the descriptor of *D* shall be *SRD* or *SAD*.

   b) The declared type of no column, attribute, or field shall be based on *D*.

   c) *D* shall have no proper subtypes.

   d) *D* shall not be the structured type of a referenceable table.

   e) The transform descriptor included in the user-defined type descriptor of *D* shall include an empty list of transform groups.

   f) *D*, *RD*, and *AD* shall not be referenced in any of the following:

      i) The <query expression> of any view descriptor.

      ii) The <search condition> of any constraint descriptor.

      iii) A trigger action of any trigger descriptor.

      iv) A user-defined cast descriptor.

      v) A user-defined type descriptor other than that of *D* itself.

   g) There shall be no SQL-invoked routine that is not dependent on *D* and whose routine descriptor includes the descriptor of *D*, *RD*, or *AD*, or whose SQL routine body references *D*, *RD*, or *AD*.

   h) Let *R* be any SQL-invoked routine that is dependent on *D* and whose routine descriptor includes the descriptor of *D* or *RD*.

i) *R* shall not be the subject routine of any <routine invocation>, <method invocation>, <static method invocation>, or <method reference> that is contained in any of the following:

1) The SQL routine body of any routine descriptor.

2) The <query expression> of any view descriptor.

3) The <search condition> of any constraint descriptor.

4) The trigger action of any trigger descriptor.

ii) The specific name of *R* shall not be included in any user-defined cast descriptor.

iii) *R* shall not be the ordering function included in the descriptor of any user-defined type.

NOTE 312 — If CASCADE is specified, then such referenced objects will be dropped by the execution of the <revoke statement> specified in the General Rules of this Subclause.

NOTE 313 — The notion of an SQL-invoked routine being dependent on a user-defined type is defined in Subclause 4.27, "SQL-invoked routines".

NOTE 314 — The notion of one data type type being based on another data type is defined in Subclause 4.1, "Data types".

Access Rules

1) The enabled authorization identifiers shall include the <authorization identifier> that owns the schema identified by the implicit or explicit <schema name> of *D*.

General Rules

1) Let *SN* be the <specific name> of any <SQL-invoked routine> that references *D*, *RD*, or *AD* or whose routine descriptor includes the descriptor of *D*, *RD*, or *AD* and that is not dependent on *D*. The following <drop routine statement> is effectively executed for each such <SQL-invoked routine> without further Access Rule checking:

```
DROP SPECIFIC ROUTINE SN CASCADE
```

NOTE 315 — The notion of an SQL-invoked routine being dependent on a user-defined type is defined in Subclause 4.27, "SQL-invoked routines".

2) The following <drop transform statement> is effectively executed without further Access Rule checking:

```
DROP TRANSFORM ALL FOR DN CASCADE
```

NOTE 316 — This Rule should have no effect, since any external routine that depends on the transform being dropped also depends on the data type for which the transform is defined and hence should have already been dropped because of General Rule 1).

3) Let *UDCD* be the user-defined cast descriptor that references *DN* as the source data type. Let *TD* be the target data type included in *UDCD*. The following <drop user-defined cast statement> is effectively executed without further Access Rule checking:

```
DROP CAST ( DN AS TD ) CASCADE
```

4) Let *UDCD* be the user-defined cast descriptor that references *DN* as the target data type. Let *SD* be the source data type included in *UDCD*. The following <drop user-defined cast statement> is effectively executed without further Access Rule checking:

```
DROP CAST ( SD AS DN ) CASCADE
```

5) Let *UDCD* be the user-defined cast descriptor that references the reference type whose referenced type is *DN* as the source data type. Let *TD* be the target data type included in *UDCD*. The following <drop user-defined cast statement> is effectively executed without further Access Rule checking:

```
DROP CAST ( REF (DN) AS TD ) CASCADE
```

6) Let *UDCD* be the user-defined cast descriptor that references the reference type whose referenced type is *DN* as the target data type. Let *SD* be the source data type included in *UDCD*. The following <drop user-defined cast statement> is effectively executed without further Access Rule checking:

```
DROP CAST ( SD AS REF (DN) ) CASCADE
```

7) For every privilege descriptor that references *D*, the following <revoke statement> is effectively executed:

```
REVOKE PRIV ON TYPE
D FROM GRANTEE CASCADE
```

where *PRIV* and *GRANTEE* are respectively the action and grantee in the privilege descriptor.

8) The descriptor of every SQL-invoked routine that is said to be dependent on *D* is destroyed.

NOTE 317 — The notion of an SQL-invoked routine being dependent on a user-defined type is defined in Subclause 4.27, "SQL-invoked routines".

9) The descriptor of *D* is destroyed.

Conformance Rules

1) Without Feature F032, "CASCADE drop behavior", conforming SQL language shall not contain a <drop data type statement> that contains a <drop behavior> that contains CASCADE.

11.50 <SQL-invoked routine>

Function

Define an SQL-invoked routine.

Format

```
<SQL-invoked routine> ::= <schema routine>

<schema routine> ::=
    <schema procedure>
  | <schema function>

<schema procedure> ::= CREATE <SQL-invoked procedure>

<schema function> ::= CREATE <SQL-invoked function>

<SQL-invoked procedure> ::=
    PROCEDURE <schema qualified routine name> <SQL parameter declaration list>
    <routine characteristics>
    <routine body>

<SQL-invoked function> ::=
    { <function specification> | <method specification designator> } <routine body>

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

<SQL parameter declaration> ::=
    [ <parameter mode> ] [ <SQL parameter name> ] <parameter type> [ RESULT ]

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

<parameter type> ::= <data type> [ <locator indication> ]

<locator indication> ::= AS LOCATOR

<function specification> ::=
    FUNCTION <schema qualified routine name> <SQL parameter declaration list>
    <returns clause>
    <routine characteristics>
    [ <dispatch clause> ]

<method specification designator> ::=
    SPECIFIC METHOD <specific method name>
  | [ INSTANCE | STATIC | CONSTRUCTOR ] METHOD <method name> <SQL parameter declaration
list>
    [ <returns clause> ]
    FOR <schema-resolved user-defined type name>
```

```
<routine characteristics> ::= [ <routine characteristic>... ]

<routine characteristic> ::=
    <language clause>
  | <parameter style clause>
  | SPECIFIC <specific name>
  | <deterministic characteristic>
  | <SQL-data access indication>
  | <null-call clause>
  | <dynamic result sets characteristic>
  | <savepoint level indication>

<savepoint level indication> ::=
    NEW SAVEPOINT LEVEL
  | OLD SAVEPOINT LEVEL

<dynamic result sets characteristic> ::=
    DYNAMIC RESULT SETS <maximum dynamic result sets>

<parameter style clause> ::= PARAMETER STYLE <parameter style>

<dispatch clause> ::= STATIC DISPATCH

<returns clause> ::= RETURNS <returns type>

<returns type> ::=
    <returns data type> [ <result cast> ]
  | <returns table type>

<returns table type> ::= TABLE <table function column list>

<table function column list> ::=
    <left paren> <table function column list element>
    [ { <comma> <table function column list element> }... ] <right paren>

<table function column list element> ::= <column name> <data type>

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

<result cast from type> ::= <data type> [ <locator indication> ]

<returns data type> ::= <data type> [ <locator indication> ]

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

<SQL routine spec> ::= [ <rights clause> ] <SQL routine body>

<rights clause> ::=
    SQL SECURITY INVOKER
  | SQL SECURITY DEFINER

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

<external body reference> ::=
    EXTERNAL [ NAME <external routine name> ]
    [ <parameter style clause> ]
    [ <transform group specification> ]
    [ <external security clause> ]

<external security clause> ::=
    EXTERNAL SECURITY DEFINER
  | EXTERNAL SECURITY INVOKER
  | EXTERNAL SECURITY IMPLEMENTATION DEFINED

<parameter style> ::=
    SQL
  | GENERAL

<deterministic characteristic> ::=
    DETERMINISTIC
  | NOT DETERMINISTIC

<SQL-data access indication> ::=
    NO SQL
  | CONTAINS SQL
  | READS SQL DATA
  | MODIFIES SQL DATA

<null-call clause> ::=
    RETURNS NULL ON NULL INPUT
  | CALLED ON NULL INPUT

<maximum dynamic result sets> ::= <unsigned integer>

<transform group specification> ::=
    TRANSFORM GROUP { <single group specification> | <multiple group specification> }

<single group specification> ::= <group name>

<multiple group specification> ::=
    <group specification> [ { <comma> <group specification> }... ]

<group specification> ::=
    <group name> FOR TYPE <path-resolved user-defined type name>
```

Syntax Rules

1) An <SQL-invoked routine> specifies an *SQL-invoked routine*. Let *R* be the SQL-invoked routine specified by <SQL-invoked routine>.

2) If <SQL-invoked routine> immediately contains <schema routine>, then the SQL-invoked routine identified by <schema qualified routine name> is a *schema-level routine*.

3) An <SQL-invoked routine> specified as an <SQL-invoked procedure> is called an *SQL-invoked procedure*; an <SQL-invoked routine> specified as an <SQL-invoked function> is called an *SQL-invoked function*. An <SQL-invoked function> that specifies a <method specification designator> is further called an *SQL-invoked method*. An SQL-invoked method that specifies STATIC is called a *static SQL-invoked method*. An SQL-invoked method that specifies CONSTRUCTOR is called an *SQL-invoked constructor method*.

4) If <returns type> *RST* specifies TABLE, then let *TCL* be the <table function column list> contained in <returns table type>.

a) For every <column name> *CN* contained in *TCL*, *CN* shall not be equivalent to any other <column name> contained in *TCL*.

b) *RST* is equivalent to the <returns type>

```
ROW TCL MULTISET
```

5) If <SQL-invoked routine> specifies an SQL-invoked method, then

Case:

a) If a <specific method name> *SMN* is specified, then:

i) Case:

1) If *SMN* does not contain <schema name>, then

Case:

A) If the <SQL-invoked routine> is contained in a <schema definition>, then the <schema name> that is specified or implicit in the <schema definition> is implicit.

B) Otherwise, the <schema name> that is specified or implicit for the <SQL-client module definition> is implicit.

2) Otherwise, if <SQL-invoked routine> is contained in a <schema definition> then the <schema name> contained in *SMN* shall be equivalent to the specified or implicit <schema name> of the containing <schema definition>.

ii) Let *S* be the schema identified by the implicit or explicit <schema name> of *SMN*.

iii) There shall exist a method specification descriptor *DMS* included in the descriptor of a user-defined type *UDT* included in *S*, whose <specific method name> is *SMN*.

iv) Let *MN* be the number of SQL parameters in the unaugmented SQL parameter declaration list in *DMS*. *MN* is the number of SQL parameters in the unaugmented SQL parameter declaration list of *R*.

v) If *DMS* includes <result cast> *RC*, then *RC* is the <result cast> of *R*.

vi) Let *SPN* be the <specific method name> in *DMS*. *SPN* is the <specific name> of *R*.

vii) Let *NPL* be the augmented SQL parameter declaration list of *DMS*. *NPL* is the augmented SQL parameter declaration list of *R*.

viii) Let *RN* be *SN*.<method name>, where *SN* is the <schema name> of the schema that includes the descriptor of *UDT*.

b) Otherwise:

i) Let *UDTN* be the <schema-resolved user-defined type name> immediately contained in <method specification designator>. Let *UDT* be the user-defined type identified by *UDTN*.

ii) There shall exist a method specification descriptor *DMS* in the descriptor of *UDT* such that the <method name> of *DMS* is equivalent to the <method name>, *DMS* indicates STATIC if and only if the <method specification designator> specifies STATIC, *DMS* indicates CONSTRUCTOR if and only if the <method specification designator> specifies CONSTRUCTOR, and the declared type of every SQL parameter in the unaugmented SQL parameter declaration list in *DMS* is compatible with the declared type of the corresponding SQL parameter in the <SQL parameter declaration list> contained in the <method specification designator>. *DMS* identifies the corresponding method specification of the <method specification designator>.

iii) Let *MN* be the number of SQL parameters in the unaugmented SQL parameter declaration list in *DMS*.

iv) Let $PCOMS_i$, 1 (one) $\leq i \leq MN$, be the *i*-th SQL parameter in the unaugmented SQL parameter declaration list of *DMS*. Let $POVMS_i$, 1 (one) $\leq i \leq MN$, be the *i*-th SQL parameter contained in <method specification designator>.

v) For *i* varying from 1 (one) to *MN*, the <SQL parameter name>s contained in $PCOMS_i$ and $POVMS_i$ shall be equivalent.

vi) Let $PDMS_i$, 1 (one) $\leq i \leq MN$, be the declared type of the *i*-th SQL parameter in the unaugmented SQL parameter declaration list in *DMS*. Let PSM_i be the declared type of the *i*-th SQL parameter contained in <method specification designator>.

vii) With *i* ranging from 1 (one) to *MN*, the Syntax Rules of Subclause 9.16, "Data type identity", are applied with $PDMS_i$ and PSM_i.

viii) Case:

1) If <returns clause> is specified, then let *RT* be the <returns data type> of *R*. Let *RDMS* be the <returns data type> in *DMS*. The Syntax Rules of Subclause 9.16, "Data type identity", are applied with *RT* and *RDMS*.

2) Otherwise, let *RDMS* be the <returns data type> of *R*.

ix) If *DMS* includes <result cast> *RC*, then

Case:

1) If <returns clause> is specified, then <returns clause> shall contain <result cast>. Let *RDCT* be the <data type> specified in *RC*. Let *RCT* be the <data type> specified in the <result cast> contained in <returns clause>. The Syntax Rules of Subclause 9.16, "Data type identity", are applied with *RDCT* and *RCT*.

2) Otherwise, *RC* is the <result cast> of *R*.

x) Let *SPN* be the <specific method name> in *DMS*. *SPN* is the <specific name> of *R*.

xi) Let *NPL* be the augmented SQL parameter declaration list of *DMS*.

xii) Let *RN* be *SN*.<method name>, where *SN* is the <schema name> of the schema that includes the descriptor of *UDT*.

6) If <SQL-invoked routine> specifies an SQL-invoked procedure or an SQL-invoked regular function, then:

a) <routine characteristics> shall contain at most one <language clause>, at most one <parameter style clause>, at most one <specific name>, at most one <deterministic characteristic>, at most one <SQL-

data access indication>, at most one <null-call clause>, and at most one <dynamic result sets characteristic>.

b) <parameter style clause> shall not be specified both in <routine characteristics> and in <external body reference>.

c) The <routine characteristics> of a <function specification> shall not contain a <dynamic result sets characteristic>.

d) If <dynamic result sets characteristic> is not specified, then DYNAMIC RESULT SETS 0 (zero) is implicit.

e) If <deterministic characteristic> is not specified, then NOT DETERMINISTIC is implicit.

f) Case:

   i) If PROCEDURE is specified, then:

      1) <null-call clause> shall not be specified.

      2) <routine characteristics> shall not contain more than one <savepoint level indication>.

   ii) Otherwise, if <null-call clause> is not specified, then CALLED ON NULL INPUT is implicit.

g) <SQL-data access indication> shall be specified.

h) If <language clause> is not specified, then LANGUAGE SQL is implicit.

i) An <SQL-invoked routine> that specifies or implies LANGUAGE SQL is called an *SQL routine*; an <SQL-invoked routine> that does not specify LANGUAGE SQL is called an *external routine*.

j) If <savepoint level indication> is specified, then PROCEDURE shall be specified.

k) If PROCEDURE is specified and <savepoint level indication> is not specified, then OLD SAVEPOINT LEVEL is implicit.

l) If NEW SAVEPOINT LEVEL is specified, then MODIFIES SQL DATA shall be specified.

m) If *R* is an SQL routine, then:

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

   ii) <SQL-data access indication> shall not specify NO SQL.

   iii) <parameter style clause> shall not be specified.

n) An array-returning external function is an SQL-invoked function that is an external routine and that satisfies one of the following conditions:

   i) A <result cast from type> is specified that simply contains an <array type> and does not contain a <locator indication>.

   ii) A <result cast from type> is not specified and <returns data type> simply contains an <array type> and does not contain a <locator indication>.

o) A multiset-returning external function is an SQL-invoked function that is an external routine and that satisfies one of the following conditions:

i) A <result cast from type> is specified that simply contains a <multiset type> and does not contain a <locator indication>.

ii) A <result cast from type> is not specified and <returns data type> simply contains a <multiset type> and does not contain a <locator indication>.

p) Let *RN* be the <schema qualified routine name> of *R*.

q) If <SQL-invoked routine> is contained in a <schema definition> and *RN* contains a <schema name> *SN*, then *SN* shall be equivalent to the specified or implicit <schema name> of the containing <schema definition>. Let *S* be the SQL-schema identified by *SN*.

r) Case:

i) If *R* is an SQL-invoked regular function and the <SQL parameter declaration list> contains an <SQL parameter declaration> that specifies a <data type> that is one of:

1) A user-defined type.

2) A collection type whose element type is a user-defined type.

3) A collection type whose element type is a reference type.

4) A reference type.

then <dispatch clause> shall be specified.

ii) Otherwise, <dispatch clause> shall not be specified.

s) If <specific name> is not specified, then an implementation-dependent <specific name> whose <schema name> is the equivalent to the <schema name> of *S* is implicit.

t) If <specific name> contains a <schema name>, then that <schema name> shall be equivalent to the <schema name> of *S*. If <specific name> does not contain a <schema name>, then the <schema name> of *S* is implicit.

u) The schema identified by the explicit or implicit <schema name> of the <specific name> shall not include a routine descriptor whose specific name is equivalent to <specific name> or a user-defined type descriptor that includes a method specification descriptor whose specific name is equivalent to <specific name>.

v) If <returns data type> *RT* simply contains <locator indication>, then:

i) *R* shall be an external routine.

ii) *RT* shall be either binary large object type, character large object type, array type, multiset type, or user-defined type.

iii) <result cast> shall not be specified.

w) If <result cast from type> *RCT* simply contains <locator indication>, then:

i) *R* shall be an external routine.

ii) *RCT* shall be either binary large object type, character large object type, array type, multiset type, or user-defined type.

x) If *R* is an external routine, then:

i) If <parameter style> is not specified, then PARAMETER STYLE SQL is implicit.

ii) If *R* is an array-returning external function or a multiset-returning external function, then PARAMETER STYLE SQL shall be either specified or implied.

iii) Case:

1) If <transform group specification> is not specified, then a <multiple group specification> with a <group specification> *GS* for each <SQL parameter declaration> contained in <SQL parameter declaration list> whose <parameter type> *UDT* identifies a user-defined type with no <locator indication> is implicit. The <group name> of *GS* is implementation-defined and its <path-resolved user-defined type name> is *UDT*.

2) If <single group specification> with a <group name> *GN* is specified, then <transform group specification> is equivalent to a <transform group specification> that contains a <multiple group specification> that contains a <group specification> *GS* for each <SQL parameter declaration> contained in <SQL parameter declaration list> whose <parameter type> *UDT* identifies a user-defined type with no <locator indication>. The <group name> of *GS* is *GN* and its <path-resolved user-defined type name> is *UDT*.

3) Otherwise, <multiple group specification> is extended with a <group specification> *GS* for each <SQL parameter declaration> contained in <SQL parameter declaration list> whose <parameter type> *UDT* identifies a user-defined type with no <locator indication> and no equivalent of *UDT* is contained in any <group specification> contained in <multiple group specification>. The <group name> of *GS* is implementation-defined and its <path-resolved user-defined type name> is *UDT*.

iv) If a <result cast> is specified, then let *V* 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 6.12, "<cast specification>":

```
CAST ( V AS RT )
```

y) Let *NPL* be the <SQL parameter declaration list> contained in the <SQL-invoked routine>.

7) *NPL* specifies the list of SQL parameters of *R*. Each SQL parameter of *R* is specified by an <SQL parameter declaration>. If <SQL parameter name> is specified, then that SQL parameter of *R* is identified by an SQL parameter name.

8) *NPL* shall specify at most one <SQL parameter declaration> that specifies RESULT.

9) If *R* is an SQL-invoked function, then no <SQL parameter declaration> in *NPL* shall contain a <parameter mode>.

10) If *R* is an SQL routine, then every <SQL parameter declaration> in *NPL* shall contain an <SQL parameter name>.

11) No two <SQL parameter name>s contained in *NPL* shall be equivalent.

12) Let *N* and *PN* be the number of <SQL parameter declaration>s contained in *NPL*. For every <SQL parameter declaration> PD_i, 1 (one) $\leq i \leq N$:

a) <parameter type> PT_i immediately contained in PD_i shall not specify ROW.

b) If PT_i simply contains <locator indication>, then:

   i) *R* shall be an external routine.

   ii) PT_i shall specify either binary large object type, character large object type, array type, multiset type, or user-defined type.

c) If PD_i immediately contains RESULT, then:

   i) *R* shall be an SQL-invoked function.

   ii) PT_i shall specify a structured type *ST*. Let *STN* be the <user-defined type name> that identifies *ST*.

   iii) The <returns data type> shall specify *STN*.

   iv) *R* is a type-preserving function and PD_i specifies the result SQL parameter of *R*.

d) If PD_i does not contain a <parameter mode>, then a <parameter mode> that specifies IN is implicit.

e) Let P_i be the *i*-th SQL parameter.

   Case:

   i) If the <parameter mode> specifies IN, then P_i is an input SQL parameter.

   ii) If the <parameter mode> specifies OUT, then P_i is an output SQL parameter.

   iii) If the <parameter mode> specifies INOUT, then P_i is both an input SQL parameter and an output SQL parameter.

13) The scope of *RN* is the <routine body> of *R*.

14) The scope of an <SQL parameter name> contained in *NPL* is the <routine body> *RB* of the <SQL-invoked procedure> or <SQL-invoked function> that contains *NPL*.

15) An <SQL-invoked routine> shall not contain a <host parameter name>, a <dynamic parameter specification>, or an <embedded variable name>.

16) Case:

a) If *R* is an SQL-invoked procedure, then *S* shall not include another SQL-invoked procedure whose <schema qualified routine name> is equivalent to *RN* and whose number of SQL parameters is *PN*.

b) Otherwise:

   i) Case:

      1) If *R* is a static SQL-invoked method, then let *SCR* be the set containing every static SQL-invoked method of type *UDT*, including *R*, whose <schema qualified routine name> is equivalent to *RN* and whose number of SQL parameters is *PN*.

2) If *R* is an SQL-invoked constructor method, then let *SCR* be the set containing every SQL-invoked constructor method of type *UDT*, including *R*, whose <schema qualified routine name> is equivalent to *RN* and whose number of SQL parameters is *PN*.

3) Otherwise, let *SCR* be the set containing every SQL-invoked function in *S* that is neither a static SQL-invoked method nor an SQL-invoked constructor method, including *R*, whose <schema qualified routine name> is equivalent to *RN* and whose number of SQL parameters is *PN*.

ii) Let *AL* be an <SQL argument list> constructed from a list of arbitrarily-selected values in which the declared type of every value A_i in *AL* is compatible with the declared type of the corresponding SQL parameter P_i of *R*.

iii) For every A_i, eliminate from *SCR* every SQL-invoked routine *SIR* for which the type designator of the declared type of the SQL parameter P_i of *SIR* is not in the type precedence list of the declared type of A_i.

iv) Let *SR* be the set of subject routines defined by applying the Syntax Rules of Subclause 9.4, "Subject routine determination", with the set of SQL-invoked routines as *SCR* and <SQL argument list> as *AL*. There shall be exactly one subject routine in *SR*.

17) If *R* is an SQL-invoked method but not a static SQL-invoked method, then the first SQL parameter of *NPL* is called the *subject parameter* of *R*.

18) If *R* is an SQL-invoked regular function *F* whose first SQL parameter has a declared type that is a user-defined type, then:

a) Let *UDT* be the declared type of the first SQL parameter of *F*.

b) Let *DMS* be a method specification descriptor of an instance method in the descriptor of *UDT* such that:

i) The <schema qualified routine name> of *F* and the <routine name> of *DMS* have equivalent <qualified identifier>s.

ii) *F* and the augmented SQL parameter declaration list of *DMS* have the same number of SQL parameters.

c) Let $PDMS_i$, 1 (one) $\leq i \leq PN$, be the declared type of the *i*-th SQL parameter in the unaugmented SQL parameter declaration list in *DMS* and let PMS_i be the declared type of the *i*-th SQL parameter contained in <function specification>.

d) One of the following conditions shall be false:

i) The declared type of $PDMS_i$, 1 (one) $\leq i \leq N$ is compatible with the declared type of SQL parameter PMS_{i+1}.

ii) *UDT* is a subtype or a supertype of the declared type of PMS_1.

19) If *R* is an SQL routine, then:

a) <SQL routine spec> shall be specified.

b) If <rights clause> is not specified, then SQL SECURITY DEFINER is implicit.

c) If READS SQL DATA is specified, then it is implementation-defined whether the <SQL routine body> shall not contain an <SQL procedure statement> *S* that satisfies at least one of the following:

   i) *S* is an <SQL data change statement>.

   ii) *S* contains a <routine invocation> whose subject routine is an SQL-invoked routine that possibly modifies SQL-data.

   iii) *S* contains an <SQL procedure statement> that is an <SQL data change statement>.

d) If CONTAINS SQL is specified, then it is implementation-defined whether the <SQL routine body> shall not contain an <SQL procedure statement> *S* that satisfies at least one of the following:

   i) *S* is an <SQL data statement> other than <free locator statement> and <hold locator statement>.

   ii) *S* contains a <routine invocation> whose subject routine is an SQL-invoked routine that possibly modifies SQL-data or possibly reads SQL-data.

   iii) *S* contains an <SQL procedure statement> that is an <SQL data statement> other than <free locator statement> and <hold locator statement>.

e) If DETERMINISTIC is specified, then it is implementation-defined whether the <SQL routine body> shall not contain an <SQL procedure statement> that is possibly non-deterministic.

f) It is implementation-defined whether the <SQL routine body> shall not contain an <SQL connection statement>, an <SQL schema statement>, an <SQL dynamic statement>, or an <SQL transaction statement> other than a <savepoint statement>, <release savepoint statement>, or a <rollback statement> that specifies a <savepoint clause>.

   NOTE 318 — Conforming SQL language shall not contain an <SQL connection statement> or an <SQL transaction statement> other than a <savepoint statement>, a <release savepoint statement>, or a <rollback statement> that specifies a <savepoint clause>, but an implementation is not required to treat this as a syntax error.

g) An <SQL routine body> shall not immediately contain an <SQL procedure statement> that simply contains a <schema definition>.

20) If *R* is an external routine, then:

a) <SQL routine spec> shall not be specified.

b) If <external security clause> is not specified, then EXTERNAL SECURITY IMPLEMENTATION DEFINED is implicit.

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

d) If PARAMETER STYLE SQL is specified, then:

   i) Case:

      1) If *R* is an array-returning external function or a multiset-returning external function with the element type being a row type, then let *FRN* be the degree of the element type.

      2) Otherwise, let *FRN* be 1 (one).

ii) If *R* is an array-returning external function or a multiset-returning external function, then let *AREF* be *FRN*+6. Otherwise, let *AREF* be *FRN*+4.

iii) If *R* is an SQL-invoked function, then let the *effective SQL parameter list* be a list of *PN*+*FRN*+*N*+*AREF* SQL parameters, as follows:

1) For *i* ranging from 1 (one) to *PN*, the *i*-th effective SQL parameter list entry is defined as follows.

Case:

A) If the <parameter type> T_i simply contained in the *i*-th <SQL parameter declaration> contains <locator indication>, then the *i*-th effective SQL parameter list entry is the *i*-th <SQL parameter declaration> with the <parameter type> replaced by INTEGER.

B) If the <parameter type> T_i immediately contained in the *i*-th <SQL parameter declaration> is a <path-resolved user-defined type name> without a <locator indication>, then:

I) Case:

1) If *R* is an SQL-invoked method that is an overriding method, then the Syntax Rules of Subclause 9.18, "Determination of a from-sql function for an overriding method", are applied with *R* and *i* as *ROUTINE* and *POSITION*, respectively. There shall be an applicable from-sql function FSF_i.

2) Otherwise, the Syntax Rules of Subclause 9.17, "Determination of a from-sql function", are applied with the data type identified by T_i, and the <group name> contained in the <group specification> that contains T_i as *TYPE* and *GROUP*, respectively. There shall be an applicable from-sql function FSF_i.

II) FSF_i is called the *from-sql function associated with the i-th SQL parameter*.

III) The *i*-th effective SQL parameter list entry is the *i*-th <SQL parameter declaration> with the <parameter type> replaced by the <returns data type> of FSF_i.

C) Otherwise, the *i*-th effective SQL parameter list entry is the *i*-th <SQL parameter declaration>.

2) Case:

A) If *FRN* is 1 (one), then effective SQL parameter list entry *PN*+*FRN* has <parameter mode> OUT; its <parameter type> *PT* is defined as follows:

I) If <result cast> is specified, then let *RT* be <result cast from type>; otherwise, let *RT* be <returns data type>.

II) Case:

1) If *RT* simply contains <locator indication>, then *PT* is INTEGER.

2) If *RT* specifies a <path-resolved user-defined type name> without a <locator indication>, then:

a) Case:

i) If *R* is an SQL-invoked method that is an overriding method, then the Syntax Rules of Subclause 9.20, "Determination of a to-sql function for an overriding method", are applied with *R* as *ROUTINE*. There shall be an applicable to-sql function *TSF*.

ii) Otherwise, the Syntax Rules of Subclause 9.19, "Determination of a to-sql function", are applied with the data type identified by *RT* and the <group name> contained in the <group specification> that contains *RT* as *TYPE* and *GROUP*, respectively. There shall be an applicable to-sql function *TSF*.

b) *TSF* is called the *to-sql function* associated with the result.

c) Case:

i) If *TSF* is an SQL-invoked method, then *PT* is the <parameter type> of the second SQL parameter of *TSF*.

ii) Otherwise, *PT* is the <parameter type> of the first SQL parameter of *TSF*.

3) If *R* is an array-returning external function or a multiset-returning external function, then let *PT* be the element type of *RT*.

4) Otherwise, *PT* is *RT*.

B) Otherwise, for *i* ranging from *PN*+1 to *PN*+*FRN*, the *i*-th effective SQL parameter list entry is defined as follows.

Case:

I) Its <parameter mode> is OUT.

II) Let RFT_{i-PN} be the data type of the *i-PN*-th field of the element type of the <returns data type>. The <parameter type> PT_i of the *i*-th effective SQL parameter list entry is determined as follows:

1) If RFT_{i-PN} specifies a <path-resolved user-defined type name>, then:

a) Case:

i) If *R* is an SQL-invoked method that is an overriding method, then the Syntax Rules of Subclause 9.20, "Determination of a to-sql function for an overriding method", are applied with *R* as *ROUTINE*. There shall be an applicable to-sql function *TSF*.

ii) Otherwise, the Syntax Rules of Subclause 9.19, "Determination of a to-sql function", are applied with the data type identified by RFT_{i-PN} and the <group name> contained in the <group specification> that contains RFT_{i-PN} as *TYPE* and *GROUP*, respectively. There shall be an applicable to-sql function *TSF*.

b) *TSF* is called the *to-sql function* associated with RFT_{i-PN}.

c) Case:

i) If *TSF* is an SQL-invoked method, then PT_i is the <parameter type> of the second SQL parameter of *TSF*.

ii) Otherwise, PT_i is the <parameter type> of the first SQL parameter of *TSF*.

2) Otherwise, PT_i is RFT_{i-PN}.

3) Effective SQL parameter list entries (*PN*+*FRN*)+1 to (*PN*+*FRN*)+*N*+*FRN* are *N*+*FRN* occurrences of SQL parameters of an implementation-defined <data type> that is an exact numeric type with scale 0 (zero). For *i* ranging from (*PN*+*FRN*)+1 to (*PN*+*FRN*)+*N*+*FRN*, the <parameter mode> for the *i*-th such effective SQL parameter is the same as that of the *i*–*FRN*–*PN*-th effective SQL parameter.

4) Effective SQL parameter list entry (*PN*+*FRN*)+(*N*+*FRN*)+1 is an SQL parameter of a <data type> that is character string of length 5 and the character set specified for SQLSTATE values, with <parameter mode> INOUT.

NOTE 319 — The character set specified for SQLSTATE values is defined in Subclause 23.1, "SQLSTATE".

5) Effective SQL parameter list entry (*PN*+*FRN*)+(*N*+*FRN*)+2 is an SQL parameter of a <data type> that is character string of implementation-defined length and character set SQL_TEXT with <parameter mode> IN.

6) Effective SQL parameter list entry (*PN*+*FRN*)+(*N*+*FRN*)+3 is an SQL parameter of a <data type> that is character string of implementation-defined length and character set SQL_TEXT with <parameter mode> IN.

7) Effective SQL parameter list entry (*PN*+*FRN*)+(*N*+*FRN*)+4 is an SQL parameter of a <data type> that is character string of implementation-defined length and character set SQL_TEXT with <parameter mode> INOUT.

8) If *R* is an array-returning external function or a multiset-returning external function, then:

A) Effective SQL parameter type list entry (*PN*+*FRN*)+(*N*+*FRN*)+5 is an SQL parameter whose <data type> is character string of implementation-defined length and character set SQL_TEXT with <parameter mode> INOUT.

B) Effective SQL parameter type list entry (*PN*+*FRN*)+(*N*+*FRN*)+6 is an SQL parameter whose <data type> is an exact numeric type with scale 0 (zero) and with <parameter mode> IN.

iv) If *R* is an SQL-invoked procedure, then let the *effective SQL parameter list* be a list of *PN*+*N*+4 SQL parameters, as follows:

1) For *i* ranging from 1 (one) to *PN*, the *i*-th effective SQL parameter list entry is defined as follows.

Case:

A) If the <parameter type> T_i simply contained in the *i*-th <SQL parameter declaration> simply contains <locator indication>, then the *i*-th effective SQL parameter list entry is the i-th <SQL parameter declaration> with the <parameter type> replaced by INTEGER.

B) If the <parameter type> T_i simply contained in the i-th <SQL parameter declaration> is a <path-resolved user-defined type name> without a <locator indication>, then:

I) Case:

1) If the <parameter mode> immediately contained in the i-th <SQL parameter declaration> is IN, then:

a) The Syntax Rules of Subclause 9.17, "Determination of a from-sql function", are applied with the data type identified by T_i and the <group name> contained in the <group specification> that contains T_i as *TYPE* and *GROUP*, respectively. There shall be an applicable from-sql function FSF_i. FSF_i is called the *from-sql function* associated with the i-th SQL parameter.

b) The i-th effective SQL parameter list entry is the i-th <SQL parameter declaration> with the <parameter type> replaced by the <returns data type> of FSF_i.

2) If the <parameter mode> immediately contained in the i-th <SQL parameter declaration> is OUT, then:

a) The Syntax Rules of Subclause 9.19, "Determination of a to-sql function", are applied with the data type identified by T_i and the <group name> contained in the <group specification> that contains T_i as *TYPE* and *GROUP*, respectively. There shall be an applicable to-sql function TSF_i. TSF_i is called the *to-sql function associated with i-th SQL parameter.*

b) The i-th effective SQL parameter list entry is the i-th <SQL parameter declaration> with the <parameter type> replaced by

Case:

i) If TSF_i is an SQL-invoked method, then the <parameter type> of the second SQL parameter of TSF_i.

ii) Otherwise, the <parameter type> of the first SQL parameter of TSF_i.

3) Otherwise:

a) The Syntax Rules of Subclause 9.17, "Determination of a from-sql function", are applied with the data type identified by T_i and the <group name> contained in the <group specification> that contains T_i as *TYPE* and *GROUP*, respectively. There shall be an applicable from-sql function FSF_i. FSF_i is called the *from-sql function associated with the i-th SQL parameter.*

b) The Syntax Rules of Subclause 9.19, "Determination of a to-sql function", are applied with the data type identified by T_i and the <group name> contained in the <group specification> that contains T_i as *TYPE* and *GROUP*, respectively. There shall be an applicable to-sql function TSF_i. TSF_i is called the *to-sql function associated with the i-th SQL parameter*.

c) The *i*-th effective SQL parameter list entry is the *i*-th <SQL parameter declaration> with the <parameter type> replaced by the <returns data type> of FSF_i.

C) Otherwise, the *i*-th effective SQL parameter list entry is the *i*-th <SQL parameter declaration>.

2) Effective SQL parameter list entries *PN*+1 to *PN*+*N* are *N* occurrences of an SQL parameter of an implementation-defined <data type> that is an exact numeric type with scale 0. The <parameter mode> for the *i*-th such effective SQL parameter is the same as that of the *i*–*PN*-th effective SQL parameter.

3) Effective SQL parameter list entry (*PN*+*N*)+1 is an SQL parameter of a <data type> that is character string of length 5 and character set SQL_TEXT with <parameter mode> INOUT.

4) Effective SQL parameter list entry (*PN*+*N*)+2 is an SQL parameter of a <data type> that is character string of implementation-defined length and character set SQL_TEXT with <parameter mode> IN.

5) Effective SQL parameter list entry (*PN*+*N*)+3 is an SQL parameter of a <data type> that is character string of implementation-defined length and character set SQL_TEXT with <parameter mode> IN.

6) Effective SQL parameter list entry (*PN*+*N*)+4 is an SQL parameter of a <data type> that is character string of implementation-defined length and character set SQL_TEXT with <parameter mode> INOUT.

e) If PARAMETER STYLE GENERAL is specified, then let the *effective SQL parameter list* be a list of *PN* parameters such that, for *i* ranging from 1 (one) to *PN*, the *i*-th effective SQL parameter list entry is defined as follows.

Case:

i) If the <parameter type> T_i simply contained in the *i*-th <SQL parameter declaration> simply contains <locator indication>, then the *i*-th effective SQL parameter list entry is the *i*-th <SQL parameter declaration> with the <parameter type> replaced by INTEGER.

ii) If the <parameter type> T_i simply contained in the *i*-th <SQL parameter declaration> is a <path-resolved user-defined type name> without a <locator indication>, then:

1) Case:

A) If the <parameter mode> immediately contained in the *i*-th <SQL parameter declaration> is IN, then:

I) The Syntax Rules of Subclause 9.17, "Determination of a from-sql function", are applied with the data type identified by T_i and the <group name> contained in the <group specification> that contains T_i as *TYPE* and *GROUP*, respectively. There shall be an applicable from-sql function FSF_i. FSF_i is called the *from-sql function associated with the i-th SQL parameter.*

II) The *i*-th effective SQL parameter list entry is the *i*-th <SQL parameter declaration> with the <parameter type> replaced by the <returns data type> of FSF_i.

B) If the <parameter mode> immediately contained in the *i*-th <SQL parameter declaration> is OUT, then:

I) The Syntax Rules of Subclause 9.19, "Determination of a to-sql function", are applied with the data type identified by T_i and the <group name> contained in the <group specification> that contains T_i as *TYPE* and *GROUP*, respectively. There shall be an applicable to-sql function TSF_i. TSF_i is called the *to-sql function associated with the i-th SQL parameter.*

II) The *i*-th effective SQL parameter list entry is the *i*-th <SQL parameter declaration> with the <parameter type> replaced by

Case:

1) If TSF_i is an SQL-invoked method, then the <parameter type> of the second SQL parameter of TSF_i.

2) Otherwise, the <parameter type> of the first SQL parameter of TSF_i.

C) Otherwise:

I) The Syntax Rules of Subclause 9.17, "Determination of a from-sql function", are applied with the data type identified by T_i and the <group name> contained in the <group specification> that contains T_i as *TYPE* and *GROUP*, respectively. There shall be an applicable from-sql function FSF_i. FSF_i is called the *from-sql function associated with the i-th SQL parameter.*

II) The Syntax Rules of Subclause 9.19, "Determination of a to-sql function", are applied with the data type identified by T_i and the <group name> contained in the <group specification> that contains T_i as *TYPE* and *GROUP*, respectively. There shall be an applicable to-sql function TSF_i. TSF_i is called the *to-sql function associated with the i-th SQL parameter.*

III) The *i*-th effective SQL parameter list entry is the *i*-th <SQL parameter declaration> with the <parameter type> replaced by the <returns data type> of FSF_i.

iii) Otherwise, the *i*-th effective SQL parameter list entry is the *i*-th <SQL parameter declaration>.

NOTE 320 — If the SQL-invoked routine is an SQL-invoked function, then the value returned from the external routine is passed to the SQL-implementation in an implementation-dependent manner. An SQL parameter is not used for this purpose.

f) Depending on whether the <language clause> specifies ADA, C, COBOL, FORTRAN, M, PASCAL, or PLI, let the *operative data type correspondences table* be Table 16, "Data type correspondences for Ada", Table 17, "Data type correspondences for C", Table 18, "Data type correspondences for COBOL", Table 19, "Data type correspondences for Fortran", Table 20, "Data type correspondences for M", Table 21, "Data type correspondences for Pascal", or Table 22, "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) Any <data type> in an effective SQL parameter list entry 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".

21) Case:

a) If <method specification designator> is specified, then:

i) *R* is deterministic if *DMS* indicates that the method is deterministic; otherwise, *R* is possibly non-deterministic.

ii) *R* possibly modifies SQL-data if the SQL-data access indication of *DMS* indicates that the method possibly modifies SQL-data. *R* possibly reads SQL-data if the SQL-data access indication of *DMS* indicates that the method possibly reads SQL-data. *R* possibly contains SQL if the SQL-data access indication of *DMS* indicates that the method possibly contains SQL. Otherwise, *R* does not possibly contain SQL.

b) Otherwise:

i) If DETERMINISTIC is specified, then *R* is *deterministic*; otherwise, it is *possibly non-deterministic.*

ii) An <SQL-invoked routine> *possibly modifies SQL-data* if and only if <SQL-data access indication> specifies MODIFIES SQL DATA.

iii) An <SQL-invoked routine> *possibly reads SQL-data* if and only if <SQL-data access indication> specifies READS SQL DATA.

iv) An <SQL-invoked routine> *possibly contains SQL* if and only if <SQL-data access indication> specifies CONTAINS SQL.

v) An <SQL-invoked routine> *does not possibly contain SQL* if and only if <SQL-data access indication> specifies NO SQL.

22) If *R* is a schema-level routine, then let the containing schema be the schema identified by the <schema name> explicitly or implicitly contained in <schema qualified routine name>.

23) If the <SQL-invoked routine> is contained in a <schema definition>, then let *A* be the explicit or implicit <authorization identifier> of the <schema definition>; otherwise, let *A* be the <authorization identifier> that owns the schema identified by the explicit or implicit <schema name> of the <schema qualified routine name>.

Access Rules

1) If an <SQL-invoked routine> is contained in an <SQL-client module definition> *M* with no intervening <schema definition>, then the enabled authorization identifiers shall include the <authorization identifier> that owns *S*.

2) If *R* is an external routine and if any of its SQL parameters have an associated from-sql function or a to-sql function, or if *R* has a to-sql function associated with the result, then

Case:

a) If <SQL-invoked routine> is contained, without an intervening <SQL routine spec> that specifies SQL SECURITY INVOKER, in an <SQL schema statement>, then the applicable privileges of the <authorization identifier> that owns the containing schema shall include EXECUTE on all from-sql functions (if any) and on all to-sql functions (if any) associated with the SQL parameters and on the to-sql function associated with the result (if any).

b) Otherwise, the current privileges shall include EXECUTE on all from-sql functions (if any) and on all to-sql functions (if any) associated with the SQL parameters and on the to-sql function associated with the result (if any).

General Rules

1) If *R* is a schema-level routine, then a privilege descriptor is created that defines the EXECUTE privilege on *R* to the <authorization identifier> that owns the schema that includes *R*. The grantor for the privilege descriptor is set to the special grantor value "_SYSTEM". This privilege is grantable if and only if one of the following is satisfied:

   a) *R* is an SQL routine and all of the privileges necessary for the <authorization identifier> to successfully execute the <SQL procedure statement> contained in the <routine body> are grantable. The necessary privileges include the EXECUTE privilege on every subject routine of every <routine invocation> contained in the <SQL procedure statement>.

   b) *R* is an SQL routine and SQL SECURITY INVOKER is specified.

   c) *R* is an external routine.

2) Case:

   a) If <SQL-invoked routine> is contained in a <schema definition>, then let *DP* be the SQL-path of that <schema definition>.

   b) If <SQL-invoked routine> is contained in a <preparable statement> or in a <direct SQL statement>, then let *DP* be the SQL-path of the current SQL-session.

   c) Otherwise, let *DP* be the SQL-path of the <SQL-client module definition> that contains <SQL-invoked routine>.

3) If <method specification designator> is not specified, then a routine descriptor is created that describes the SQL-invoked routine being defined:

   a) The routine name included in the routine descriptor is <schema qualified routine name>.

   b) The specific name included in the routine descriptor is <specific name>.

   c) The routine descriptor includes, for each SQL parameter in *NPL*, the name, declared type, ordinal position, an indication of whether the SQL parameter is input, output, or both, and an indication of whether the SQL parameter is a RESULT SQL parameter.

   d) If the SQL-invoked routine is an SQL-invoked procedure, then the explicit or implicit value of <maximum dynamic result sets>.

e) The routine descriptor includes an indication of whether the SQL-invoked routine is an SQL-invoked function or an SQL-invoked procedure.

f) If the SQL-invoked routine is an SQL-invoked function, then:

  i) The routine descriptor includes an indication that the SQL-invoked function is not an SQL-invoked method.

  ii) The routine descriptor includes the data type in the <returns data type>. If the <returns data type> simply contains <locator indication>, then the routine descriptor includes an indication that the return value is a locator.

  iii) The SQL-invoked routine descriptor includes an indication of whether the SQL-invoked routine is a null-call function.

g) If the SQL-invoked routine is a type-preserving function, then the routine descriptor includes an indication that the SQL-invoked routine is a type-preserving function.

h) The name of the language in which the body of the SQL-invoked routine was written is the <language name> contained in the <language clause>.

i) If the SQL-invoked routine is an SQL routine, then the SQL routine body of the routine descriptor is the <SQL routine body>.

j) If the SQL-invoked routine is an SQL-invoked function or NEW SAVEPOINT LEVEL is specified, then an indication that a new savepoint level is to be established whenever the routine is invoked.

NOTE 321 — The use of savepoint levels is dependent on Feature T272, "Enhanced savepoint management".

k) Case:

  i) If SQL SECURITY INVOKER is specified, then the SQL security characteristic in the routine descriptor is INVOKER.

  ii) Otherwise, the SQL security characteristic in the routine descriptor is DEFINER.

l) If the SQL-invoked routine is an external routine, then:

  i) The external name of the routine descriptor is <external routine name>.

  ii) The routine descriptor includes an indication of whether the *parameter passing style* is PARAMETER STYLE SQL or PARAMETER STYLE GENERAL.

m) The SQL-invoked routine descriptor includes an indication of whether the SQL-invoked routine is DETERMINISTIC or NOT DETERMINISTIC.

n) The SQL-invoked routine descriptor includes an indication of whether the SQL-invoked routine does not possibly contain SQL, possibly contains SQL, possibly reads SQL-data, or possibly modifies SQL-data.

o) If the SQL-invoked routine specifies a <result cast>, then the routine descriptor includes an indication that the SQL-invoked routine specifies a <result cast> and the <data type> specified in the <result cast>. If <result cast> contains <locator indication>, then the routine descriptor includes an indication that the <data type> specified in the <result cast> has a locator indication.

p) For every SQL parameter that has an associated from-sql function *FSF*, the routine descriptor includes the specific name of *FSF*.

q) For every SQL parameter that has an associated to-sql function *TSF*, the routine descriptor includes the specific name of *TSF*.

r) If *R* is an external function and if *R* has a to-sql function associated with its result *TRF*, then the routine descriptor includes the specific name of *TRF*.

s) For every SQL parameter whose <SQL parameter declaration> contains <locator indication>, the routine descriptor includes an indication that the SQL parameter is a locator parameter.

t) The routine authorization identifier is the <authorization identifier> that owns *S*.

u) The routine SQL-path is *DP*.

NOTE 322 — The routine SQL-path is used to set the routine SQL-path of the current SQL-session when *R* is invoked. The routine SQL-path of the current SQL-session is used by the Syntax Rules of Subclause 10.4, "<routine invocation>", to define the subject routines of <routine invocation>s contained in *R*. The same routine SQL-path is used whenever *R* is invoked.

v) An indication that the routine is a schema-level routine.

w) An indication of whether the SQL-invoked routine is dependent on a user-defined type.

NOTE 323 — The notion of an SQL-invoked routine being dependent on a user-defined type is defined in Subclause 4.27, "SQL-invoked routines".

4) If <method specification designator> is specified, then let *DMS* be the descriptor of the corresponding method specification. A routine descriptor is created that describes the SQL-invoked routine being defined.

a) The routine name included in the routine descriptor is *RN*.

b) The specific name included in the routine descriptor is <specific name>.

c) The routine descriptor includes, for each SQL parameter in *NPL*, the name, data type, ordinal position, an indication of whether the SQL parameter is input, output, or both, and an indication of whether the SQL parameter is a RESULT SQL parameter.

d) The routine descriptor includes an indication that the SQL-invoked routine is an SQL-invoked function that is an SQL-invoked method, an indication of the user-defined type *UDT*, and an indication of whether STATIC or CONSTRUCTOR was specified.

e) If the SQL-invoked routine is a type-preserving function, then the routine descriptor includes an indication that the SQL-invoked routine is a type-preserving function.

f) If the SQL-invoked routine is a mutator function, then the routine descriptor includes an indication that the SQL-invoked routine is a mutator function.

g) The routine descriptor includes the data type in the <returns data type>.

h) The name of the language in which the body of the SQL-invoked routine was written is the <language name> contained in the <language clause> in *DMS*.

i) If the SQL-invoked routine is an SQL routine, then the SQL routine body of the routine descriptor is the <SQL routine body>.

j) Case:

i) If SQL SECURITY INVOKER is specified, then the SQL security characteristic in the routine descriptor is INVOKER.

ii) Otherwise, the SQL security characteristic in the routine descriptor is DEFINER.

k) If the SQL-invoked routine is an external routine, then:

i) The external name of the routine descriptor is <external routine name>.

ii) The routine descriptor includes an indication of whether the parameter passing style is PARAMETER STYLE SQL or PARAMETER STYLE GENERAL, which is the same as the indication of <parameter style> in *DMS*.

l) The SQL-invoked routine descriptor includes an indication of whether the SQL-invoked routine is deterministic.

m) The SQL-invoked routine descriptor includes an indication of whether the SQL-invoked routine possibly modifies SQL-data, possibly read SQL-data, possibly contains SQL, or does not possibly contain SQL.

n) The SQL-invoked routine descriptor includes an indication of whether the SQL-invoked routine is a null-call function, which is the same as the indication in *DMS*.

o) If *DMS* specifies a <result cast>, then the routine descriptor includes an indication that the SQL-invoked routine specifies a <result cast> and the <data type> specified in the <result cast> of *DMS*.

p) The routine authorization identifier is the <authorization identifier> that owns *S*.

q) The routine SQL-path is *DP*.

NOTE 324 — The routine SQL-path is used to set the routine SQL-path of the current SQL-session when *R* is invoked. The routine SQL-path of the current SQL-session is used by the Syntax Rules of Subclause 10.4, "<routine invocation>", to define the subject routine of <routine invocation>s contained in *R*. The same routine SQL-path is used whenever *R* is invoked.

r) An indication of whether the routine is a schema-level routine.

s) An indication of whether the SQL-invoked routine is dependent on a user-defined type.

NOTE 325 — The notion of an SQL-invoked routine being dependent on a user-defined type is defined in Subclause 4.27, "SQL-invoked routines".

5) The creation timestamp and the last-altered timestamp included in the routine descriptor are the values of CURRENT_TIMESTAMP.

6) If *R* is an external routine, then the routine descriptor of *R* includes further elements determined as follows:

a) Case:

i) If <SQL-data access indication> in the descriptor of *R* is MODIFIES SQL DATA, READS SQL DATA, or CONTAINS SQL, then:

1) Let *P* be the program identified by the <external routine name>.

2) The external routine authorization identifier of *R* is the <module authorization identifier> of the <SQL-client module definition> of *P*.

3) The external routine SQL-path is the <schema name list> immediately contained in the <path specification> that is immediately contained in the <module path specification> of the <SQL-client module definition> of *P*.

ii) Otherwise:

1) The external routine authorization identifier is implementation-defined.

2) The external routine SQL-path is implementation-defined.

NOTE 326 — The external routine SQL-path is used to set the routine SQL-path of the current SQL-session when *R* is invoked. The routine SQL-path of the current SQL-session is used by the Syntax Rules of Subclause 10.4, "<routine invocation>", to define the subject routines of <routine invocation>s contained in the <SQL-client module definition> of *P*. The same external routine SQL-path is used whenever *R* is invoked.

b) The external security characteristic in the routine descriptor is

Case:

i) If <external security clause> specifies EXTERNAL SECURITY DEFINER, then DEFINER.

ii) If <external security clause> specifies EXTERNAL SECURITY INVOKER, then INVOKER.

iii) Otherwise, EXTERNAL SECURITY IMPLEMENTATION DEFINED.

c) The effective SQL parameter list is the *effective SQL parameter list.*

Conformance Rules

1) Without Feature T471, "Result sets return value", conforming SQL language shall not contain a <dynamic result sets characteristic>.

2) Without Feature T322, "Overloading of SQL-invoked functions and procedures", conforming SQL language shall not contain a <schema routine> in which the schema identified by the explicit or implicit schema name of the <schema qualified routine name> includes a routine descriptor whose routine name is <schema qualified routine name>.

3) Without Feature S023, "Basic structured types", conforming SQL language shall not contain a <method specification designator>.

4) Without Feature S241, "Transform functions", conforming SQL language shall not contain a <transform group specification>.

5) Without Feature S024, "Enhanced structured types", an <SQL parameter declaration> shall not contain RESULT.

6) Without Feature T571, "Array-returning external SQL-invoked functions", conforming SQL language shall not contain an <SQL-invoked routine> that defines an array-returning external function.

7) Without Feature T572, "Multiset-returning external SQL-invoked functions", conforming SQL language shall not contain an <SQL-invoked routine> that defines a multiset-returning external function.

8) Without Feature S201, "SQL-invoked routines on arrays", conforming SQL language shall not contain a <parameter type> that is based on an array type.

9) Without Feature S201, "SQL-invoked routines on arrays", conforming SQL language shall not contain a <returns data type> that is based on an array type.

10) Without Feature S202, "SQL-invoked routines on multisets", conforming SQL language shall not contain a <parameter type> that is based on a multiset type.

11) Without Feature S202, "SQL-invoked routines on multisets", conforming SQL language shall not contain a <returns data type> that is based on a multiset type.

12) Without Feature T323, "Explicit security for external routines", conforming SQL language shall not contain an <external security clause>.

13) Without Feature S231, "Structured type locators", conforming SQL language shall not contain a <parameter type> that contains a <locator indication> and that simply contains a <data type> that identifies a structured type.

14) Without Feature S231, "Structured type locators", conforming SQL language shall not contain a <returns data type> that contains a <locator indication> and that simply contains a <data type> that identifies a structured type.

15) Without Feature S232, "Array locators", conforming SQL language shall not contain a <parameter type> that contains a <locator indication> and that simply contains a <data type> that identifies an array type.

16) Without Feature S232, "Array locators", conforming SQL language shall not contain a <returns data type> that contains a <locator indication> and that simply contains a <data type> that identifies an array type.

17) Without Feature S233, "Multiset locators", conforming SQL language shall not contain a <parameter type> that contains a <locator indication> and that simply contains a <data type> that identifies a multiset type.

18) Without Feature S233, "Multiset locators", conforming SQL language shall not contain a <returns data type> that contains a <locator indication> and that simply contains a <data type> that identifies a multiset type.

19) Without Feature T041, "Basic LOB data type support", conforming SQL language shall not contain a <parameter type> that contains a <locator indication> and that simply contains a <data type> that identifies a large object type.

20) Without Feature T041, "Basic LOB data type support", conforming SQL language shall not contain a <returns data type> that contains a <locator indication> and that simply contains a <data type> that identifies a large object type.

21) Without Feature S027, "Create method by specific method name", conforming SQL language shall not contain a <method specification designator> that contains SPECIFIC METHOD.

22) Without Feature T324, "Explicit security for SQL routines", conforming SQL language shall not contain a <rights clause>.

23) Without Feature T326, "Table functions", conforming SQL language shall not contain a <returns table type>.

24) Without Feature T651, "SQL-schema statements in SQL routines", conforming SQL language shall not contain an <SQL routine body> that contains an SQL-schema statement.

25) Without Feature T652, "SQL-dynamic statements in SQL routines", conforming SQL language shall not contain an <SQL routine body> that contains an SQL-dynamic statement.

26) Without Feature T653, "SQL-schema statements in external routines", conforming SQL language shall not contain an <external routine name> that identifies a program in which an SQL-schema statement appears.

27) Without Feature T654, "SQL-dynamic statements in external routines", conforming SQL language shall not contain an <external routine name> that identifies a program in which an SQL-dynamic statement appears.

28) Without Feature T655, "Cyclically dependent routines", conforming SQL language shall not contain an <SQL routine body> that contains a <routine invocation> whose subject routine is generally dependent on the routine descriptor of the SQL-invoked routine specified by <SQL-invoked routine>.

29) Without Feature T272, "Enhanced savepoint management", conforming SQL language shall not contain a <routine characteristics> that contains a <savepoint level indication>.

30) Without Feature B121, "Routine language Ada", conforming SQL language shall not contain a <routine characteristic> that contains a <language clause> that contains ADA.

31) Without Feature B122, "Routine language C", conforming SQL language shall not contain a <routine characteristic>that contains a <language clause> that contains C.

32) Without Feature B123, "Routine language COBOL", conforming SQL language shall not contain a <routine characteristic> that contains a <language clause> that contains COBOL.

33) Without Feature B124, "Routine language Fortran", conforming SQL language shall not contain a <routine characteristic> that contains a <language clause> that contains FORTRAN.

34) Without Feature B125, "Routine language MUMPS", conforming SQL language shall not contain a <routine characteristic> that contains a <language clause> that contains M.

35) Without Feature B126, "Routine language Pascal", conforming SQL language shall not contain a <routine characteristic> that contains a <language clause> that contains PASCAL.

36) Without Feature B127, "Routine language PL/I", conforming SQL language shall not contain a <routine characteristic> that contains a <language clause> that contains PLI.

37) Without Feature B128, "Routine language SQL", conforming SQL language shall not contain a <routine characteristic> that contains a <language clause> that contains SQL.

11.51 <alter routine statement>

Function

Alter a characteristic of an SQL-invoked routine.

Format

```
<alter routine statement> ::=
    ALTER <specific routine designator>
    <alter routine characteristics> <alter routine behavior>

<alter routine characteristics> ::= <alter routine characteristic>...

<alter routine characteristic> ::=
    <language clause>
  | <parameter style clause>
  | <SQL-data access indication>
  | <null-call clause>
  | <dynamic result sets characteristic>
  | NAME <external routine name>

<alter routine behavior> ::= RESTRICT
```

Syntax Rules

1) Let *SR* be the SQL-invoked routine identified by the <specific routine designator> and let *SN* be the <specific name> of *SR*. The schema identified by the explicit or implicit <schema name> of *SN* shall include the descriptor of *SR*.

2) *SR* shall be a schema-level routine.

3) *SR* shall not be an SQL-invoked routine that is dependent on a user-defined type.

   NOTE 327 — "SQL-invoked routine dependent on a user-defined type" is defined in Subclause 4.27, "SQL-invoked routines".

4) If RESTRICT is specified, then:

   a) *SR* shall not be the ordering function included in the descriptor of any user-defined type *UDT*.

   b) *SR* shall not be the subject routine of any <routine invocation>, <method invocation>, <static method invocation>, or <method reference> that is contained in any of the following:

      i) The SQL routine body of any routine descriptor.

      ii) The <query expression> of any view descriptor.

      iii) The <search condition> of any constraint descriptor.

      iv) The triggered action of any trigger descriptor.

   c) *SN* shall not be included in any of the following:

      i) A group descriptor of any transform descriptor.

ii) A user-defined cast descriptor.

5) *SR* shall be an external routine.

6) *SR* shall not be an SQL-invoked method that is an overriding method and the set of overriding methods of *SR* shall be empty.

7) <alter routine characteristics> shall contain at most one <language clause>, at most one <parameter style clause>, at most one <SQL-data access indication>, at most one <null-call clause>, at most one <maximum dynamic result sets>, and at most one <external routine name>.

8) If <maximum dynamic result sets> is specified, then *SR* shall be an SQL-invoked procedure.

9) If <language clause> is specified, then:

   a) <language clause> shall not specify SQL.

   b) Depending on whether the <language clause> specifies ADA, C, COBOL, FORTRAN, M, PASCAL, or PLI, let the operative data type correspondences table be Table 16, "Data type correspondences for Ada", Table 17, "Data type correspondences for C", Table 18, "Data type correspondences for COBOL", Table 19, "Data type correspondences for Fortran", Table 20, "Data type correspondences for M", Table 21, "Data type correspondences for Pascal", or Table 22, "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".

   c) Any <data type> in the effective SQL parameter list entry of *SR* 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".

   NOTE 328 — "Effective SQL parameter list" is defined in Subclause 11.50, "<SQL-invoked routine>".

Access Rules

1) The enabled authorization identifiers shall include the <authorization identifier> that owns the schema identified by the implicit or explicit <schema name> of *SN*.

General Rules

1) If *SR* is not a method, then the routine descriptor of *SR* is modified:

   a) If <dynamic result sets characteristic> is specified, then the value of <maximum dynamic result sets>.

   b) If <language clause> is specified, then the <language name> contained in the <language clause>.

   c) If <external routine name> is specified, then the external name of the routine descriptor is <external routine name>.

   d) If <parameter style clause> is specified, then the routine descriptor includes an indication of whether the parameter passing style is PARAMETER STYLE SQL or PARAMETER STYLE GENERAL.

   e) If the <SQL-data access indication> is specified, then an indication of whether the SQL-invoked routine's <SQL-data access indication> is READS SQL DATA, MODIFIES SQL DATA, CONTAINS SQL, or NO SQL.

   f) If <null-call clause> is specified, then an indication of whether the SQL-invoked routine is a null-call function.

2) If *SR* is a method, then let *DMS* be the descriptor of the corresponding method specification. *DMS* is modified:

   a) If <language clause> is specified, then the <language name> contained in the <language clause>.

   b) If <parameter style clause> is specified, then the method specification descriptor includes an indication of whether the parameter passing style is PARAMETER STYLE SQL or PARAMETER STYLE GENERAL.

   c) If the <SQL-data access indication> is specified, then an indication of whether the SQL-invoked routine's <SQL-data access indication> is READS SQL DATA, MODIFIES SQL DATA, CONTAINS SQL, or NO SQL.

   d) If <null-call clause> is specified, then an indication of whether the method should not be invoked if any argument is the null value.

3) If *SR* is a method, then the routine descriptor of *SR* is modified:

   a) If <external routine name> is specified, then the external name of the routine descriptor is <external routine name>. If <parameter style clause> is specified, then the method specification descriptor includes an indication of whether the parameter passing style is PARAMETER STYLE SQL or PARAMETER STYLE GENERAL.

Conformance Rules

1) Without Feature F381, "Extended schema manipulation", conforming SQL language shall not contain an <alter routine statement>.

11.52 <drop routine statement>

Function

Destroy an SQL-invoked routine.

Format

```
<drop routine statement> ::= DROP <specific routine designator> <drop behavior>
```

Syntax Rules

1) Let *SR* be the SQL-invoked routine identified by the <specific routine designator> and let *SN* be the <specific name> of *SR*. The schema identified by the explicit or implicit <schema name> of *SN* shall include the descriptor of *SR*.

2) *SR* shall be a schema-level routine.

3) *SR* shall not be dependent on any user-defined type.

   NOTE 329 — The notion of an SQL-invoked routine being dependent on a user-defined type is defined in Subclause 4.27, "SQL-invoked routines".

4) If RESTRICT is specified, then *SR* shall not be the ordering function included in the descriptor of any user-defined type *UDT*.

5) If RESTRICT is specified, then:

   a) *SR* shall not be the subject routine of any <routine invocation>, <method invocation>, <static method invocation>, or <method reference> that is contained in any of the following:

      i) The SQL routine body of any routine descriptor.

      ii) The <query expression> of any view descriptor.

      iii) The <search condition> of any constraint descriptor.

      iv) The triggered action of any trigger descriptor.

   b) *SN* shall not be a included in any of the following:

      i) A group descriptor of any transform descriptor.

      ii) A user-defined cast descriptor.

   NOTE 330 — If CASCADE is specified, then such referencing objects will be dropped by the execution of the <revoke statement> specified in the General Rules of this Subclause.

6) Let the containing schema be the schema identified by the <schema name> explicitly or implicitly contained in *SN*.

Access Rules

1) Let *A* be the <authorization identifier> that owns the schema identified by the <schema name> of *SN*. The enabled authorization identifiers shall include *A*.

General Rules

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

```
REVOKE EXECUTE ON SPECIFIC ROUTINE
    SN FROM A CASCADE
```

2) Let *DN* be the <user-defined type name> of a user-defined type whose descriptor includes *SN* in the group descriptor of any transform descriptor. Let *GN* be the <group name> of that group descriptor. The following <drop transform statement> is effectively executed without further Access Rule checking:

```
DROP TRANSFORM GN FOR DN CASCADE
```

3) Let *UDCD* be a user-defined cast descriptor that includes *SN* as its cast function. Let *SDT* be the source data type included in *UDCD*. Let *TDT* be the target data type included in *UDCD*. The following <drop user-defined cast statement> is effectively executed without further Access Rule checking:

```
DROP CAST ( DN AS TD ) CASCADE
```

4) If *SR* is the ordering function included in the descriptor of a user-defined type *UDT*, then let *UDTN* be a <path-resolved user-defined type name> that identifies *UDT*. The following <drop user-defined ordering statement> is effectively executed without further Access Rule checking:

```
DROP ORDERING FOR UDTN CASCADE
```

5) The descriptor of *SR* is destroyed.

Conformance Rules

1) Without Feature F032, "CASCADE drop behavior", conforming SQL language shall not contain a <drop routine statement> that contains a <drop behavior> that contains CASCADE.

2) Without Feature S024, "Enhanced structured types", conforming SQL language shall not contain a <drop routine statement> that contains a <specific routine designator> that identifies a method.

11.53 <user-defined cast definition>

Function

Define a user-defined cast.

Format

```
<user-defined cast definition> ::=
    CREATE CAST <left paren> <source data type> AS <target data type> <right paren>
    WITH <cast function>
    [ AS ASSIGNMENT ]

<cast function> ::= <specific routine designator>

<source data type> ::= <data type>

<target data type> ::= <data type>
```

Syntax Rules

1) Let *SDT* be the <source data type>. The data type identified by *SDT* is called the *source data type.*

2) Let *TDT* be the <target data type>. The data type identified by *TDT* is called the *target data type.*

3) There shall be no user-defined cast for *SDT* and *TDT.*

4) At least one of *SDT* or *TDT* shall contain a <schema-resolved user-defined type name> or a <reference type>.

5) If *SDT* contains a <schema-resolved user-defined type name>, then let *SSDT* be the schema that includes the descriptor of the user-defined type identified by *SDT.*

6) If *SDT* contains a <reference type>, then let *SSDT* be the schema that includes the descriptor of the referenced type of the reference type identified by *SDT.*

7) If *TDT* contains a <schema-resolved user-defined type name>, then let *STDT* be the schema that includes the descriptor of the user-defined type identified by *TDT.*

8) If *TDT* contains a <reference type>, then let *STDT* be the schema that includes the descriptor of the referenced type of the reference type identified by *TDT.*

9) If both *SDT* and *TDT* contain a <schema-resolved user-defined type name> or a <reference type>, then the <authorization identifier> that owns *SSDT* and the <authorization identifier> that owns *STDT* shall be equivalent.

10) Let *F* be the SQL-invoked routine identified by <cast function>. *F* is called the *cast function* for source data type *SDT* and target data type *TDT.*

    a) *F* shall have exactly one SQL parameter, and its declared type shall be *SDT.*

    b) The result data type of *F* shall be *TDT.*

c) The <authorization identifier> that owns *SSDT* or *STDT* (both, if both *SDT* and *TDT* are <schema-resolved user-defined type name>s) shall own the schema that includes the SQL-invoked routine descriptor of *F*.

d) *F* shall be deterministic.

e) *F* shall not possibly modify SQL-data.

f) *F* shall not possibly read SQL-data.

Access Rules

1) The enabled authorization identifiers shall include the <authorization identifier> that owns the schema that includes the routine descriptor of *F*.

2) If *SDT* contains a <schema-resolved user-defined type name> or a <reference type>, then the enabled authorization identifiers shall include the <authorization identifier> that owns *SSDT*.

3) If *TDT* contains a <schema-resolved user-defined type name> or a <reference type>, then the enabled authorization identifiers shall include the <authorization identifier> that owns *STDT*.

General Rules

1) A user-defined cast descriptor *CFD* is created that describes the user-defined cast. *CFD* includes the name of the source data type, the name of the target data type, the specific name of the cast function, and, if and only if AS ASSIGNMENT is specified, an indication that the cast function is implicitly invocable.

Conformance Rules

1) Without Feature S211, "User-defined cast functions", conforming SQL language shall not contain a <user-defined cast definition>.

11.54 <drop user-defined cast statement>

Function

Destroy a user-defined cast.

Format

```
<drop user-defined cast statement> ::=
    DROP CAST <left paren> <source data type> AS <target data type> <right paren>
    <drop behavior>
```

Syntax Rules

1) Let *SDT* be the <source data type> and let *TDT* be the <target data type>.

2) Let *CF* be the user-defined cast whose user-defined cast descriptor includes *SDT* as the source data type and *TDT* as the target data type.

3) Let *SN* be the specific name of the cast function *F* included in the user-defined cast descriptor of *CF*.

4) The schema identified by the <schema name> of *SN* shall include the descriptor of *F*.

5) Let *CS* be any <cast specification> such that:

   a) The <value expression> of *CS* has declared type *P*.

   b) The <cast target> of *CS* is either *TDT* or a domain with declared type *TDT*.

   c) The type designator of *SDT* is in the type precedence of *P*.

   d) No other data type *Q* whose type designator precedes *SDT* in the type precedence list of *P* such that there is a user-defined cast CF_q whose user-defined cast descriptor includes *Q* as the source data type and *TDT* as the target data type.

6) Let *PS* be any SQL procedure statement that is dependent on *F*.

7) If RESTRICT is specified, then neither *CS* nor *PS* shall be generally contained in any of the following:

   a) The SQL routine body of any routine descriptor.

   b) The <query expression> of any view descriptor.

   c) The <search condition> of any constraint descriptor.

   d) The trigger action of any trigger descriptor.

   NOTE 331 — If CASCADE is specified, then such referencing objects will be dropped as specified in the General Rules of this Subclause.

Access Rules

1) The enabled authorization identifier shall include the <authorization identifier> that owns the schema identified by the implicit or explicit <schema name> of *SN*.

General Rules

1) Let *R* be any SQL-invoked routine that contains *CS* or *PS* in its SQL routine body. Let *SN* be the specific name of *R*. The following <drop routine statement> is effectively executed without further Access Rule checking:

```
DROP SPECIFIC ROUTINE SN CASCADE
```

2) Let *V* be any view that contains *CS* or *PS* in its <query expression>. Let *VN* be the <table name> of *V*. The following <drop view statement> is effectively executed without further Access Rule checking:

```
DROP VIEW VN CASCADE
```

3) Let *T* be any table that contains *CS* or *PS* in the <search condition> of any constraint descriptor included in the table descriptor of *T*. Let *TN* be the <table name> of *T*. The following <drop table statement> is effectively executed without further Access Rule checking:

```
DROP TABLE TN CASCADE
```

4) Let *A* be any assertion that contains *CS* or *PS* in its <search condition>. Let *AN* be the <constraint name> of *A*. The following <drop assertion statement> is effectively executed without further Access Rule checking:

```
DROP ASSERTION AN CASCADE
```

5) Let *D* be any domain that contains *CS* or *PS* in the <search condition> of any constraint descriptor. Let *DN* be the <domain name> of *D*. The following <drop domain statement> is effectively executed without further Access Rule checking:

```
DROP DOMAIN DN CASCADE
```

6) Let *T* be any trigger whose trigger descriptor includes a trigger action that contains *CS* or *PS*. Let *TN* be the <trigger name> of *T*. The following <drop trigger statement> is effectively executed without further Access Rule checking:

```
DROP TRIGGER TN
```

7) The descriptor of *CF* is destroyed.

Conformance Rules

1) Without Feature S211, "User-defined cast functions", conforming SQL language shall not contain a <drop user-defined cast statement>.

11.55 <user-defined ordering definition>

Function

Define a user-defined ordering for a user-defined type.

Format

```
<user-defined ordering definition> ::=
    CREATE ORDERING FOR <schema-resolved user-defined type name> <ordering form>

<ordering form> ::=
    <equals ordering form>
  | <full ordering form>

<equals ordering form> ::= EQUALS ONLY BY <ordering category>

<full ordering form> ::= ORDER FULL BY <ordering category>

<ordering category> ::=
    <relative category>
  | <map category>
  | <state category>

<relative category> ::= RELATIVE WITH <relative function specification>

<map category> ::= MAP WITH <map function specification>

<state category> ::= STATE [ <specific name> ]

<relative function specification> ::= <specific routine designator>

<map function specification> ::= <specific routine designator>
```

Syntax Rules

1) Let *UDTN* be the <schema-resolved user-defined type name>. Let *UDT* be the user-defined type identified by *UDTN*.

2) The descriptor of *UDT* shall include an ordering form that specifies NONE.

3) If *UDT* is not a maximal supertype, then

   Case:

   a) If <equals ordering form> is specified, then the comparison form of every direct supertype of *UDT* shall be EQUALS.

   b) Otherwise, the comparison form of every direct supertype of *UDT* shall be FULL.

4) If <relative category> or <state category> is specified, then *UDT* shall be a maximal supertype.

5) If <map category> is specified and *UDT* is not a maximal supertype, then the comparison category of every direct supertype of *UDT* shall be MAP.

   NOTE 332 — The comparison categories of two user-defined types in the same subtype family shall be the same.

6) Case:

   a) If <state category> is specified, then

      i) *UDT* shall not be a distinct type.

      ii) EQUALS ONLY shall be specified.

      iii) The declared type of each attribute of *UDT* shall not be UDT-NC-ordered.

      iv) Case:

         1) If <specific name> is specified, then let *SN* be <specific name>. If *SN* contains a <schema name>, then that <schema name> shall be equivalent to the <schema name> of *UDTN*.

         2) Otherwise, let *SN* be an implementation-dependent <specific name> whose <schema name> is equivalent to the <schema name> *S* of *UDTN*. This implementation-dependent <specific name> shall not be equivalent to the <specific name> of any other routine descriptor in the schema identified by *S*.

   b) Otherwise:

      i) Let *F* be the SQL-invoked routine identified by the <specific routine designator> *SRD*.

      ii) *F* shall be deterministic.

      iii) *F* shall not possibly modify SQL-data.

7) If <relative function specification> is specified, then:

   a) *F* shall have exactly two SQL parameters whose declared type is *UDT*.

   b) *F* shall be an SQL-invoked regular function.

   c) The result data type of *F* shall be INTEGER.

8) If <map function specification> is specified, then:

   a) *F* shall have exactly one SQL parameter whose declared type is *UDT*.

   b) The result data type of *F* shall be a predefined data type.

   c) The result data type of *F* is an operand of an equality operation. The Syntax Rules of Subclause 9.9, "Equality operations", apply.

   d) If FULL is specified, then the result data type of *F* is an operand of an ordering operation. The Syntax Rules of Subclause 9.12, "Ordering operations", apply.

Access Rules

1) The enabled authorization identifiers shall include the <authorization identifier> that owns the schema that includes the descriptor of *UDT* and the schema that includes the routine descriptor of *F*.

General Rules

1) If <state category> is specified, then:

   a) Let C_1, ..., C_n be the components of the representation of the user-defined type.

   b) Let *SNUDT* be the <schema name> of the schema that includes the descriptor of *UDT*.

   c) The following <SQL-invoked routine> is effectively executed:

```
CREATE FUNCTION SNUDT.EQUALS ( UDT1 UDTN, UDT2 UDTN )
    RETURNS BOOLEAN
    SPECIFIC SN
    DETERMINISTIC
    CONTAINS SQL
    STATIC DISPATCH
    RETURN
      ( TRUE AND
        UDT1.SPECIFICTYPE = UDT2.SPECIFICTYPE AND
        UDT1.C1 = UDT2.C1 AND
        ...
        UDT1.Cn = UDT2.Cn )
```

2) Case:

   a) If EQUALS is specified, then the ordering form in the user-defined type descriptor of *UDT* is set to EQUALS.

   b) Otherwise, the ordering form in the user-defined type descriptor of *UDT* is set to FULL.

3) Case:

   a) If RELATIVE is specified, then the ordering category in the user-defined type descriptor of *UDT* is set to RELATIVE.

   b) If MAP is specified, then the ordering category in the user-defined type descriptor of *UDT* is set to map.

   c) Otherwise, the ordering category in the user-defined type descriptor of *UDT* is set to STATE.

4) The <specific routine designator> identifying the ordering function, depending on the ordering category, in the descriptor of *UDT* is set to *SRD*.

Conformance Rules

1) Without Feature S251, "User-defined orderings", conforming SQL shall not contain a <user-defined ordering definition>.

   NOTE 333 — If MAP is specified, then the Conformance Rules of Subclause 9.9, "Equality operations", apply. If ORDER FULL BY MAP is specified, then the Conformance Rules of Subclause 9.12, "Ordering operations", also apply.

11.56 <drop user-defined ordering statement>

Function

Destroy a user-defined ordering method.

Format

```
<drop user-defined ordering statement> ::=
    DROP ORDERING FOR <schema-resolved user-defined type name> <drop behavior>
```

Syntax Rules

1) Let *UDTN* be the <schema-resolved user-defined type name>. Let *UDT* be the user-defined type identified by *UDTN.*
2) The descriptor of *UDT* shall include an ordering form that specifies EQUALS or FULL.
3) Let *OF* be the ordering function of *UDT.*
4) If RESTRICT is specified, then none of the following shall contain an operand of an equality operation, grouping operation or ordering operation whose declared type is some user-defined type *T1* whose comparison type is *UDT*:
   a) The SQL routine body of any routine descriptor.
   b) The <query expression> of any view descriptor.
   c) The <search condition> of any constraint descriptor.
   d) The triggered action of any trigger descriptor.

   NOTE 334 — If CASCADE is specified, then such referencing objects will be dropped as specified in the General Rules of this Subclause.

Access Rules

1) The enabled authorization identifiers shall include the <authorization identifier> that owns the schema identified by the implicit or explicit <schema name> of *UDTN.*

General Rules

1) Let *R* be any SQL-invoked routine whose SQL routine body contains an operand of an equality operation, grouping operation, or ordering operation whose declared type is some user-defined type *T1* whose comparison type is *UDT*. Let *SN* be the specific name of *R*. The following <drop routine statement> is effectively executed without further Access Rule checking:

   ```
   DROP SPECIFIC ROUTINE SN CASCADE
   ```

2) Let *V* be any view whose <query expression> contains an operand of an equality operation, grouping operation, or ordering operation whose declared type is some user-defined type *T1* whose comparison type is *UDT*. Let *VN* be the <table name> of *V*. The following <drop view statement> is effectively executed without further Access Rule checking:

```
DROP VIEW VN CASCADE
```

3) Let *T* be any table whose table descriptor includes a constraint descriptor of a constraint *C* whose <search condition> contains an operand of an equality operation, grouping operation, or ordering operation whose declared type is some user-defined type *T1* whose comparison type is *UDT*. Let *TN* be the <table name> of *T*. Let *TCN* be the <constraint name> of *C*. The following <alter table statement> is effectively executed without further Access Rule checking:

```
ALTER TABLE TN DROP CONSTRAINT TCN CASCADE
```

4) Let *A* be any assertion whose <search condition> contains an operand of an equality operation, grouping operation, or ordering operation whose declared type is some user-defined type *T1* whose comparison type is *UDT*. Let *AN* be the <constraint name> of *A*. The following <drop assertion statement> is effectively executed without further Access Rule checking:

```
DROP ASSERTION AN CASCADE
```

5) Let *D* be any domain whose descriptor includes a constraint descriptor that includes an operand of an equality operation, grouping operation, or ordering operation whose declared type is some user-defined type *T1* whose comparison type is *UDT* in the <search condition> of any constraint descriptor or in the <default option> included in the domain descriptor of *D*. Let *DN* be the <domain name> of *D*. The following <drop domain statement> is effectively executed without further Access Rule checking:

```
DROP DOMAIN DN CASCADE
```

6) Let *T* be any trigger whose triggered action contains an operand of an equality operation, grouping operation, or ordering operation whose declared type is some user-defined type *T1* whose comparison type is *UDT*. Let *TN* be the <trigger name> of *T*. The following <drop trigger statement> is effectively executed without further Access Rule checking:

```
DROP TRIGGER TN
```

7) In the descriptor of *UDT*, the ordering form is set to NONE and the ordering category is set to STATE. No ordering function is included in the descriptor of *UDT*.

Conformance Rules

1) Without Feature S251, "User-defined orderings", conforming SQL language shall not contain a <drop user-defined ordering statement>.

11.57 <transform definition>

Function

Define one or more transform functions for a user-defined type.

Format

```
<transform definition> ::=
    CREATE { TRANSFORM | TRANSFORMS } FOR
    <schema-resolved user-defined type name> <transform group>...

<transform group> ::=
    <group name> <left paren> <transform element list> <right paren>

<group name> ::= <identifier>

<transform element list> ::= <transform element> [ <comma> <transform element> ]

<transform element> ::=
    <to sql>
  | <from sql>

<to sql> ::= TO SQL WITH <to sql function>

<from sql> ::= FROM SQL WITH <from sql function>

<to sql function> ::= <specific routine designator>

<from sql function> ::= <specific routine designator>
```

Syntax Rules

1) Let *TD* be the <transform definition>. Let *DTN* be the <schema-resolved user-defined type name> immediately contained in *TD*. Let *DT* be the data type identified by *DTN*. Let *SDT* be the schema that includes the descriptor of *DT*. Let *TRD* be the transform descriptor included in the data type descriptor of *DT*.

2) No two <transform group>s immediately contained in *TD* shall have the same <group name>.

3) The SQL-invoked function identified by <to sql function> is called the *to-sql function*. The SQL-invoked function identified by <from sql function> is called the *from-sql function*.

4) Let *n* be the number of <transform group>s immediately contained in *TD*. For *i* ranging from 1 to *n*:

   a) Let TG_i be the *i*-th <transform group> immediately contained in *TD*. Let GN_i be the <group name> contained in TG_i.

   b) Each of <to sql> and <from sql> immediately contained in TG_i shall be contained at most once in a <transform element list>.

   c) The SQL-invoked routines identified by <to sql function> and <from sql function> shall be SQL-invoked functions that are deterministic and do not possibly modify SQL-data.

d) *TRD* shall not include a transform group descriptor *GD* that includes a group name that is equivalent to GN_i.

e) Let *SDTT* be the set that includes every data type DTT_j that is either a proper supertype or a proper subtype of *DT* such that the transform descriptor included in the data type descriptor of DTT_j includes a group descriptor $GDT_{j,k}$ that includes a group name that is equivalent to GN_i. *SDTT* shall be empty.

f) If <to sql> is specified, then let TSF_i be the SQL-invoked function identified by <to sql function>.

   i) Case:

      1) If TSF_i is an SQL-invoked method, then TSF_i shall have exactly two SQL parameters such that the declared type of the first SQL parameter is *DT* and the declared type of the second SQL parameter is a predefined data type. The result data type of TSF_i shall be *DT*.

      2) Otherwise, TSF_i shall have exactly one SQL parameter whose declared type is a predefined data type. The result data type of TSF_i shall be *DT*.

   ii) If *DT* is a structured type and TSF_i is an SQL-invoked method, then TSF_i shall be a type-preserving function.

g) If <from sql> is specified, then let FSF_i be the SQL-invoked function identified by <from sql function>. FSF_i shall have exactly one SQL parameter whose declared type is *DT*. The result data type of FSF_i shall be a predefined data type.

h) If <to sql> and <from sql> are both specified, then

   Case:

   i) If TSF_i is an SQL-invoked method, then the result data type of FSF_i and the data type of the second SQL parameter of TSF_i shall be compatible.

   ii) Otherwise, the result data type of FSF_i and the data type of the first SQL parameter of TSF_i shall be compatible.

Access Rules

1) For *i* ranging from 1 to *n*, the enabled authorization identifiers shall include the <authorization identifier> that owns *SDT* and the schema that includes the routine descriptors of TSF_i, if any, and FSF_i, if any.

General Rules

1) A <group name> specifies the group name that identifies a transform group.

2) For every TG_i, 1 (one) $\leq i \leq n$:

   a) A new group descriptor GD_i is created that includes the <group name> immediately contained in TG_i. GD_i is included in the list of transform group descriptors included in *TRD*.

b) If <to sql> is specified, then the specific name of the to-sql function in GD_i is set to TSF_i.

c) If <from sql> is specified, then the specific name of the from-sql function in GD_i is set to FSF_i.

Conformance Rules

1) Without Feature S241, "Transform functions", conforming SQL language shall not contain a <transform definition>.

11.58 <alter transform statement>

Function

Change the definition of one or more transform groups.

Format

```
<alter transform statement> ::=
    ALTER { TRANSFORM | TRANSFORMS }
    FOR <schema-resolved user-defined type name> <alter group>...

<alter group> ::=
    <group name> <left paren> <alter transform action list> <right paren>

<alter transform action list> ::=
    <alter transform action> [ { <comma> <alter transform action> }... ]

<alter transform action> ::=
    <add transform element list>
  | <drop transform element list>
```

Syntax Rules

1) Let *DN* be the <schema-resolved user-defined type name> and let *D* be the data type identified by *DN*. The schema identified by the explicit or implicit schema name of *DN* shall include the data type descriptor of *D*. Let *S* be that schema. Let *TD* be the transform descriptor included in the data type descriptor of *D*.

2) The scope of *DN* is the entire <alter transform statement> *AT*.

3) Let *n* be the number of <group name>s contained in *AT*. For *i* ranging from 1 to *n*:

   a) Let GN_i be the *i*-th <group name> contained in *AT*.

   b) For each GN_i, there shall be a transform group descriptor included in *TD* whose group name is equivalent to GN_i. Let GD_i be this transform group descriptor.

Access Rules

1) The enabled authorization identifiers shall include the <authorization identifier> that owns *S*.

General Rules

1) For *i* ranging from 1 to *n*, GD_i is modified as specified by <alter transform action list>.

Conformance Rules

1) Without Feature S242, “Alter transform statement”, conforming SQL language shall not contain an <alter transform statement>.

11.59 <add transform element list>

Function

Add a transform element (<to sql> and/or <from sql>) to an existing transform group.

Format

```
<add transform element list> ::=
    ADD <left paren> <transform element list> <right paren>
```

Syntax Rules

1) Let *AD* be the <add transform element list>.

2) Let *DN* be the <schema-resolved user-defined type name> immediately contained in the containing <alter transform statement>. Let *D* be the user-defined type identified by *DN*. Let *TD* be the transform descriptor included in the data type descriptor of *D*.

3) Let *GD* be the transform group descriptor included in *TD* whose group name is equivalent to <group name> immediately contained in the containing <alter group>.

4) Each of <to sql> and <from sql> (immediately contained in *AD*) shall be contained at most once in the <transform element list>.

5) If *GD* includes a specific name of the to-sql function, then *AD* shall not contain <to sql>.

6) If *GD* includes a specific name of the from-sql function, then *AD* shall not contain <from sql>.

7) The SQL-invoked routine identified by either <to sql function> or <from sql function> shall be an SQL-invoked function that is deterministic and does not possibly modify SQL-data.

8) If <to sql> is specified, then let *TSF* be the SQL-invoked function identified by <to sql function>.

   a) Case:

      i) If *TSF* is an SQL-invoked method, then *TSF* shall have exactly two SQL parameters such that the declared type of the first SQL parameter is *D* and the declared type of the second SQL parameter is a predefined data type. The result data type of *TSF* shall be *D*.

      ii) Otherwise, *TSF* shall have exactly one SQL parameter whose declared type is a predefined data type. The result data type of *TSF* shall be *D*.

   b) If *D* is a structured type, then *TSF* shall be a type-preserving function.

   c) If *GD* includes the specific name of a from-sql function, then let *FS* be the SQL-invoked function that is identified by this specific name.

      Case:

      i) If *TSF* is an SQL-invoked method, then the result data type of *FS* and the data type of the second SQL parameter of *TSF* shall be compatible.

ii) Otherwise, the result data type of *FS* and the data type of the first SQL parameter of *TSF* shall be compatible.

9) If <from sql> is specified, then let *FSF* be the SQL-invoked function identified by <from sql function>.

a) *FSF* shall have exactly one SQL parameter whose declared type is *D*. The result data type of *FSF* shall be a predefined data type.

b) If *GD* includes the specific name of a to-sql function, then let *TS* be the SQL-invoked routine that is identified by this specific name.

Case:

i) If *TS* is an SQL-invoked method, then the result data type of *FSF* and the data type of the second SQL parameter of *TS* shall be compatible.

ii) Otherwise, the result data type of *FSF* and the data type of the first SQL parameter of *TS* shall be compatible.

Access Rules

1) The enabled authorization identifiers shall include the <authorization identifier> that owns the schema that includes the routine descriptors of *TSF*, if any, and *FSF*, if any.

General Rules

1) If <to sql> is specified, then the specific name of the to-sql function in *GD* is set to *TSF*.

2) If <from sql> is specified, then the specific name of the from-sql function in *GD* is set to *FSF*.

Conformance Rules

None.

11.60 <drop transform element list>

Function

Remove a transform element (<to sql> and/or <from sql>) from a transform group.

Format

```
<drop transform element list> ::=
    DROP <left paren> <transform kind>
    [ <comma> <transform kind> ] <drop behavior> <right paren>

<transform kind> ::=
    TO SQL
  | FROM SQL
```

Syntax Rules

1) Let *DN* be the <schema-resolved user-defined type name> immediately contained in the containing <alter transform statement>. Let *D* be the user-defined type identified by *DN*. Let *TD* be the transform descriptor included in the data type descriptor of *D*.

2) Let *GD* be the transform group descriptor included in *TD* whose group name is equivalent to <group name> immediately contained in the containing <alter group>.

3) Each of TO SQL and FROM SQL shall only be specified at most once in the <drop transform element list>.

4) If TO SQL is specified then *GD* shall include the specific name of a to-sql function. Let this function be *TSF*.

5) If FROM SQL is specified then *GD* shall include the specific name of a from-sql function. Let this function be *FSF*.

6) If RESTRICT is specified, then:

   a) If TO SQL is specified, then there shall be no external routine that has an SQL parameter whose associated to-sql function is *TSF* nor shall there be an external function that has *TSF* as the to-sql function associated with the result.

   b) If FROM SQL is specified, then there shall be no external routine that has an SQL parameter whose associated from-sql function is *FSF*.

Access Rules

None.

General Rules

1) If FROM SQL is specified, then:
   a) Let *FSN* be the <specific name> of any external routine that has an SQL parameter whose associated from-sql function is *FSF*. The following <drop routine statement> is effectively executed without further Access Rule checking:

   ```
   DROP SPECIFIC ROUTINE FSN CASCADE
   ```

   b) The specific name of the from-sql function is removed from *GD*.
2) If TO SQL is specified, then:
   a) Let *TSN* be the <specific name> of any external routine that has an SQL parameter whose associated to-sql function is *TSF*. The following <drop routine statement> is effectively executed without further Access Rule checking:

   ```
   DROP SPECIFIC ROUTINE TSN CASCADE
   ```

   b) Let *RSN* be the <specific name> of any external function that has *TSF* as the to-sql function associated with the result. The following <drop routine statement> is effectively executed without further Access Rule checking:

   ```
   DROP SPECIFIC ROUTINE RSN CASCADE
   ```

   c) The specific name of the to-sql function is removed from *GD*.

Conformance Rules

None.

11.61 <drop transform statement>

Function

Remove one or more transform functions associated with a transform.

Format

```
<drop transform statement> ::=
    DROP { TRANSFORM | TRANSFORMS } <transforms to be dropped>
    FOR <schema-resolved user-defined type name> <drop behavior>

<transforms to be dropped> ::=
    ALL
  | <transform group element>

<transform group element> ::= <group name>
```

Syntax Rules

1) Let *DT* be the data type identified by <schema-resolved user-defined type name>. Let *SDT* be the schema that includes the descriptor of *DT*. Let *TRD* be the transform descriptor included in the data type descriptor of *DT*. Let *n* be the number of transform group descriptors in *TRD*.

2) If <transform group element> is specified, then *TRD* shall include a transform group descriptor *GD* that includes a group name that is equivalent to the <group name> immediately contained in <transform group element>.

3) If RESTRICT is specified, then

   Case:

   a) If ALL is specified, then for *i* ranging from 1 (one) to *n*:

      i) Let GD_i be the *i*-th transform group descriptor included in *TRD*.

      ii) If GD_i includes the specific name of a from-sql function FSF_i then there shall be no external routine that has an SQL parameter whose associated from-sql function is FSF_i.

      iii) If GD_i includes the specific name of a to-sql function TSF_i then there shall be no external routine that has an SQL parameter whose associated to-sql function is TSF_i nor shall there be an external function that has TSF_i as the to-sql function associated with the result.

   b) Otherwise:

      i) If *GD* includes the specific name of a from-sql function *FSF* then there shall be no external routine that has an SQL parameter whose associated from-sql function is *FSF*.

      ii) If *GD* includes the specific name of a to-sql function *TSF* then there shall be no external routine that has an SQL parameter whose associated to-sql function is *TSF* nor shall there be an external function that has *TSF* as the to-sql function associated with the result.

Access Rules

1) The enabled authorization identifiers shall include the <authorization identifier> that owns *SDT*.

General Rules

1) Case:

   a) If ALL is specified, then, for *i* ranging from 1 (one) to *n*:

      i) Let GD_i be the *i*-th transform group descriptor included in *TRD*.

      ii) If GD_i includes the specific name of a from-sql function FSF_i, then let *FSN* be the <specific name> of any external routine that has an SQL parameter whose associated from-sql function is FSF_i. The following <drop routine statement> is effectively executed without further Access Rule checking:

      ```
      DROP SPECIFIC ROUTINE FSN CASCADE
      ```

      iii) If GD_i includes the specific name of a to-sql function TSF_i, then:

         1) Let *TSN* be the <specific name> of any external routine that has an SQL parameter whose associated to-sql function is TSF_i. The following <drop routine statement> is effectively executed without further Access Rule checking:

         ```
         DROP SPECIFIC ROUTINE TSN CASCADE
         ```

         2) Let *RSN* be the <specific name> of any external function that has TSF_i as the to-sql function associated with the result. The following <drop routine statement> is effectively executed without further Access Rule checking:

         ```
         DROP SPECIFIC ROUTINE RSN CASCADE
         ```

      iv) GD_i is removed from *TRD*.

   b) Otherwise:

      i) If *GD* includes the specific name of a from-sql function *FSF*, then let *FSN* be the <specific name> of any external routine that has an SQL parameter whose associated from-sql function is *FSF*. The following <drop routine statement> is effectively executed without further Access Rule checking:

      ```
      DROP SPECIFIC ROUTINE FSN CASCADE
      ```

      ii) If *GD* includes the specific name of a to-sql function *TSF*, then:

         1) Let *TSN* be the <specific name> of any external routine that has an SQL parameter whose associated to-sql function is *TSF*. The following <drop routine statement> is effectively executed without further Access Rule checking:

         ```
         DROP SPECIFIC ROUTINE TSN CASCADE
         ```

2) Let *RSN* be the <specific name> of any external function that has *TSF* as the to-sql function associated with the result. The following <drop routine statement> is effectively executed without further Access Rule checking:

```
DROP SPECIFIC ROUTINE RSN CASCADE
```

iii) *GD* is removed from *TRD*.

Conformance Rules

1) Without Feature S241, “Transform functions”, conforming SQL language shall not contain a <drop transform statement>.

11.62 <sequence generator definition>

Function

Define an external sequence generator.

Format

```
<sequence generator definition> ::=
    CREATE SEQUENCE <sequence generator name> [ <sequence generator options> ]

<sequence generator options> ::= <sequence generator option> ...

<sequence generator option> ::=
    <sequence generator data type option>
  | <common sequence generator options>

<common sequence generator options> ::= <common sequence generator option> ...

<common sequence generator option> ::=
    <sequence generator start with option>
  | <basic sequence generator option>

<basic sequence generator option> ::=
    <sequence generator increment by option>
  | <sequence generator maxvalue option>
  | <sequence generator minvalue option>
  | <sequence generator cycle option>

<sequence generator data type option> ::= AS <data type>

<sequence generator start with option> ::= START WITH <sequence generator start value>

<sequence generator start value> ::= <signed numeric literal>

<sequence generator increment by option> ::= INCREMENT BY <sequence generator increment>

<sequence generator increment> ::= <signed numeric literal>

<sequence generator maxvalue option> ::=
    MAXVALUE <sequence generator max value>
  | NO MAXVALUE

<sequence generator max value> ::= <signed numeric literal>

<sequence generator minvalue option> ::=
    MINVALUE <sequence generator min value>
  | NO MINVALUE

<sequence generator min value> ::= <signed numeric literal>

<sequence generator cycle option> ::=
    CYCLE
  | NO CYCLE
```

Syntax Rules

1) Let *SEQ* be the sequence generator defined by the <sequence generator definition> *SEQD*.

2) If *SEQD* is contained in a <schema definition> *SD* and the <sequence generator name> *SQN* contains a <schema name>, then that <schema name> shall be equivalent to the implicit or explicit <schema name> of *SD*.

3) The schema identified by the explicit or implicit schema name of *SQN* shall not include a sequence generator descriptor whose sequence generator name is equivalent to *SQN*.

4) If *SEQD* is contained in a <schema definition>, then let *A* be the explicit or implicit <authorization identifier> of the <schema definition>. Otherwise, let *A* be the <authorization identifier> that owns the schema identified by the implicit or explicit <schema name> of *SQN*.

5) Each of <sequence generator data type option>, <sequence generator start with option>, <sequence generator increment by option>, <sequence generator maxvalue option>, <sequence generator minvalue option>, and <sequence generator cycle option> shall be specified at most once.

6) If <sequence generator data type option> is specified, then <data type> shall be an exact numeric type *DT* with scale 0 (zero); otherwise, let *DT* be an implementation-defined exact numeric type with scale 0 (zero).

7) The Syntax Rules of Subclause 9.22, "Creation of a sequence generator", are applied with <common sequence generator options> as *OPTIONS* and *DT* as *DATA TYPE*.

Access Rules

1) If a <sequence generator definition> is contained in an <SQL-client module definition>, then the enabled authorization identifiers shall include *A*.

General Rules

1) The General Rules of Subclause 9.22, "Creation of a sequence generator", are applied with <common sequence generator options> as *OPTIONS* and *DT* as *DATA TYPE*, yielding a sequence generator descriptor *SEQDS*. The sequence generator name included in *SEQDS* is set to *SQN*.

2) A privilege descriptor is created that defines the USAGE privilege on *SEQ* to *A*. This privilege is grantable. The grantor for this privilege descriptor is set to the special grantor value "_SYSTEM".

Conformance Rules

1) Without Feature T176, "Sequence generator support", conforming SQL language shall not contain a <sequence generator definition>.

11.63 <alter sequence generator statement>

Function

Change the definition of an external sequence generator.

Format

```
<alter sequence generator statement> ::=
    ALTER SEQUENCE <sequence generator name> <alter sequence generator options>

<alter sequence generator options> ::= <alter sequence generator option>...

<alter sequence generator option> ::=
    <alter sequence generator restart option>
  | <basic sequence generator option>

<alter sequence generator restart option> ::=
    RESTART WITH <sequence generator restart value>

<sequence generator restart value> ::= <signed numeric literal>
```

Syntax Rules

1) Let *SEQ* be the sequence generator descriptor identified by the <sequence generator name> *SQN*. Let *DT* be the data type of *SEQ*.
2) The schema identified by the explicit or implicit schema name of *SQN* shall include *SEQ*.
3) The scope of *SQN* is the <alter sequence generator statement>.
4) The Syntax Rules of Subclause 9.23, "Altering a sequence generator", are applied with <alter sequence generator options> as *OPTIONS* and *SEQ* as *SEQUENCE*.

Access Rules

1) The enabled authorization identifiers shall include the <authorization identifier> that owns the schema identified by the explicit or implicit schema name of *SQN*.

General Rules

1) The General Rules of Subclause 9.23, "Altering a sequence generator", are applied with <alter sequence generator options> as *OPTIONS* and *SEQ* as *SEQUENCE*.

Conformance Rules

1) Without Feature T176, "Sequence generator support", conforming SQL language shall not contain an <alter sequence generator statement>.

11.64 <drop sequence generator statement>

Function

Destroy an external sequence generator.

Format

```
<drop sequence generator statement> ::=
    DROP SEQUENCE <sequence generator name> <drop behavior>
```

Syntax Rules

1) Let *SEQ* be the sequence generator identified by the <sequence generator name> *SQN*.

2) The schema identified by the explicit or implicit schema name of *SQN* shall include the descriptor of *SEQ*.

3) If RESTRICT is specified, then *SEQ* shall not be referenced in any of the following:

   a) The SQL routine body of any routine descriptor.

   b) The trigger action of any trigger descriptor.

   NOTE 335 — If CASCADE is specified, then such referenced objects will be dropped by the execution of the <revoke statement> specified in the General Rules of this Subclause.

4) Let *A* be the <authorization identifier> that owns the schema identified by the <schema name> of the sequence generator identified by *SQN*.

Access Rules

1) The enabled authorization identifiers shall include *A*.

General Rules

1) The following <revoke statement> is effectively executed with a current authorization identifier of "_SYSTEM" and without further Access Rule checking:
   ```
   REVOKE USAGE ON SEQUENCE SQN FROM A CASCADE
   ```

2) The descriptor of *SEQ* is destroyed.

Conformance Rules

1) Without Feature T176, "Sequence generator support", conforming SQL language shall not contain a <drop sequence generator statement>.

This page intentionally left blank.

12 Access control

12.1 <grant statement>

Function

Define privileges and role authorizations.

Format

```
<grant statement> ::=
    <grant privilege statement>
  | <grant role statement>
```

Syntax Rules

None.

Access Rules

None.

General Rules

1) For every involved grantee *G* and for every domain *D1* owned by *G*, if all of the following are true:
   a) The applicable privileges for *G* include the grantable REFERENCES privilege on every column referenced in the <search condition> *SC* included in a domain constraint descriptor included in the domain descriptor of *D1*.
   b) The applicable privileges for *G* include the grantable EXECUTE privileges on all SQL-invoked routines that are subject routines of <routine invocation>s contained in *SC*.
   c) The applicable privileges for *G* include the grantable SELECT privilege on every table *T1* and every method *M* such that there is a <method reference> *MR* contained in *SC* such that *T1* is in the scope of the <value expression primary> of *MR* and *M* is the method identified by the <method name> of *MR* included in a domain constraint descriptor included in the domain descriptor of *D1*.
   d) The applicable privileges for *G* include the grantable SELECT privilege WITH HIERARCHY OPTION on at least one supertable of the scoped table of every <reference resolution> contained in *SC*.

e) The applicable privileges for *G* include the grantable USAGE privilege on all domains, character sets, collations, and transliterations whose <domain name>s, <character set name>s, <collation name>s, and <transliteration name>s, respectively, are included in the domain descriptor of *D1*.

then for every privilege descriptor with <action> USAGE, a grantor of "_SYSTEM", object *D1*, and grantee *G* that is not grantable, the following <grant statement> is effectively executed with a current user identifier of "_SYSTEM" and without further Access Rule checking:

```
GRANT USAGE ON DOMAIN D1 TO G WITH GRANT OPTION
```

2) For every involved grantee *G* and for every collation *C1* owned by *G*, if the applicable privileges for *G* include a grantable USAGE privilege for the character set name included in the collation descriptor of *C1* and a grantable USAGE privilege for the transliteration name, if any, included in the collation descriptor of *C1*, then for every privilege descriptor with <action> USAGE, a grantor of "_SYSTEM", object of *C1*, and grantee *G* that is not grantable, the following <grant statement> is effectively executed with a current user identifier of "_SYSTEM" and without further Access Rule checking:

```
GRANT USAGE ON COLLATION C1 TO G WITH GRANT OPTION
```

3) For every involved grantee *G* and for every transliteration *T1* owned by *G*, if the applicable privileges for *G* contain a grantable USAGE privilege for every character set identified by a <character set specification> contained in the <transliteration definition> of *T1*, then for every privilege descriptor with <action> *P*, a grantor of "_SYSTEM", object of *T1*, and grantee *G* that is not grantable, the following <grant statement> is effectively executed as though the current user identifier were "_SYSTEM" and without further Access Rule checking:

```
GRANT P ON TRANSLATION T1 TO G WITH GRANT OPTION
```

4) For every table *T* specified by some involved privilege descriptor and for each view *V* owned by some involved grantee *G* such that *T* or some column *CT* of *T* is referenced in the <query expression> *QE* of *V*, or *T* is a supertable of the scoped table of a <reference resolution> contained in *QE*, let RT_i, for *i* ranging from 1 (one) to the number of tables identified by the <table reference>s contained in *QE*, be the <table name>s of those tables. For every column *CV* of *V*:

   a) Let $CRT_{i,j}$, for *j* ranging from 1 (one) to the number of columns of RT_i that are underlying columns of *CV*, be the <column name>s of those columns.

   b) If, following successful execution of the <grant statement>, all of the following are true:

      i) The applicable privileges for *G* include grantable SELECT privileges on all of the columns $CRT_{i,j}$.

      ii) The applicable privileges for *G* include grantable EXECUTE privileges on all SQL-invoked routines that are subject routines of <routine invocation>s contained in *QE*.

      iii) The applicable privileges for *G* include grantable SELECT privilege on every table *T1* and every method *M* such that there is a <method reference>. *MR* contained in *QE* such that *T1* is in the scope of the <value expression primary> of *MR* and *M* is the method identified by the <method name> of *MR*.

iv) The applicable privileges for *G* include grantable SELECT privilege WITH HIERARCHY OPTION on at least one supertable of the scoped table of every <reference resolution> that is contained in *QE*.

then the following <grant statement> is effectively executed as though the current user identifier were "_SYSTEM" and without further Access Rule checking:

```
GRANT SELECT (CV) ON V TO G WITH GRANT OPTION
```

c) If, following successful execution of the <grant statement>, the applicable privileges for *G* will include REFERENCES($CRT_{i,j}$) for all *i* and for all *j*, and will include a REFERENCES privilege on some column of RT_i for all *i*, then:

i) Case:

1) If all of the following are true, then let *WGO* be "WITH GRANT OPTION".

A) The applicable privileges for *G* will include grantable REFERENCES($CRT_{i,j}$) for all *i* and for all *j*, and will include a grantable REFERENCES privilege on some column of RT_i for all *i*.

B) The applicable privileges for *G* include grantable EXECUTE privileges on all SQL-invoked routines that are subject routines of <routine invocation>s contained in *QE*.

C) The applicable privileges for *G* include grantable SELECT privilege on every table *T1* and every method *M* such that there is a <method reference>. *MR* contained in *QE* such that *T1* is in the scope of the <value expression primary> of *MR* and *M* is the method identified by the <method name> of *MR*.

D) The applicable privileges for *G* include grantable SELECT privilege WITH HIERARCHY OPTION on at least one supertable of the scoped table of every <reference resolution> that is contained in *QE*.

2) Otherwise, let *WGO* be a zero-length string.

ii) The following <grant statement> is effectively executed as though the current user identifier were "_SYSTEM" and without further Access Rule checking:

```
GRANT REFERENCES (CV) ON V TO G WGO
```

d) If, following successful execution of the <grant statement>, the applicable privileges for *G* include grantable SELECT privilege on every column of *V*, then the following <grant statement> is effectively executed as though the current user identifier were "_SYSTEM" and without further Access Rule checking:

```
GRANT SELECT ON V TO G WITH GRANT OPTION
```

e) Following successful execution of the <grant statement>,

Case:

i) If the applicable privileges for *G* include REFERENCES privilege on every column of *V*, then let *WGO* be a zero-length string.

ii) If the applicable privileges for *G* include grantable REFERENCES privilege on every column of *V*, then let *WGO* be "WITH GRANT OPTION".

iii) The following <grant statement> is effectively executed as though the current user identifier were "_SYSTEM" and without further Access Rule checking:

```
GRANT REFERENCES ON V TO G WITH GRANT OPTION
```

5) Following the successful execution of the <grant statement>, for every table *T* specified by some involved privilege descriptor and for every updatable view *V* owned by some grantee *G* such that *T* is some leaf underlying table of the <query expression> of *V*:

a) Let *VN* be the <table name> of *V*.

b) If *QE* is fully updatable with respect to *T*, and the applicable privileges for *G* include *PA*, where *PA* is either INSERT, UPDATE, or DELETE, then the following <grant statement> is effectively executed as though the current user identifier were "_SYSTEM" and without further Access Rule checking:

```
GRANT PA ON VN TO G
```

c) If *QE* is fully updatable with respect to *T*, and the applicable privileges for *G* include grantable *PA* privilege on *T*, where *PA* is either INSERT, UPDATE, or DELETE, then the following <grant statement> is effectively executed as though the current user identifier were "_SYSTEM" and without further Access Rule checking:

```
GRANT PA ON VN TO G WITH GRANT OPTION
```

d) For each column *CV* of *V*, named *CVN*, that has a counterpart *CT* in *T*, named *CTN*, if *QE* is fully or partially updatable with respect to *T*, and the applicable privileges for *G* include *PA*(*CTN*) privilege on *T*, where *PA* is INSERT or UPDATE, then the following <grant statement> is effectively executed as though the current user identifier were "_SYSTEM" and without further Access Rule checking:

```
GRANT PA(CVN) ON VN TO G
```

e) For each column *CV* of *V*, named *CVN*, that has a counterpart *CT* in *T*, named *CTN*, if *QE* is fully or partially updatable with respect to *T*, and the applicable privileges for *G* include grantable *PA*(*CTN*) privilege on *T*, where *PA* is INSERT or UPDATE, then the following <grant statement> is effectively executed as though the current user identifier were "_SYSTEM" and without further Access Rule checking:

```
GRANT PA(CVN) ON VN TO G WITH GRANT OPTION
```

6) For every involved grantee *G* and for every referenceable view *V*, named *VN*, owned by *G*, if following the successful execution of the <grant statement>, the applicable privileges for *G* include grantable UNDER privilege on the direct supertable of *V*, then the following <grant statement> is effectively executed with a current authorization identifier of "_SYSTEM" and without further Access Rule checking:

```
GRANT UNDER ON VN TO G WITH GRANT OPTION
```

7) For every involved grantee *G* and for every schema-level SQL-invoked routine *R1* owned by *G*, if the applicable privileges for *G* contain all of the privileges necessary to successfully execute every <SQL procedure statement> contained in the <routine body> of *R1* are grantable, then for every privilege

descriptor with <action> EXECUTE, a grantor of "_SYSTEM", object of *R1*, and grantee *G* that is not grantable, the following <grant statement> is effectively executed with a current authorization identifier of "_SYSTEM" and without further Access Rule checking:

```
GRANT EXECUTE ON R1 TO G WITH GRANT OPTION
```

NOTE 336 — The privileges necessary include the EXECUTE privilege on every subject routine of every <routine invocation> contained in the <SQL procedure statement>.

8) If two privilege descriptors are identical except that one indicates that the privilege is grantable and the other indicates that the privilege is not grantable, then both privilege descriptors are set to indicate that the privilege is grantable.

9) If two privilege descriptors are identical except that one indicates WITH HIERARCHY OPTION and the other does not, then both privilege descriptors are set to indicate that the privilege has the WITH HIERARCHY OPTION.

10) Redundant duplicate privilege descriptors are removed from the collection of all privilege descriptors.

Conformance Rules

None.

12.2 <grant privilege statement>

Function

Define privileges.

Format

```
<grant privilege statement> ::=
    GRANT <privileges> TO <grantee> [ { <comma> <grantee> }... ]
    [ WITH HIERARCHY OPTION ]
    [ WITH GRANT OPTION ]
    [ GRANTED BY <grantor> ]
```

Syntax Rules

1) Let *O* be the object identified by the <object name> contained in <privileges>.
2) Let *U* be the current user identifier and let *R* be the current role name.
3) Case:
   a) If GRANTED BY <grantor> is not specified, then

      Case:
      i) If *U* is not the null value, then let *A* be *U*.
      ii) Otherwise, let *A* be *R*.
   b) If GRANTED BY CURRENT_USER is specified, then let *A* be *U*.
   c) If GRANTED BY CURRENT_ROLE is specified, then let *A* be *R*.
4) A set of privilege descriptors is identified. The privilege descriptors identified are those defining, for each <action> explicitly or implicitly in <privileges>, that <action> on *O* held by *A* with grant option.
5) The schema identified by the explicit or implicit qualifier of the <object name> shall include the descriptor of *O*.
6) If WITH HIERARCHY OPTION is specified, then:
   a) <privileges> shall specify an <action> of SELECT without a <privilege column list> and without a <privilege method list>.
   b) *O* shall be a table of a structured type.

Access Rules

1) The applicable privileges shall include a privilege identifying *O*.

General Rules

1) The <object privileges> specify one or more privileges on the object identified by the <object name>.

2) For every identified privilege descriptor *IPD*, a privilege descriptor is created for each <grantee>, that specifies grantee <grantee>, action <action>, object *O*, and grantor *A*. Let *CPD* be the set of privilege descriptors created.

3) For every privilege descriptor in *CPD* whose action is INSERT, UPDATE, or REFERENCES without a column name, privilege descriptors are also created and added to *CPD* for each column *C* in *O* for which *A* holds the corresponding privilege with grant option. For each such column, a privilege descriptor is created that specifies the identical <grantee>, the identical <action>, object *C*, and grantor *A*.

4) For every privilege descriptor in *CPD* whose action is SELECT without a column name or method name, privilege descriptors are also created and added to *CPD* for each column *C* in *O* for which *A* holds the corresponding privilege with grant option. For each such column, a privilege descriptor is created that specifies the identical <grantee>, the identical <action>, object *C*, and grantor *A*.

5) For every privilege descriptor in *CPD* whose action is SELECT without a column name or method name, if the table *T* identified by the object of the privilege descriptor is a table of a structured type *TY*, then table/method privilege descriptors are also created and added to *CPD* for each method *M* of *TY* for which *A* holds the corresponding privilege with grant option. For each such method, a table/method privilege descriptor is created that specifies the identical <grantee>, the identical <action>, object consisting of the pair of table *T* and method *M*, and grantor *A*.

6) If WITH GRANT OPTION was specified, then each privilege descriptor also indicates that the privilege is grantable.

7) Let *SWH* be the set of privilege descriptors in *CPD* whose action is SELECT WITH HIERARCHY OPTION. Let *ST* be the set of subtables of *O*. For every table *T* in *ST* and for every privilege descriptor in *SWH* grantee *G*, and grantor *A*,

   Case:

   a) If the privilege is grantable, then let *WG* be "WITH GRANT OPTION".

   b) Otherwise, let *WGO* be the zero-length string.

   The following <grant statement> is effectively executed without further Access Rule checking:

   ```
   GRANT SELECT ON T TO G WGO GRANTED BY A
   ```

8) For every combination of <grantee> and <action> on *O* specified in <privileges>, if there is no corresponding privilege descriptor in *CPD*, then a completion condition is raised: *warning — privilege not granted.*

9) If ALL PRIVILEGES was specified, then for each grantee *G*, if there is no privilege descriptor in *CPD* specifying grantee *G*, then a completion condition is raised: *warning — privilege not granted.*

10) The *set of involved privilege descriptors* is defined to be *CPD*.

11) The *set of involved grantees* is defined as the set of specified <grantee>s.

Conformance Rules

1) Without Feature S024, "Enhanced structured types", conforming SQL language shall not contain a <specific routine designator> contained in a <grant privilege statement> that identifies a method.

2) Without Feature S081, "Subtables", conforming SQL language shall not contain a <grant privilege statement> that contains WITH HIERARCHY OPTION.

12.3 <privileges>

Function

Specify privileges.

Format

```
<privileges> ::= <object privileges> ON <object name>

<object name> ::=
    [ TABLE ] <table name>
  | DOMAIN <domain name>
  | COLLATION <collation name>
  | CHARACTER SET <character set name>
  | TRANSLATION <transliteration name>
  | TYPE <schema-resolved user-defined type name>
  | SEQUENCE <sequence generator name>
  | <specific routine designator>

<object privileges> ::=
    ALL PRIVILEGES
  | <action> [ { <comma> <action> }... ]

<action> ::=
    SELECT
  | SELECT <left paren> <privilege column list> <right paren>
  | SELECT <left paren> <privilege method list> <right paren>
  | DELETE
  | INSERT [ <left paren> <privilege column list> <right paren> ]
  | UPDATE [ <left paren> <privilege column list> <right paren> ]
  | REFERENCES [ <left paren> <privilege column list> <right paren> ]
  | USAGE
  | TRIGGER
  | UNDER
  | EXECUTE

<privilege method list> ::=
    <specific routine designator> [ { <comma> <specific routine designator> }... ]

<privilege column list> ::= <column name list>

<grantee> ::=
    PUBLIC
  | <authorization identifier>

<grantor> ::=
    CURRENT_USER
  | CURRENT_ROLE
```

Syntax Rules

1) ALL PRIVILEGES is equivalent to the specification of all of the privileges on <object name> for which the <grantor> has grantable privilege descriptors.

2) If the <object name> of the <grant statement> or <revoke statement> specifying <privileges> specifies <table name>, then let *T* be the table identified by that <table name>. *T* shall not be a declared local temporary table.

3) If <object name> specifies a <domain name>, <collation name>, <character set name>, <transliteration name>, <schema-resolved user-defined type name>, or <sequence generator name>, then <privileges> may specify USAGE. Otherwise, USAGE shall not be specified.

4) If <object name> specifies a <table name> that identifies a base table, then <privileges> may specify TRIGGER; otherwise, TRIGGER shall not be specified.

5) If <object name> specifies a <schema-resolved user-defined type name> that identifies a structured type or specifies a <table name>, then <privileges> may specify UNDER; otherwise, UNDER shall not be specified.

6) If *T* is a temporary table, then <privileges> shall specify ALL PRIVILEGES.

7) If the object identified by <object name> of the <grant statement> or <revoke statement> is an SQL-invoked routine, then <privileges> may specify EXECUTE; otherwise, EXECUTE shall not be specified.

8) The <object privileges> specify one or more privileges on the object identified by <object name>.

9) Each <column name> in a <privilege column list> shall identify a column of *T*.

10) If <privilege method list> is specified, then <object name> shall specify a <table name> that identifies a table of a structured type *TY* and each <specific routine designator> in the <privilege method list> shall identify a method of *TY*.

11) UPDATE (<privilege column list>) is equivalent to the specification of UPDATE (<column name>) for each <column name> in <privilege column list>. INSERT (<privilege column list>) is equivalent to the specification of INSERT (<column name>) for each <column name> in <privilege column list>. REFERENCES (<privilege column list>) is equivalent to the specification of REFERENCES (<column name>) for each <column name> in <privilege column list>. SELECT (<privilege column list>) is equivalent to the specification of SELECT (<column name>) for each <column name> in <privilege column list>. SELECT (<privilege method list>) is equivalent to the specification of SELECT (<specific routine designator>) for each <specific routine designator> in <privilege method list>.

Access Rules

None.

General Rules

1) Case:

   a) If a <grantor> of CURRENT_USER is specified and there is no current user identifier, then an exception condition is raised: *invalid grantor.*

b) If a <grantor> of CURRENT_ROLE is specified and there is no current role, then an exception condition is raised: *invalid grantor.*

2) A <grantee> of PUBLIC denotes at all times a list of <grantee>s containing all of the <authorization identifier>s in the SQL-environment.

3) SELECT (<column name>) specifies the SELECT privilege on the indicated column and implies one or more column privilege descriptors.

4) SELECT (<specific routine designator>) specifies the SELECT privilege on the indicated method for the table identified by <object name> and implies one or more table/method privilege descriptors.

5) SELECT with neither <privilege column list> nor <privilege method list> specifies the SELECT privilege on all columns of *T* including any columns subsequently added to *T* and implies a table privilege descriptor and one or more column privilege descriptors. If *T* is a table of a structured type *TY*, then SELECT also specifies the SELECT privilege on all methods of the type *TY*, including any methods subsequently added to the type *TY*, and implies one or more table/method privilege descriptors.

6) UPDATE (<column name>) specifies the UPDATE privilege on the indicated column and implies one or more column privilege descriptors. If the <privilege column list> is omitted, then UPDATE specifies the UPDATE privilege on all columns of *T*, including any column subsequently added to *T* and implies a table privilege descriptor and one or more column privilege descriptors.

7) INSERT (<column name>) specifies the INSERT privilege on the indicated column and implies one or more column privilege descriptors. If the <privilege column list> is omitted, then INSERT specifies the INSERT privilege on all columns of *T*, including any column subsequently added to *T* and implies a table privilege descriptor and one or more column privilege descriptors.

8) REFERENCES (<column name>) specifies the REFERENCES privilege on the indicated column and implies one or more column privilege descriptors. If the <privilege column list> is omitted, then REFERENCES specifies the REFERENCES privilege on all columns of *T*, including any column subsequently added to *T* and implies a table privilege descriptor and one or more column privilege descriptors.

9) *B* has the WITH ADMIN OPTION on a role if a role authorization descriptor identifies the role as granted to *B* WITH ADMIN OPTION or a role authorization descriptor identifies it as granted WITH ADMIN OPTION to another applicable role for *B*.

Conformance Rules

1) Without Feature T332, "Extended roles", conforming SQL language shall not contain a <grantor>.

2) Without Feature T211, "Basic trigger capability", conforming SQL language shall not contain an <action> that contains TRIGGER.

3) Without Feature S081, "Subtables", conforming SQL language shall not contain a <privileges> that contains an <action> that contains UNDER and that contains an <object name> that contains a <table name>.

4) Without Feature S023, "Basic structured types", conforming SQL language shall not contain a <privileges> that contains an <action> that contains UNDER and that contains an <object name> that contains a <schema-resolved user-defined type name> that identifies a structured type.

5) Without Feature S024, "Enhanced structured types", conforming SQL language shall not contain a <privileges> that contains an <action> that contains USAGE and that contains an <object name> that contains a <schema-resolved user-defined type name> that identifies a structured type.

6) Without Feature T281, "SELECT privilege with column granularity", in conforming SQL language, an <action> that contains SELECT shall not contain a <privilege column list>.

7) Without Feature F731, "INSERT column privileges", in conforming SQL language, an <action> that contains INSERT shall not contain a <privilege column list>.

8) Without Feature S024, "Enhanced structured types", conforming SQL language shall not contain a <privilege method list>.

12.4 <role definition>

Function

Define a role.

Format

```
<role definition> ::= CREATE ROLE <role name> [ WITH ADMIN <grantor> ]
```

Syntax Rules

1) The specified <role name> shall not be equivalent to any other <authorization identifier> in the SQL-environment.

Access Rules

1) The privileges necessary to execute the <role definition> are implementation-defined.

General Rules

1) A <role definition> defines a role.
2) Let *U* be the current user identifier and *R* be the current role name.
3) Case:
   a) If WITH ADMIN <grantor> is not specified, then

      Case:
      i) If *U* is not the null value, then let *A* be *U*.
      ii) Otherwise, let *A* be *R*.
   b) If WITH ADMIN CURRENT_USER is specified, then let *A* be *U*.
   c) If WITH ADMIN CURRENT_ROLE is specified, then let *A* be *R*.
4) A role authorization descriptor is created that identifies that the role identified by <role name> has been granted to *A* WITH ADMIN OPTION, with a grantor of "_SYSTEM".

Conformance Rules

1) Without Feature T331, "Basic roles", conforming SQL language shall not contain a <role definition>.
2) Without Feature T332, "Extended roles", conforming SQL language shall not contain a <role definition> that immediately contains WITH ADMIN.

12.5 <grant role statement>

Function

Define role authorizations.

Format

```
<grant role statement> ::=
    GRANT <role granted> [ { <comma> <role granted> }... ]
    TO <grantee> [ { <comma> <grantee> }... ]
    [ WITH ADMIN OPTION ]
    [ GRANTED BY <grantor> ]

<role granted> ::= <role name>
```

Syntax Rules

1) No role identified by a specified <grantee> shall be contained in any role identified by a specified <role granted>; that is, no cycles of role grants are allowed.

2) Let *U* be the current user identifier and *R* be the current role name.

3) Case:

   a) If GRANTED BY <grantor> is not specified, then

      Case:

      i) If *U* is not the null value, then let *A* be *U*.

      ii) Otherwise, let *A* be *R*.

   b) If GRANTED BY CURRENT_USER is specified, then let *A* be *U*.

   c) If GRANTED BY CURRENT_ROLE is specified, then let *A* be *R*.

Access Rules

1) Every role identified by <role granted> shall be contained in the applicable roles for *A* and the corresponding role authorization descriptors shall specify WITH ADMIN OPTION.

General Rules

1) For every <grantee> specified, a set of role authorization descriptors is created that defines the grant of each role identified by a <role granted> to the <grantee> with a grantor of *A*.

2) If WITH ADMIN OPTION is specified, then each role authorization descriptor also indicates that the role is grantable with the WITH ADMIN OPTION.

3) If two role authorization descriptors are identical except that one indicates that the role is grantable WITH ADMIN OPTION and the other indicates that the role is not, then both role authorization descriptors are set to indicate that the role is grantable with the WITH ADMIN OPTION.

4) Redundant duplicate role authorization descriptors are removed from the collection of all role authorization descriptors.

5) The *set of involved privilege descriptors* is the union of the sets of privilege descriptors corresponding to the applicable privileges for every <role granted> specified.

6) The *set of involved grantees* is the union of the set of <grantee>s and the set of <role name>s that contain at least one of the <role name>s that is possibly specified as a <grantee>.

Conformance Rules

1) Without Feature T331, "Basic roles", conforming SQL language shall not contain a <grant role statement>.

12.6 <drop role statement>

Function

Destroy a role.

Format

```
<drop role statement> ::= DROP ROLE <role name>
```

Syntax Rules

1) Let *R* be the role identified by the specified <role name>.

Access Rules

1) At least one of the enabled authorization identifiers shall have a role authorization identifier that authorizes *R* with the WITH ADMIN OPTION.

General Rules

1) Let *A* be any <authorization identifier> identified by a role authorization descriptor as having been granted to *R*.
2) The following <revoke role statement> is effectively executed without further Access Rule checking:

   ```
   REVOKE R FROM A
   ```

3) The descriptor of *R* is destroyed.

Conformance Rules

1) Without Feature T331, "Basic roles", conforming SQL language shall not contain a <drop role statement>.

12.7 <revoke statement>

Function

Destroy privileges and role authorizations.

Format

```
<revoke statement> ::=
    <revoke privilege statement>
  | <revoke role statement>

<revoke privilege statement> ::=
    REVOKE [ <revoke option extension> ] <privileges>
    FROM <grantee> [ { <comma> <grantee> }... ]
    [ GRANTED BY <grantor> ]
    <drop behavior>

<revoke option extension> ::=
    GRANT OPTION FOR
  | HIERARCHY OPTION FOR

<revoke role statement> ::=
    REVOKE [ ADMIN OPTION FOR ] <role revoked> [ { <comma> <role revoked> }... ]
    FROM <grantee> [ { <comma> <grantee> }... ]
    [ GRANTED BY <grantor> ]
    <drop behavior>

<role revoked> ::= <role name>
```

Syntax Rules

1) Let *O* be the object identified by the <object name> contained in <privileges>. If *O* is a table *T*, then let *S* be the set of subtables of *O*. If *T* is a table of a structured type, then let *TY* be that type.

2) If WITH HIERARCHY OPTION is specified, the <privileges> shall specify an <action> of SELECT without a <privilege column list> and without a <privilege method list> and *O* shall be a table of a structured type.

3) Let *U* be the current user identifier and *R* be the current role name.

4) Case:

   a) If GRANTED BY <grantor> is not specified, then

      Case:

      i) If *U* is not the null value, then let *A* be *U*.

      ii) Otherwise, let *A* be *R*.

   b) If GRANTED BY CURRENT_USER is specified, then let *A* be *U*.

c) If GRANTED BY CURRENT_ROLE is specified, then let *A* be *R*.

5) SELECT is equivalent to specifying both the SELECT table privilege and SELECT (<privilege column list>) for all columns of <table name>. If *T* is a table of a structured type *TY*, then SELECT also specifies SELECT (<privilege column list>) for all columns inherited from *T* in each of the subtables of *T*, and SELECT (<privilege method list>) for all methods of *TY* in each of the subtables of *T*.

6) INSERT is equivalent to specifying both the INSERT table privilege and INSERT (<privilege column list>) for all columns of <table name>.

7) UPDATE is equivalent to specifying both the UPDATE table privilege and UPDATE (<privilege column list>) for all columns of <table name>, as well as UPDATE (<privilege column list>) for all columns inherited from *T* in each of the subtables of *T*.

8) REFERENCES is equivalent to specifying both the REFERENCES table privilege and REFERENCES (<privilege column list>) for all columns of <table name>, as well as REFERENCES (<privilege column list>) for all columns inherited from *T* in each of the subtables of *T*.

9) Case:

a) If the <revoke statement> is a <revoke privilege statement>, then for every <grantee> specified, a set of privilege descriptors is identified. A privilege descriptor *P* is said to be *identified* if it belongs to the set of privilege descriptors that defined, for any <action> explicitly or implicitly in <privileges>, that <action> on *O*, or any of the objects in *S*, granted by *A* to <grantee>.

NOTE 337 — Column privilege descriptors become identified when <action> explicitly or implicitly contains a <privilege column list>. Table/method descriptors become identified when <action> explicitly or implicitly contains a <privilege method list>.

b) If the <revoke statement> is a <revoke role statement>, then for every <grantee> specified, a set of role authorization descriptors is identified. A role authorization descriptor is said to be *identified* if it defines the grant of any of the specified <role revoked>s to <grantee> with grantor *A*.

10) A privilege descriptor *D* is said to be *directly dependent on* another privilege descriptor *P* if either:

a) All of the following conditions hold:

i) *P* indicates that the privilege that it represents is grantable.

ii) The grantee of *P* is the same as the grantor of *D* or the grantee of *P* is PUBLIC, or, if the grantor of *D* is a <role name>, the grantee of *P* belongs to the set of applicable roles of the grantor of *D*.

iii) Case:

1) *P* and *D* are both column privilege descriptors. The action and the identified column of *P* are the same as the action and identified column of *D*, respectively.

2) *P* and *D* are both table privilege descriptors. The action and the identified table of *P* are the same as the action and the identified table of *D*, respectively.

3) *P* and *D* are both execute privilege descriptors. The action and the identified SQL-invoked routine of *P* are the same as the action and the identified SQL-invoked routine of *D*, respectively.

4) *P* and *D* are both usage privilege descriptors. The action and the identified domain, character set, collation, transliteration, user-defined type, or sequence generator of *P* are the same as the action and the identified domain, character set, collation, transliteration, user-defined type, or sequence generator of *D*, respectively.

5) *P* and *D* are both under privilege descriptors. The action and the identified user-defined type or table of *P* are the same as the action and the identified user-defined type or table of *D*, respectively.

6) *P* and *D* are both table/method privilege descriptors. The action and the identified method and table of *P* are the same as the action and the identified method and table of *D*, respectively.

b) All of the following conditions hold:

i) The privilege descriptor for *D* indicates that its grantor is the special grantor value "_SYSTEM".

ii) The action of *P* is the same as the action of *D*.

iii) The grantee of *P* is the owner of the table, collation, or transliteration identified by *D* or the grantee of *P* is PUBLIC.

iv) One of the following conditions hold:

1) *P* and *D* are both table privilege descriptors, the privilege descriptor for *D* identifies the <table name> of an updatable view *V*, and the identified table of *P* is the underlying table of the <query expression> of *V*.

2) *P* and *D* are both column privilege descriptors, the privilege descriptor *D* identifies a <column name> *CVN* explicitly or implicitly contained in the <view column list> of a <view definition> *V*, and one of the following is true:

A) *V* is an updatable view. For every column *CV* identified by a <column name> *CVN*, there is a corresponding column in the underlying table of the <query expression> *TN*. Let *CTN* be the <column name> of the column of the <query expression> from which *CV* is derived. The action for *P* is UPDATE or INSERT and the identified column of *P* is *TN.CTN*.

B) For every table *T* identified by a <table reference> contained in the <query expression> of *V* and for every column *CT* that is a column of *T* and an underlying column of *CV*, the action for *P* is REFERENCES and either the identified column of *P* is *CT* or the identified table of *P* is *T*.

C) For every table *T* identified by a <table reference> contained in the <query expression> of *V* and for every column *CT* that is a column of *T* and an underlying column of *CV*, the action for *P* is SELECT and either the identified column of *P* is *CT* or the identified table of *P* is *T*.

3) The privilege descriptor *D* identifies the <collation name> of a <collation definition> *CO* and the identified character set name of *P* is included in the collation descriptor for *CO*, or the identified transliteration name of *P* is included in the collation descriptor for *CO*.

4) The privilege descriptor *D* identifies the <transliteration name> of a <transliteration definition> *TD* and the identified character set name of *P* is contained in the <source character set specification> or the <target character set specification> immediately contained in *TD*.

c) All of the following conditions hold:

i) The privilege descriptor for *D* indicates that its grantor is the special grantor value "_SYSTEM".

ii) The grantee of *P* is the owner of the domain identified by *D* or the grantee of *P* is PUBLIC.

iii) The privilege descriptor *D* identifies the <domain name> of a <domain definition> *DO* and either the column privilege descriptor *P* has an action of REFERENCES and identifies a column referenced in the <search condition> included in the domain descriptor for *DO*, or the privilege descriptor *P* has an action of USAGE and identifies a domain, collation, character set, or transliteration whose <domain name>, <collation name>, <character set name> or <transliteration name>, respectively, is contained in the <search condition> of the domain descriptor for *DO*.

11) The *privilege dependency graph* is a directed graph such that all of the following are true:

a) Each node represents a privilege descriptor.

b) Each arc from node *P1* to node *P2* represents the fact that *P2* directly depends on *P1*.

An *independent node* is a node that has no incoming arcs.

12) A privilege descriptor *P* is said to be *modified* if all of the following are true:

a) *P* indicates that the privilege that it represents is grantable.

b) *P* directly depends on an identified privilege descriptor or a modified privilege descriptor.

c) Case:

i) If *P* is neither a SELECT nor a REFERENCES column privilege descriptor that identifies a <column name> *CVN* explicitly or implicitly contained in the <view column list> of a <view definition> *V*, then let *XO* and *XA* respectively be the identifier of the object identified by a privilege descriptor *X* and the action of *X*. Within the set of privilege descriptors upon which *P* directly depends, there exist some *XO* and *XA* for which the set of identified privilege descriptors unioned with the set of modified privilege descriptors include all privilege descriptors specifying the grant of *XA* on *XO* WITH GRANT OPTION.

ii) If *P* is a column privilege descriptor that identifies a column *CV* identified by a <column name> *CVN* explicitly or implicitly contained in the <view column list> of a <view definition> *V* with an action *PA* of REFERENCES or SELECT, then let *SP* be the set of privileges upon which *P* directly depends. For every table *T* identified by a <table reference> contained in the <query expression> of *V*, let *RT* be the <table name> of *T*. There exists a column *CT* whose <column name> is *CRT*, such that all of the following are true:

1) *CT* is a column of *T* and an underlying column of *CV*.

2) Every privilege descriptor *PD* that is the descriptor of some member of *SP* that specifies the action *PA* on *CRT* WITH GRANT OPTION is either an identified privilege descriptor for *CRT* or a modified privilege descriptor for *CRT*.

d) At least one of the following is true:

i) GRANT OPTION FOR is specified and the grantor of *P* is the special grantor value "_SYSTEM".

ii) There exists a path to *P* from an independent node that includes no identified or modified privilege descriptors. *P* is said to be a *marked modified privilege descriptor*.

iii) *P* directly depends on a marked modified privilege descriptor, and the grantor of *P* is the special grantor value "_SYSTEM". *P* is said to be a *marked modified privilege descriptor*.

13) A role authorization descriptor *D* is said to be *directly dependent* on another role authorization descriptor *RD* if all of the following conditions hold:

a) *RD* indicates that the role that it represents is grantable.

b) The role name of *D* is the same as the role name of *RD*.

c) The grantee of *RD* is the same as the grantor of *D* or the grantee of *RD* is PUBLIC, or, if the grantor of *D* is a <role name>, the grantee of *RD* belongs to the set of applicable roles of the grantor of *D*.

14) The *role dependency graph* is a directed graph such that all of the following are true:

a) Each node represents a role authorization descriptor.

b) Each arc from node *R1* to node *R2* represents the fact that *R2* directly depends on *R1*.

An independent node is one that has no incoming arcs.

15) A role authorization descriptor *RD* is said to be *abandoned* if it is not an independent node, and it is not itself an identified role authorization descriptor, and there exists no path to *RD* from any independent node other than paths that include an identified role authorization descriptor.

16) An arc from a node *P* to a node *D* of the privilege dependency graph is said to be *unsupported* if all of the following are true:

a) The grantor of *D* and the grantee of *P* are both <role name>s.

b) The destruction of all abandoned role authorization descriptors and, if ADMIN OPTION FOR is not specified, all identified role authorization descriptors would result in the grantor of *D* no longer having in its applicable roles the grantee of *P*.

17) A privilege descriptor *P* is *abandoned* if:

Case:

a) It is not an independent node, and *P* is not itself an identified or a modified privilege descriptor, and there exists no path to *P* from any independent node other than paths that include an identified privilege descriptor or a modified privilege descriptor or an unsupported arc and, if <revoke statement> specifies WITH HIERARCHY OPTION, then *P* has the WITH HIERARCHY OPTION.

b) All of the following conditions hold:

i) *P* is a column privilege descriptor that identifies a <column name> *CVN* explicitly or implicitly contained in the <view column list> of a <view definition> *V*, with an action *PA* of REFERENCES or SELECT.

ii) Letting *SP* be the set of privileges upon which *P* directly depends, at least one of the following is true:

**

1) There exists some table name *RT* such that all of the following are true:

   A) *RT* is the name of the table identified by some <table reference> contained in the <query expression> of *V*.

   B) For every column privilege descriptor *CPD* that is the descriptor of some member of *SP* that specifies the action *PA* on *RT*, *CPD* is either an identified privilege descriptor for *RT* or an abandoned privilege descriptor for *RT*.

2) There exists some column name *CRT* such that all of the following are true:

   A) *CRT* is the name of some column of the table identified by some <table reference> contained in the <query expression> of *V*.

   B) For every column privilege descriptor *CPD* that is the descriptor of some member of *SP* that specifies the action *PA* on *CRT*, *CPD* is either an identified privilege descriptor for *CRT* or an abandoned privilege descriptor for *CRT*.

18) The *revoke destruction action* is defined as

Case:

a) If the <revoke statement> is a <revoke privilege statement>, then

   Case:

   i) If the <revoke statement> specifies the WITH HIERARCHY OPTION, then the removal of the WITH HIERARCHY OPTION from all identified and abandoned privilege descriptors.

   ii) Otherwise, the destruction of all abandoned privilege descriptors and, if GRANT OPTION FOR is not specified, all identified privilege descriptors.

b) If the <revoke statement> is a <revoke role statement>, then the destruction of all abandoned role authorization descriptors, all abandoned privilege descriptors and, if GRANT OPTION FOR is not specified, all identified role authorization descriptors.

19) Let *S1* be the name of any schema and *A1* be the <authorization identifier> that owns the schema identified by *S1*.

20) Let *V* be any view descriptor included in *S1*. Let *QE* be the <query expression> of *V*. *V* is said to be *abandoned* if the revoke destruction action would result in *A1* no longer having in its applicable privileges all of the following:

a) SELECT privilege on at least one column of every table identified by a <table reference> is contained in *QE*.

b) SELECT privilege on every column identified by a <column reference> contained in *QE*.

c) USAGE privilege on every domain, every collation, every character set, and every transliteration whose names are contained in *QE*.

d) USAGE privilege on any user-defined type *UDT* such that some <data type> contained in *V* is usage-dependent on *UDT*.

e) EXECUTE privilege on every SQL-invoked routine that is the subject routine of any <routine invocation>, <method invocation>, <static method invocation>, or <method reference> that is contained in *QE*.

f) The table/method privilege on every table *T1* and every method *M* such that there is a <method reference> *MR* contained in *QE* such that *T1* is in the scope of the <value expression primary> of *MR* and *M* is subject routine of *MR*.

g) SELECT privilege on any column identified by a <column reference> contained in the <scalar subquery> that is equivalent to some <dereference operation> contained in *QE*.

h) SELECT privilege WITH HIERARCHY OPTION on at least one supertable of the scoped table of any <reference resolution> that is contained in *QE*.

i) SELECT privilege on the scoped table of any <reference resolution> that is contained in *QE*.

j) If *V* is the descriptor of a referenceable table, then USAGE privilege on the structured type associated with the view described by *V*.

k) UNDER privilege on every direct supertable of the view described by *V*.

l) SELECT privilege WITH HIERARCHY OPTION privilege on at least one supertable of every typed table identified by a <table reference> that simply contains an <only spec> and that is contained in *QE*.

21) Let *T* be any table descriptor included in *S1*. *T* is said to be *abandoned* if the revoke destruction action would result in *A1* no longer having all of the following:

a) If *T* is the descriptor of a referenceable table, then USAGE privilege on the structured type associated with the table described by *T*.

b) UNDER privilege on every direct supertable of the table described by *T*.

22) Let *TC* be any table constraint descriptor included in *S1*. *TC* is said to be *abandoned* if the revoke destruction action would result in *A1* no longer having in its applicable privileges all of the following:

a) REFERENCES privilege on at least one column of every table identified by a <table reference> contained in *TC*.

b) REFERENCES privilege on every column identified by a <column reference> contained in the <search condition> of *TC*.

c) USAGE privilege on every domain, every collation, every character set, and every transliteration whose names are contained in any <search condition> of *TC*.

d) USAGE privilege on any user-defined type *UDT* such that some <data type> contained in *TC* is usage-dependent on *UDT*.

e) EXECUTE privilege on every SQL-invoked routine that is the subject routine of any <routine invocation>, <method invocation>, <static method invocation>, or <method reference> that is contained in any <search condition> of *TC*.

f) The table/method privilege on every table *T1* and every method *M* such that there is a <method reference> *MR* contained in any <search condition> of *TC* such that *T1* is in the scope of the <value expression primary> of *MR* and *M* is the subject routine of *MR*.

g) SELECT privilege on any column identified by a <column reference> contained in the <scalar subquery> that is equivalent to some <dereference operation> contained in any <search condition> of *TC*.

h) SELECT privilege WITH HIERARCHY OPTION on at least one supertable of the scoped table of any <reference resolution> that is contained in any <search condition> of *TC*.

i) SELECT privilege on the scoped table of any <reference resolution> that is contained in any <search condition> of *TC*.

j) SELECT privilege WITH HIERARCHY OPTION on at least one supertable of every typed table identified by a <table reference> that simply contains an <only spec> and that is contained in *TC*.

23) Let *AX* be any assertion descriptor included in *S1*. *AX* is said to be *abandoned* if the revoke destruction action would result in *A1* no longer having in its applicable privileges all of the following:

a) REFERENCES privilege on at least one column of every table identified by a <table reference> contained in *AX*.

b) REFERENCES privilege on every column identified by a <column reference> contained in the <search condition> of *AX*.

c) USAGE privilege on every domain, every collation, every character set, and every transliteration whose names are contained in any <search condition> of *AX*.

d) USAGE privilege on any user-defined type *UDT* such that some <data type> contained in *AX* is usage-dependent on *UDT*.

e) EXECUTE privilege on every SQL-invoked routine that is the subject routine of any <routine invocation>, <method invocation>, <static method invocation>, or <method reference> that is contained in any <search condition> of *AX*.

f) The table/method privilege on every table *T1* and every method *M* such that there is a <method reference> *MR* contained in *AX* such that *T1* is in the scope of the <value expression primary> of *MR* and *M* is the subject routine of *MR*.

g) SELECT privilege on any column identified by a <column reference> contained in the <scalar subquery> that is equivalent to some <dereference operation> contained in any <search condition> of *AX*.

h) SELECT privilege WITH HIERARCHY OPTION on at least one supertable of the scoped table of any <reference resolution> that is contained in any <search condition> of *AX*.

i) SELECT privilege on the scoped table of any <reference resolution> that is contained in any <search condition> of *AX*.

j) SELECT privilege WITH HIERARCHY OPTION on at least one supertable of every typed table identified by a <table reference> that simply contains an <only spec> and that is contained in *AX*.

24) Let *TR* be any trigger descriptor included in *S1*. *TR* is said to be *abandoned* if the revoke destruction action would result in *A1* no longer having in its applicable privileges all of the following:

a) TRIGGER privilege on the subject table of *TR*.

b) REFERENCES privilege on at least one column of every table identified by a <table reference> contained in any <search condition> of *TR*.

c) SELECT privilege on every column identified by a <column reference> contained in any <search condition> of *TR*.

d) USAGE privilege on every domain, collation, character set, and transliteration whose name is contained in any <search condition> of *TR*.

e) USAGE privilege on any user-defined type *UDT* such that some <data type> contained in any <search condition> of *TR* is usage-dependent on *UDT*.

f) The table/method privilege on every table *T1* and every method *M* such that there is a <method reference> *MR* contained in any <search condition> of *TR* such that *T1* is in the scope of the <value expression primary> of *MR* and *M* is the subject routine of *MR*.

g) EXECUTE privilege on the SQL-invoked routine that is the subject routine of any <routine invocation>, <method invocation>, <static method invocation>, or <method reference> that is contained in any <search condition> of *TR*.

h) EXECUTE privilege on the SQL-invoked routine that is the subject routine of any <routine invocation>, <method invocation>, <static method invocation>, or <method reference> that is contained in the <triggered SQL statement> of *TR*.

i) SELECT privilege on at least one column of every table identified by a <table reference> contained in a <query expression> simply contained in a <cursor specification>, an <insert statement>, or a <merge statement> contained in the <triggered SQL statement> of *TR*.

j) SELECT privilege on at least one column of every table identified by a <table reference> contained in a <table expression> or <select list> immediately contained in a <select statement: single row> contained in the <triggered SQL statement> of *TR*.

k) SELECT privilege on at least one column of every table identified by a <table reference> and <column reference> contained in a <search condition> contained in a <delete statement: searched>, an <update statement: searched>, or a <merge statement> contained in the <triggered SQL statement> of *TR*.

l) SELECT privilege on at least one column of every table identified by a <table reference> and <column reference> contained in a <value expression> simply contained in a an <update source> or an <assigned row>contained in the <triggered SQL statement> of *TR*.

m) INSERT privilege on every column

   Case:

   i) Identified by a <column name> contained in the <insert column list> of an <insert statement> or a <merge statement> contained in the <triggered SQL statement> of *TR*.

   ii) Of the table identified by the <table name> immediately contained in an <insert statement> that does not contain an <insert column list> and that is contained in the <triggered SQL statement> of *TR*.

   iii) Of the table identified by the <target table> contained in a <merge statement> that contains a <merge insert specification> and that does not contain an <insert column list> and that is contained in the <triggered SQL statement> of *TR*.

n) UPDATE privilege on every column identified by a <column name> is contained in an <object column> contained in either an <update statement: positioned>, an <update statement: searched>, or a <merge statement> contained in the <triggered SQL statement> of *TR*.

o) DELETE privilege on every table identified by a <table name> contained in either a <delete statement: positioned> or a <delete statement: searched> contained in the <triggered SQL statement> of *TR*.

p) USAGE privilege on every domain, collation, character set, transliteration, and sequence generator whose name is contained in the <triggered SQL statement> of *TR*.

q) USAGE privilege on any user-defined type *UDT* such that some <data type> contained in the <triggered SQL statement> of *TR* is usage-dependent on *UDT*.

r) The table/method privilege on every table *T1* and every method *M* such that there is a <method reference> *MR* contained in any <triggered SQL statement> of *TR* such that *T1* is in the scope of the <value expression primary> of *MR* and *M* is the subject routine of *MR*.

s) SELECT privilege on any column identified by a <column reference> contained in the <scalar subquery> that is equivalent to some <dereference operation> contained in any of the following:

  i) A <search condition> of *TR*.

  ii) A <query expression> simply contained in a <cursor specification>, an <insert statement>, or a <merge statement> contained in the <triggered SQL statement> of *TR*.

  iii) A <table expression> or <select list> immediately contained in a <select statement: single row> contained in the <triggered SQL statement> of *TR*.

  iv) A <search condition> contained in a <delete statement: searched>, an <update statement: searched>, or a <merge statement> contained in the <triggered SQL statement> of *TR*.

  v) A <value expression> contained in an <update source> or an <assigned row> contained in the <triggered SQL statement> of *TR*.

t) SELECT privilege WITH HIERARCHY OPTION on at least one supertable of the scoped table of any <reference resolution> that is contained in any of the following:

  i) A <search condition> of *TR*.

  ii) A <query expression> simply contained in a <cursor specification>, an <insert statement>, or a <merge statement> contained in the <triggered SQL statement> of *TR*.

  iii) A <table expression> or <select list> immediately contained in a <select statement: single row> contained in the <triggered SQL statement> of *TR*.

  iv) A <search condition> contained in a <delete statement: searched>, an <update statement: searched>, or a <merge statement> contained in the <triggered SQL statement> of *TR*.

  v) A <value expression> contained in an <update source> or an <assigned row> contained in the <triggered SQL statement> of *TR*.

u) SELECT privilege on the scoped table of any <reference resolution> contained in any of the following:

  i) A <search condition> of *TR*.

  ii) A <query expression> simply contained in a <cursor specification>, an <insert statement>, or a <merge statement> contained in the <triggered SQL statement> of *TR*.

  iii) A <table expression> or <select list> immediately contained in a <select statement: single row> contained in the <triggered SQL statement> of *TR*.

iv) A <search condition> contained in a <delete statement: searched>, an <update statement: searched>, or a <merge statement> contained in the <triggered SQL statement> of *TR*.

v) A <value expression> contained in an <update source> or an <assigned row> contained in the <triggered SQL statement> of *TR*.

v) SELECT privilege WITH HIERARCHY OPTION on at least one supertable of every typed table identified by a <table reference> that simply contains an <only spec> and that is contained in the <triggered SQL statement> of *TR*.

25) Let *DC* be any domain constraint descriptor included in *S1*. *DC* is said to be *abandoned* if the revoke destruction action would result in *A1* no longer having in its applicable privileges all of the following:

a) REFERENCES privilege on at least one column of every table identified by a <table reference> contained in *TR*.

b) REFERENCES privilege on every column identified by a <column reference> contained in the <search condition> of *DC*.

c) USAGE privilege on every domain, every user-defined type, every collation, every character set, and every transliteration whose names are contained in any <search condition> of *DC*.

d) USAGE privilege on any user-defined type *UDT* such that some <data type> contained in any <search condition> of *DC* is usage-dependent on *UDT*.

e) EXECUTE privilege on every SQL-invoked routine that is the subject routine of any <routine invocation>, <method invocation>, <static method invocation>, or <method reference> that is contained in any <search condition> of *DC*.

f) The table/method privilege on every table *T1* and every method *M* such that there is a <method reference> *MR* contained in any <search condition> of *DC* such that *T1* is in the scope of the <value expression primary> of *MR* and *M* is the subject routine of *MR*.

g) SELECT privilege on any column identified by a <column reference> contained in a <scalar subquery> that is equivalent to some <dereference operation> contained in any <search condition> of *DC*.

h) SELECT privilege WITH HIERARCHY OPTION on at least one supertable of the scoped table of any <reference resolution> that is contained in any <search condition> of *DC*.

i) SELECT privilege on the scoped table of any <reference resolution> that is contained in contained in any <search condition> of *DC*.

j) SELECT privilege WITH HIERARCHY OPTION on at least one supertable of every typed table identified by a <table reference> that simply contains an <only spec> and that is contained in the <triggered SQL statement> of *TR*.

26) For every domain descriptor *DO* included in *S1*, *DO* is said to be *lost* if the revoke destruction action would result in *A1* no longer having in its applicable privileges USAGE privilege on every character set included in the data type descriptor included in *DO*.

27) For every table descriptor *TD* contained in *S1*, for every column descriptor *CD* included in *TD*, *CD* is said to be *lost* if any of the following are true:

a) The revoke destruction action would result in *A1* no longer having in its applicable privileges USAGE privilege on any character set included in the data type descriptor included in *CD*.

  b) The revoke destruction action would result in *A1* no longer having in its applicable privileges USAGE privilege on any user-defined type *UDT* such that a data type descriptor included in *CD* describes a type that is usage-dependent on *UDT*.

  c) The name of the domain *DN* included in *CD*, if any, identifies a lost domain descriptor and the revoke destruction action would result in *A1* no longer having in its applicable privileges USAGE privilege on any character set included in the data type descriptor of the domain descriptor of *DN*.

28) For every SQL-client module *MO*, let *G* be the <module authorization identifier> that owns *MO*. *MO* is said to be *lost* if the revoke destruction action would result in *G* no longer having in its applicable privileges USAGE privilege on the character set referenced in the <module character set specification> of *MO*.

29) For every user-defined type descriptor *DT* included in *S1*, *DT* is said to be *abandoned* if any of the following are true:

  a) The revoke destruction action would result in *A1* no longer having in its applicable privileges USAGE privilege on any user-defined type *UDT* such that a data type descriptor included in *DT* describes a type that is usage-dependent on *UDT*.

  b) The revoke destruction action would result in *A1* no longer having in its applicable privileges the UNDER privilege on any user-defined type that is a direct supertype of *DT*.

30) *S1* is said to be *lost* if the revoke destruction action would result in *A1* no longer having in its applicable privileges USAGE privilege on the default character set included in the *S1*.

31) For every collation descriptor *CN* contained in *S1*, *CN* is said to be *impacted* if the revoke destruction action would result in *A1* no longer having in its applicable privileges USAGE privilege on the collation whose name is contained in the <existing collation name> of *CN*.

32) For every character set descriptor *CSD* contained in *S1*, *CSD* is said to be *impacted* if the revoke destruction action would result in *A1* no longer having in its applicable privileges USAGE privilege on the collation whose name is contained in *CSD*.

33) For every descriptor included in *S1* that includes a data type descriptor *DTD*, *DTD* is said to be *impacted* if the revoke destruction action would result in *A1* no longer having, in its applicable privileges, USAGE privilege on the collation whose name is included in *DTD*.

34) Let *RD* be any routine descriptor with an SQL security characteristic of DEFINER that is included in *S1*. *RD* is said to be *abandoned* if the revoke destruction action would result in *A1* no longer having in its applicable privileges all of the following:

  a) EXECUTE privilege on the SQL-invoked routine that is the subject routine of any <routine invocation>, <method invocation>, <static method invocation>, or <method reference> that is contained in the <routine body> of *RD*.

  b) SELECT privilege on at least one column of each table identified by a <table reference> contained in a <query expression> simply contained in a <cursor specification>, an <insert statement>, or a <merge statement> contained in the SQL routine body of *RD*.

  c) SELECT privilege on at least one column of each table identified by a <table reference> contained in a <table expression> or <select list> immediately contained in a <select statement: single row> contained in the SQL routine body of *RD*.

d) SELECT privilege on at least one column of each table identified by a <table reference> contained in a <search condition> contained in a <delete statement: searched>, an <update statement: searched>, or a <merge statement> contained in the SQL routine body of *RD*.

e) SELECT privilege on at least one column of each table identified by a <table reference> contained in a <value expression> simply contained in an <update source> or an <assigned row> contained in the SQL routine body of *RD*.

f) SELECT privilege on at least one column identified by a <column reference> contained in a <search condition> contained in a <delete statement: searched>, an <update statement: searched>, or a <merge statement> contained in the <SQL routine body> of *RD*.

g) SELECT privilege on at least one column identified by a <column reference> contained in a <value expression> simply contained in an <update source> or an <assigned row> contained in the SQL routine body of *RD*.

h) INSERT privilege on each column

   Case:

   i) Identified by a <column name> contained in the <insert column list> of an <insert statement> or a <merge statement> contained in the SQL routine body of *RD*.

   ii) Of the table identified by the <table name> immediately contained in an <insert statement> that does not contain an <insert column list> and that is contained in the SQL routine body of *RD*.

   iii) Of the table identified by the <target table> immediately contained in a <merge statement> that contains a <merge insert specification> and that does not contain an <insert column list> and that is contained in the SQL routine body of *RD*.

i) UPDATE privilege on each column whose name is contained in an <object column> contained in either an <update statement: positioned>, an <update statement: searched>, or a <merge statement> contained in the SQL routine body of *RD*.

j) DELETE privilege on each table whose name is contained in a <table name> contained in either a <delete statement: positioned> or a <delete statement: searched> contained in the SQL routine body of *RD*.

k) USAGE privilege on each domain, collation, character set, transliteration, and sequence generator whose name is contained in the SQL routine body of *RD*.

l) USAGE privilege on each user-defined type *UDT* such that a declared type of any SQL parameter, returns data type, or result cast included in *RD* is usage-dependent on *UDT*.

m) USAGE privilege on each user-defined type *UDT* such that some <data type> contained in the SQL routine body of *RD* is usage-dependent on *UDT*.

n) The table/method privilege on every table *T1* and every method *M* such that there is a <method reference> *MR* contained in the SQL routine body of *RI* such that *T1* is in the scope of the <value expression primary> of *MR* and *M* is the subject routine of *MR*.

o) SELECT privilege on any column identified by a <column reference> contained in a <scalar subquery> that is equivalent to a <dereference operation> contained in any of the following:

i) A <query expression> simply contained in a <cursor specification>, an <insert statement>, or a <merge statement> contained in the <SQL routine body> of *RD*.

ii) A <table expression> or <select list> immediately contained in a <select statement: single row> contained in the <SQL routine body> of *RD*.

iii) A <search condition> contained in a <delete statement: searched>, an <update statement: searched>, or a <merge statement> contained in the <SQL routine body> of *RD*.

iv) A <value expression> contained in an <update source> or an <assigned row> contained in the <SQL routine body> of *RD*.

p) SELECT privilege WITH HIERARCHY OPTION on at least one supertable of the scoped table of any <reference resolution> that is contained in any of the following:

i) A <query expression> simply contained in a <cursor specification>, an <insert statement>, or a <merge statement> contained in the SQL routine body of *RD*.

ii) A <table expression> or <select list> immediately contained in a <select statement: single row> contained in the SQL routine body of *RD*.

iii) A <search condition> contained in a <delete statement: searched>, an <update statement: searched>, or a <merge statement> contained in the SQL routine body of *RD*.

iv) A <value expression> simply contained in an <update source> or an <assigned row> contained in the SQL routine body of *RD*.

q) SELECT privilege on the scoped table of any <reference resolution> that is contained in any of the following:

i) A <query expression> simply contained in a <cursor specification>, an <insert statement>, or a <merge statement> contained in the <SQL routine body> of *RD*.

ii) A <table expression> or <select list> immediately contained in a <select statement: single row> contained in the <SQL routine body> of *RD*.

iii) A <search condition> contained in a <delete statement: searched>, an <update statement: searched>, or a <merge statement> contained in the <SQL routine body> of *RD*.

iv) A <value expression> contained in an <update source> or an <assigned row> contained in the <SQL routine body> of *RD*.

r) SELECT privilege WITH HIERARCHY OPTION on at least one supertable of every typed table identified by a <table reference> that simply contains an <only spec> and that is contained in the <SQL routine body> of *RD*.

35) For every table descriptor *TD* included in *S1*, for every column descriptor *CD* included in *TD*, *CD* is said to be *contaminated* if *CD* includes one of the following:

a) A user-defined type descriptor that describes a supertype of a user-defined type described by an abandoned user-defined type descriptor.

b) A reference type descriptor that includes a user-defined type descriptor that describes a supertype of a user-defined type described by an abandoned user-defined type descriptor.

c) A collection type descriptor that includes a user-defined type descriptor that describes a supertype of a user-defined type described by an abandoned user-defined type descriptor.

d) A collection type descriptor that includes a reference type descriptor that includes a user-defined type descriptor that describes a supertype of a user-defined type described by an abandoned user-defined type descriptor.

36) If RESTRICT is specified, then there shall be no abandoned privilege descriptor, abandoned view, abandoned table constraint, abandoned assertion, abandoned domain constraint, lost domain, lost column, lost schema, and no descriptor that includes an impacted data type descriptor, impacted collation, impacted character set, abandoned user-defined type, forsaken column descriptor, forsaken domain descriptor, or abandoned routine descriptor.

37) If CASCADE is specified, then the impact on an SQL-client module that is determined to be a lost module is implementation-defined.

Access Rules

1) Case:

   a) If the <revoke statement> is a <revoke privilege statement>, then the applicable privileges for *A* shall include a privilege identifying *O*.

   b) If the <revoke statement> is a <revoke role statement>, then for every role *R* identified by a <role revoked>, the applicable roles of *A* shall include a role *AR* such that there exists a role authorization descriptor with role *R*, grantee *AR*, and the indication that the WITH ADMIN OPTION was granted.

General Rules

1) Case:

   a) If the <revoke statement> is a <revoke privilege statement>, then

      Case:

      i) If neither WITH HIERARCHY OPTION nor GRANT OPTION FOR is specified, then:

         1) All abandoned privilege descriptors are destroyed.

         2) The identified privilege descriptors are destroyed.

         3) The modified privilege descriptors are set to indicate that they are not grantable.

      ii) If WITH HIERARCHY OPTION is specified, then the WITH HIERARCHY OPTION is removed from all identified and abandoned privilege descriptors, if present.

      iii) If GRANT OPTION FOR is specified, then

         Case:

         1) If CASCADE is specified, then all abandoned privilege descriptors are destroyed.

**

2) Otherwise, if there are any privilege descriptors directly dependent on an identified privilege descriptor that are not modified privilege descriptors, then an exception condition is raised: *dependent privilege descriptors still exist.*

The identified privilege descriptors and the modified privilege descriptors are set to indicate that they are not grantable.

b) If the <revoke statement> is a <revoke role statement>, then:

i) If CASCADE is specified, then all abandoned role authorization descriptors are destroyed.

ii) All abandoned privilege descriptors are destroyed.

iii) Case:

1) If ADMIN OPTION FOR is specified, then the identified role authorization descriptors are set to indicate that they are not grantable.

2) If ADMIN OPTION FOR is not specified, then the identified role authorization descriptors are destroyed.

2) For every abandoned view descriptor *V*, let *S1.VN* be the <table name> of *V*. The following <drop view statement> is effectively executed without further Access Rule checking:

```
DROP VIEW S1.VN CASCADE
```

3) For every abandoned table descriptor *T*, let *S1.TN* be the <table name> of *T*. The following <drop table statement> is effectively executed without further Access Rule checking:

```
DROP TABLE S1.TN CASCADE
```

4) For every abandoned table constraint descriptor *TC*, let *S1.TCN* be the <constraint name> of *TC* and let *S2.T2* be the <table name> of the table that contains *TC* (*S1* and *S2* possibly equivalent). The following <alter table statement> is effectively executed without further Access Rule checking:

```
ALTER TABLE S2.T2 DROP CONSTRAINT S1.TCN CASCADE
```

5) For every abandoned assertion descriptor *AX*, let *S1.AXN* be the <constraint name> of *AX*. The following <drop assertion statement> is effectively executed without further Access Rule checking:

```
DROP ASSERTION S1.AXN CASCADE
```

6) For every abandoned trigger descriptor *TR*, let *S1.TRN* be the <trigger name> of *TR*. The following <drop trigger statement> is effectively executed without further Access Rule checking:

```
DROP TRIGGER S1.TRN
```

7) For every abandoned domain constraint descriptor *DC*, let *S1.DCN* be the <constraint name> of *DC* and let *S2.DN* be the <domain name> of the domain that contains *DC*. The following <alter domain statement> is effectively executed without further Access Rule checking:

```
ALTER DOMAIN S2.DN DROP CONSTRAINT S1.DCN
```

8) For every lost column descriptor *CD*, let *S1.TN* be the <table name> of the table whose descriptor includes the descriptor *CD* and let *CN* be the <column name> of *CD*. The following <alter table statement> is effectively executed without further Access Rule checking:

```
ALTER TABLE S1.TN DROP COLUMN CN CASCADE
```

9) For every lost domain descriptor *DO*, let *S1.DN* be the <domain name> of *DO*. The following <drop domain statement> is effectively executed without further Access Rule checking:

```
DROP DOMAIN S1.DN CASCADE
```

10) For every lost schema *S1*, the default character set of that schema is modified to include the name of the implementation-defined <character set specification> that would have been this schema's default character set had the <schema definition> not specified a <schema character set specification>.

11) If the object identified by *O* is a collation, let *OCN* be the name of that collation.

12) For every descriptor that includes an impacted data type descriptor *DTD*, *DTD* is modified such that it does not include *OCN*.

13) For every impacted collation descriptor *CD* with included collation name *CN*, the following <drop collation statement> is effectively executed without further Access Rule checking:

```
DROP COLLATION CN CASCADE
```

14) For every impacted character set descriptor *CSD* with included character set name *CSN*, *CSD* is modified so that the included collation name is the name of the default collation for the character set on which *CSD* is based.

15) For every abandoned user-defined type descriptor *DT* with <user-defined type name> *S1.DTN*, the following <drop data type statement> is effectively executed without further Access Rule checking:

```
DROP TYPE S1.DTN CASCADE
```

16) For every abandoned SQL-invoked routine descriptor *RD*, let *R* be the SQL-invoked routine whose descriptor is *RD*. Let *SN* be the <specific name> of *R*. The following <drop routine statement> is effectively executed without further Access Rule checking:

```
DROP SPECIFIC ROUTINE SN CASCADE
```

17) If the <revoke statement> is a <revoke privilege statement>, then:

    a) For every combination of <grantee> and <action> on *O* specified in <privileges>, if there is no corresponding privilege descriptor in the set of identified privilege descriptors, then a completion condition is raised: *warning — privilege not revoked.*

    b) If ALL PRIVILEGES was specified, then for each <grantee>, if no privilege descriptors were identified, then a completion condition is raised: *warning — privilege not revoked.*

18) For every contaminated column descriptor *CD*, let *S1.TN* be the <table name> of the table whose descriptor includes the descriptor *CD* and let *CN* be the <column name> of *CD*. The following <alter table statement> is effectively executed without further Access Rule checking:

```
ALTER TABLE S1.TN DROP COLUMN CN CASCADE
```

Conformance Rules

1) Without Feature T331, "Basic roles", conforming SQL language shall not contain a <revoke role statement>.
2) Without Feature F034, "Extended REVOKE statement", conforming SQL language shall not contain a <revoke statement> that contains a <drop behavior> that contains CASCADE.
3) Without Feature F034, "Extended REVOKE statement", conforming SQL language shall not contain a <revoke option extension> that contains GRANT OPTION FOR.
4) Without Feature F034, "Extended REVOKE statement", conforming SQL language shall not contain a <revoke statement> that contains a <privileges> that contains an <object name> where the owner of the SQL-schema that is specified explicitly or implicitly in the <object name> is not the current authorization identifier.
5) Without Feature F034, "Extended REVOKE statement", conforming SQL language shall not contain a <revoke statement> such that there exists a privilege descriptor *PD* that satisfies all the following conditions:
   a) *PD* identifies the object identified by <object name> simply contained in <privileges> contained in the <revoke statement>.
   b) *PD* identifies the <grantee> identified by any <grantee> simply contained in <revoke statement> and that <grantee> does not identify the owner of the SQL-schema that is specified explicitly or implicitly in the <object name> simply contained in <privileges> contained in the <revoke statement>.
   c) *PD* identifies the action identified by the <action> simply contained in <privileges> contained in the <revoke statement>.
   d) *PD* indicates that the privilege is grantable.
6) Without Feature S081, "Subtables", conforming SQL language shall not contain a <revoke option extension> that contains HIERARCHY OPTION FOR.

13 SQL-client modules

13.1 <SQL-client module definition>

Function

Define an SQL-client module.

Format

```
<SQL-client module definition> ::=
    <module name clause> <language clause> <module authorization clause>
    [ <module path specification> ]
    [ <module transform group specification> ]
    [ <module collations> ]
    [ <temporary table declaration>... ]
    <module contents>...

<module authorization clause> ::=
    SCHEMA <schema name>
  | AUTHORIZATION <module authorization identifier>
    [ FOR STATIC { ONLY | AND DYNAMIC } ]
  | SCHEMA <schema name> AUTHORIZATION <module authorization identifier>
    [ FOR STATIC { ONLY | AND DYNAMIC } ]

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

<module path specification> ::= <path specification>

<module transform group specification> ::= <transform group specification>

<module collations> ::= <module collation specification>...

<module collation specification> ::=
    COLLATION <collation name> [ FOR <character set specification list> ]

<character set specification list> ::=
    <character set specification> [ { <comma> <character set specification> }... ]

<module contents> ::=
    <declare cursor>
  | <dynamic declare cursor>
  | <externally-invoked procedure>
```

Syntax Rules

1) The <language clause> shall not specify SQL.

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

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

4) The implicit or explicit <catalog name> is the implicit <catalog name> for all unqualified <schema name>s in the <SQL-client module definition>.

5) If <module path specification> is not specified, then a <module path specification> containing an implementation-defined <schema name list> that contains the <schema name> contained in <module authorization clause> is implicit.

6) The explicit or implicit <catalog name> of each <schema name> contained in the <schema name list> of the <module path specification> shall be equivalent to the <catalog name> of the explicit or implicit <schema name> contained in <module authorization clause>.

7) The <schema name list> of the explicit or implicit <module path specification> is used as the SQL-path of the <SQL-client module definition>. The SQL-path is used to effectively qualify unqualified <routine name>s that are immediately contained in <routine invocation>s that are contained in the <SQL-client module definition>.

8) Case:

   a) If <module transform group specification> is not specified, then a <module transform group specification> containing a <multiple group specification> with a <group specification> *GS* for each <host parameter declaration> contained in <host parameter declaration list> of each <externally-invoked procedure> contained in <SQL-client module definition> whose <host parameter data type> *UDT* identifies a user-defined type with no <locator indication> is implicit. The <group name> of *GS* is implementation-defined and its <path-resolved user-defined type name> is *UDT*.

   b) If <module transform group specification> contains a <single group specification> with a <group name> *GN*, then a <module transform group specification> containing a <multiple group specification> that contains a <group specification> *GS* for each <host parameter declaration> contained in <host parameter declaration list> of each <externally-invoked procedure> contained in <SQL-client module definition> whose <host parameter data type> *UDT* identifies a user-defined type with no <locator indication> is implicit. The <group name> of *GS* is *GN* and its <path-resolved user-defined type name> is *UDT*.

   c) If <module transform group specification> contains a <multiple group specification> *MGS*, then a <module transform group specification> containing <multiple group specification> that contains *MGS* extended with a <group specification> *GS* for each <host parameter declaration> contained in <host parameter declaration list> of each <externally-invoked procedure> contained in <SQL-client module definition> whose <host parameter data type> *UDT* identifies a user-defined type with no <locator indication> and no equivalent of *UDT* is contained in any <group specification> contained in *MGS* is implicit. The <group name> of *GS* is implementation-defined and its <path-resolved user-defined type name> is *UDT*.

9) No two <character set specification>s contained in any <module collation specification> shall be equivalent.

10) A <module collation specification> *MCS* specifies the SQL-client module collation for one or more character sets for the SQL-client module. Let *CO* be the collation identified by the <collation name> contained in *MCS*.

Case:

a) If <character set specification list> is specified, then the collation specified by *CO* shall be applicable to every character set identified by a <character set specification> simply contained in the <module collation specification>. For each character set specified, the SQL-client module collation for that character set is set to *CO*.

b) Otherwise, the character sets for which the SQL-client module collation is set to *CO* are implementation-defined.

11) A <declare cursor> shall precede in the text of the <SQL-client module definition> any <externally-invoked procedure> or <SQL-invoked routine> that references the <cursor name> of the <declare cursor>.

12) A <dynamic declare cursor> shall precede in the text of the <SQL-client module definition> any <externally-invoked procedure> that references the <cursor name> of the <dynamic declare cursor>.

13) If neither FOR STATIC ONLY nor FOR STATIC AND DYNAMIC is specified, then FOR STATIC AND DYNAMIC is implicit.

14) For every <declare cursor> in an <SQL-client module definition>, the <SQL-client module definition> shall contain exactly one <open statement> that specifies the <cursor name> declared in the <declare cursor>.

NOTE 338 — See the Syntax Rules of Subclause 14.1, "<declare cursor>".

15) Let *EIP1* and *EIP2* be two <externally-invoked procedure>s contained in an <SQL-client module definition> that have the same number of <host parameter declaration>s and immediately contain a <fetch statement> referencing the same <cursor name>. Let n be the number of <host parameter declaration>s. Let $P1_i$, 1 (one) $\leq i \leq n$, be the i-th <host parameter declaration> of *EIP1*. Let $DT1_i$ be the <data type> contained in $P1_i$. Let $P2_i$ be the i-th <host parameter declaration> of *EIP2*. Let $DT2_i$ be the <data type> contained in $P2_i$. For each i, 1 (one) $\leq i \leq n$,

Case:

a) If $DT1_i$ and $DT2_i$ both identify a binary large object type, then either $P1_i$ and $P2_i$ shall both be binary large object locator parameters or neither shall be binary large object locator parameters.

b) If $DT1_i$ and $DT2_i$ both identify a character large object type, then either $P1_i$ and $P2_i$ shall both be character large object locator parameters or neither shall be character large object locator parameters.

c) If $DT1_i$ and $DT2_i$ both identify an array type, then either $P1_i$ and $P2_i$ shall both be array locator parameters or neither shall be array locator parameters.

d) If $DT1_i$ and $DT2_i$ both identify a multiset type, then either $P1_i$ and $P2_i$ shall both be multiset locator parameters or neither shall be multiset locator parameters.

e) If $DT1_i$ and $DT2_i$ both identify a user-defined type, then either $P1_i$ and $P2_i$ shall both be user-defined type locator parameters or neither shall be user-defined type locator parameters.

Access Rules

None.

General Rules

1) If the SQL-agent that performs a call of an <externally-invoked procedure> in an <SQL-client module definition> is not a program that conforms to the programming language standard for the programming language specified by the <language clause> of that <SQL-client module definition>, then the effect is implementation-dependent.

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

3) If FOR STATIC ONLY is specified, then the SQL-client module includes an indication that prepared statements resulting from execution of externally-invoked procedures included in that module have no owner.

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

   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. If the implementation choice is <commit statement>, then all holdable cursors are first closed. The determination of whether an SQL-agent has terminated unexpectedly is implementation-dependent.

   b) For every SQL descriptor area that is currently allocated within an SQL-session associated with the SQL-agent, let *D* be the <descriptor name> of that SQL descriptor area; a <deallocate descriptor statement> that specifies

      ```
      DEALLOCATE DESCRIPTOR D
      ```

      is effectively executed.

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

Conformance Rules

1) Without Feature S071, “SQL paths in function and type name resolution”, conforming SQL language shall not contain a <module path specification>.

2) Without Feature S241, “Transform functions”, conforming SQL language shall not contain a <module transform group specification>.

3) Without Feature F693, “SQL-session and client module collations”, conforming SQL language shall not contain a <module collation specification>.

4) Without Feature B051, “Enhanced execution rights”, conforming SQL language shall not contain a <module authorization clause> that immediately contains FOR STATIC ONLY or FOR STATIC AND DYNAMIC.

5) Without Feature B111, “Module language Ada”, conforming SQL language shall not contain an <SQL-client module definition> that contains a <language clause> that contains ADA.

6) Without Feature B112, “Module language C”, conforming SQL language shall not contain an <SQL-client module definition> that contains a <language clause> that contains C.

7) Without Feature B113, “Module language COBOL”, conforming SQL language shall not contain an <SQL-client module definition> that contains a <language clause> that contains COBOL.

8) Without Feature B114, “Module language Fortran”, conforming SQL language shall not contain an <SQL-client module definition> that contains a <language clause> that contains FORTRAN.

9) Without Feature B115, “Module language MUMPS”, conforming SQL language shall not contain an <SQL-client module definition> that contains a <language clause> that contains M.

10) Without Feature B116, “Module language Pascal”, conforming SQL language shall not contain an <SQL-client module definition> that contains a <language clause> that contains PASCAL.

11) Without Feature B117, “Module language PL/I”, conforming SQL language shall not contain an <SQL-client module definition> that contains a <language clause> that contains PLI.

13.2 <module name clause>

Function

Name an SQL-client module.

Format

```
<module name clause> ::=
    MODULE [ <SQL-client module name> ] [ <module character set specification> ]

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

Syntax Rules

1) If a <module name clause> does not specify an <SQL-client module name>, then the <SQL-client module definition> is unnamed.

2) The <SQL-client module name> shall not be equivalent to the <SQL-client module name> of any other <SQL-client module definition> in the same SQL-environment.

   NOTE 339 — An SQL-environment may have multiple <SQL-client module definition>s that are unnamed.

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

4) 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.

Access Rules

None.

General Rules

1) If <SQL-client module name> is specified, then, in the SQL-environment, the containing <SQL-client module definition> has the name given by <SQL-client module name>.

Conformance Rules

1) Without Feature F461, "Named character sets", conforming SQL language shall not contain a <module character set specification>.

13.3 <externally-invoked procedure>

Function

Define an externally-invoked procedure.

Format

```
<externally-invoked procedure> ::=
    PROCEDURE <procedure name> <host parameter declaration list> <semicolon>
    <SQL procedure statement> <semicolon>

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

<host parameter declaration> ::=
    <host parameter name> <host parameter data type>
  | <status parameter>

<host parameter data type> ::= <data type> [ <locator indication> ]

<status parameter> ::= SQLSTATE
```

Syntax Rules

1) The <procedure name> shall not be equivalent to the <procedure name> of any other <externally-invoked procedure> in the containing <SQL-client module definition>.

   NOTE 340 — The <procedure name> should be a standard-conforming procedure, function, or routine name of the language specified by the subject <language clause>. Failure to observe this recommendation will have implementation-dependent effects.

2) The <host parameter name> of each <host parameter declaration> in an <externally-invoked procedure> shall not be equivalent to the <host parameter name> of any other <host parameter declaration> in that <externally-invoked procedure>.

3) Any <host parameter name> contained in the <SQL procedure statement> of an <externally-invoked procedure> shall be specified in a <host parameter declaration> in that <externally-invoked procedure>.

4) If <locator indication> is simply contained in <host parameter declaration>, then:

   a) The declared type *T* identified by the <data type> immediately contained in <host parameter data type> shall be either binary large object type, character large object type, array type, multiset type, or user-defined type.

   b) If *T* is a binary large object type, then the host parameter identified by <host parameter name> is called a *binary large object locator parameter*.

   c) If *T* is a character large object type, then the host parameter identified by <host parameter name> is called a *character large object locator parameter*.

d) If *T* is an array type, then the host parameter identified by <host parameter name> is called an *array locator parameter.*

e) If *T* is a multiset type, then the host parameter identified by <host parameter name> is called a *multiset locator parameter.*

f) If *T* is a user-defined type, then the host parameter identified by <host parameter name> is called a *user-defined type locator parameter.*

5) A call of an <externally-invoked procedure> shall supply *n* arguments, where *n* is the number of <host parameter declaration>s in the <externally-invoked procedure>.

6) An <externally-invoked procedure> shall contain one <status parameter> referred to as an *SQLSTATE host parameter.* The SQLSTATE host parameter is referred to as a *status parameter.*

7) The Syntax Rules of Subclause 9.6, "Host parameter mode determination", with <host parameter declaration> as *PD* and <SQL procedure statement> as *SPS* for each <host parameter declaration>, are applied to determine whether the corresponding host parameter is an input host parameter, an output host parameter, or both an input host parameter and an output host parameter.

8) The Syntax Rules of Subclause 13.4, "Calls to an <externally-invoked procedure>", shall be satisfied.

Access Rules

None.

General Rules

1) An <externally-invoked procedure> defines an *externally-invoked procedure* that may be called by an SQL-agent.

2) If the <SQL-client module definition> that contains the <externally-invoked procedure> is associated with an SQL-agent that is associated with another <SQL-client module definition> that contains an <externally-invoked procedure> with equivalent <procedure name>s, then the effect is implementation-defined.

3) The language identified by the <language name> contained in the <language clause> of the <SQL-client module definition> that contains an <externally-invoked procedure> is the *caller language* of the <externally-invoked procedure>.

4) If the SQL-agent that performs a call of a <externally-invoked procedure> is not a program that conforms to the programming language standard specified by the caller language of the <externally-invoked procedure>, then the effect is implementation-dependent.

5) If the caller language of an <externally-invoked procedure> is ADA and the SQL-agent performs calls of <externally-invoked procedure>s from more than one Ada task, then the results are implementation-dependent.

6) If the <SQL-client module definition> that contains the <externally-invoked procedure> has an explicit <module authorization identifier> *MAI* that is not equivalent to the SQL-session <authorization identifier> *SAI*, then:

a) Whether or not *SAI* can invoke <externally-invoked procedure>s in an <SQL-client module definition> with explicit <module authorization identifier> *MAI* is implementation-defined, as are any restrictions pertaining to such invocation.

b) If *SAI* is restricted from invoking an <externally-invoked procedure> in an <SQL-client module definition> with explicit <module authorization identifier> *MAI*, then an exception condition is raised: *invalid authorization specification.*

7) If the value of any input host parameter provided by the SQL-agent falls outside the set of allowed values of the declared type of the host parameter, or if the value of any output host parameter resulting from the execution of the <externally-invoked procedure> falls outside the set of values supported by the SQL-agent for that host parameter, then the effect is implementation-defined. If the implementation-defined effect is the raising of an exception condition, then an exception condition is raised: *data exception — invalid parameter value.*

8) A copy of the top cell of the authorization stack is pushed onto the authorization stack. If the SQL-client module *M* that includes the externally-invoked procedure has an owner, then the top cell of the authorization stack is set to contain only the authorization identifier of the owner of *M*.

9) Let *S* be the <SQL procedure statement> of the <externally-invoked procedure>.

10) The General Rules of Subclause 13.5, "<SQL procedure statement>", are evaluated with *S* as the executing statement.

11) Upon completion of execution, the top cell in the authorization stack is removed.

Conformance Rules

1) Without Feature S231, "Structured type locators", conforming SQL language shall not contain a <host parameter data type> that simply contains a <data type> that specifies a structured type and that contains a <locator indication>.

2) Without Feature S232, "Array locators", conforming SQL language shall not contain a <host parameter data type> that simply contains an <array type> and that contains a <locator indication>.

3) Without Feature S233, "Multiset locators", conforming SQL language shall not contain a <host parameter data type> that simply contains a <multiset type> and that contains a <locator indication>.

13.4 Calls to an <externally-invoked procedure>

Function

Define the call to an <externally-invoked procedure> by an SQL-agent.

Syntax Rules

1) Let *n* be the number of <host parameter declaration>s in the <externally-invoked procedure> *EP* being called. Let PD_i, 1 (one) ≤ *i* ≤ *n*, be the *i*-th <host parameter declaration>. Let PDT_i be the <data type> contained in PD_i.

2) If the caller language of the <externally-invoked procedure> is ADA, then:

   a) The SQL-implementation shall generate the source code of an Ada library unit package *ALUP* the name of which shall be

      Case:

      i) If the <SQL-client module name> *SCMN* of the <SQL-client module definition> <SQL-client module name> is a valid Ada identifier, then equivalent to *SCMN*.

      ii) Otherwise, implementation-defined.

   b) For each <externally-invoked procedure> of the <SQL-client module definition>, there shall appear within *ALUP* a subprogram declaration declaring a procedure.

      i) If <procedure name> is a valid Ada identifier, then the name of that procedure *PN* shall be equivalent to <procedure name>; otherwise, *PN* shall be implementation-defined.

      ii) The parameters in each Ada procedure declaration *APD* shall appear in the same order as the <host parameter declaration>s of the corresponding <externally-invoked procedure> *EIP*. If the names of the parameters declared in the <host parameter declaration>s of *EIP* are valid Ada identifiers, then the parameters in *APD* shall have parameter names that are equivalent to the names of the corresponding parameters declared in the <host parameter declaration>s contained in *EIP*; otherwise, the parameters in *APD* shall parameter names that are implementation-defined

      iii) The parameter modes and subtype marks used in the parameter specifications are constrained by the remaining paragraphs of this Subclause.

   c) For each *i*, 1 (one) ≤ *i* ≤ *n*, PDT_i shall not identify a data type listed in the "SQL data type" column of Table 16, "Data type correspondences for Ada", for which the corresponding row in the "Ada data type" column is 'None'.

   d) The types of parameter specifications within the Ada subprogram declarations shall be taken from the library unit package `Interfaces.SQL` and its children `Numerics` and `Varying` and optional children `Adacsn` and `Adacsn.Varying`.

   e) The declaration of the library unit package `Interfaces.SQL` shall conform to the following template:

```
package Interfaces.SQL is
```

```
- The declarations of CHAR and NCHAR may be subtype declarations
  type CHAR is (See the Syntax Rules)
  type NCHAR is (See the Syntax Rules)
  type SMALLINT is range bs .. ts;
  type INT is range bi .. ti;
  type BIGINT is range bb .. tb;
  type REAL is digits dr;
  type DOUBLE_PRECISION is digits dd;
  type BOOLEAN is new Boolean;
  subtype INDICATOR_TYPE is t;
  type SQLSTATE_TYPE is new CHAR (1 .. 5);
  package SQLSTATE_CODES is
    AMBIGUOUS_CURSOR_NAME_NO_SUBCLASS:
      constant SQLSTATE_TYPE :="3C000";
    ATTEMPT_TO_ASSIGN_TO_NON_UPDATABLE_COLUMN_NO_SUBCLASS:
      constant SQLSTATE_TYPE := "0U000";
    ATTEMPT_TO_ASSIGN_TO_ORDERING_COLUMN_NO_SUBCLASS:
      constant SQLSTATE_TYPE := "0V000";
    CARDINALITY_VIOLATION_NO_SUBCLASS:
      constant SQLSTATE_TYPE :="21000";
    CLI_SPECIFIC_CONDITION_NO_SUBCLASS:
      constant SQLSTATE_TYPE :="HY000";
    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_TRANSACTION_RESOLUTION_UNKNOWN:
      constant SQLSTATE_TYPE :="08007";
    DATA_EXCEPTION_NO_SUBCLASS:
      constant SQLSTATE_TYPE :="22000";
    DATA_EXCEPTION_ARRAY_ELEMENT_ERROR:
      constant SQLSTATE_TYPE :="2202E";
    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_ESCAPE_CHARACTER_CONFLICT:
      constant SQLSTATE_TYPE :="2200B";
    DATA_EXCEPTION_INDICATOR_OVERFLOW:
      constant SQLSTATE_TYPE :="22022";
    DATA_EXCEPTION_INTERVAL_FIELD_OVERFLOW:
      constant SQLSTATE_TYPE :="22015";
    DATA_EXCEPTION_INTERVAL_VALUE_OUT_OF_RANGE:
      constant SQLSTATE_TYPE :="2200P";
```

```
DATA_EXCEPTION_INVALID_ARGUMENT_FOR_NATURAL_LOGARITHM:
  constant SQLSTATE_TYPE :="2201E";
DATA_EXCEPTION_INVALID_ARGUMENT_FOR_POWER_FUNCTION:
  constant SQLSTATE_TYPE :="2201F";
DATA_EXCEPTION_INVALID_ARGUMENT_FOR_WIDTH_BUCKET_FUNCTION:
  constant SQLSTATE_TYPE :="2201G";
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_OCTET:
  constant SQLSTATE_TYPE :="2200D";
DATA_EXCEPTION_INVALID_ESCAPE_SEQUENCE:
  constant SQLSTATE_TYPE :="22025";
DATA_EXCEPTION_INVALID_INDICATOR_PARAMETER_VALUE:
  constant SQLSTATE_TYPE :="22010";
DATA_EXCEPTION_INVALID_INTERVAL_FORMAT:
  constant SQLSTATE_TYPE :="22006";
DATA_EXCEPTION_INVALID_PARAMETER_VALUE:
  constant SQLSTATE_TYPE :="22023";
DATA_EXCEPTION_INVALID_PRECEDING_OR_FOLLOWING_SIZE_IN_WINDOW_FUNCTION:
  constant SQLSTATE_TYPE :="22013";
DATA_EXCEPTION_INVALID_REGULAR_EXPRESSION:
  constant SQLSTATE_TYPE :="2201B";
DATA_EXCEPTION_INVALID_REPEAT_ARGUMENT_IN_A_SAMPLE_CLAUSE:
  constant SQLSTATE_TYPE :="2202G";
DATA_EXCEPTION_INVALID_SAMPLE_SIZE:
  constant SQLSTATE_TYPE :="2202H";
DATA_EXCEPTION_INVALID_TIME_ZONE_DISPLACEMENT_VALUE:
  constant SQLSTATE_TYPE :="22009";
DATA_EXCEPTION_INVALID_USE_OF_ESCAPE_CHARACTER:
  constant SQLSTATE_TYPE :="2200C";
DATA_EXCEPTION_NULL_VALUE_NO_INDICATOR_PARAMETER:
  constant SQLSTATE_TYPE :="2200G";
DATA_EXCEPTION_MOST_SPECIFIC_TYPE_MISMATCH:
  constant SQLSTATE_TYPE :="22002";
DATA_EXCEPTION_MULTISET_VALUE_OVERFLOW:
  constant SQLSTATE_TYPE :="2200Q";
DATA_EXCEPTION_NONCHARACTER_IN_UCS_STRING:
  constant SQLSTATE_TYPE :="22029";
DATA_EXCEPTION_NULL_VALUE_NOT_ALLOWED:
  constant SQLSTATE_TYPE :="22004";
DATA_EXCEPTION_NULL_VALUE_SUBSTITUTED_FOR_MUTATOR_SUBJECT_PARAMETER:
  constant SQLSTATE_TYPE :="2202D";
DATA_EXCEPTION_NUMERIC_VALUE_OUT_OF_RANGE:
  constant SQLSTATE_TYPE :="22003";
DATA_EXCEPTION_SEQUENCE_GENERATOR_LIMIT_EXCEEDED:
  constant SQLSTATE_TYPE :="2200H";
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";
DATA_EXCEPTION_ZERO_LENGTH_CHARACTER_STRING:
  constant SQLSTATE_TYPE :="2200F";
DEPENDENT_PRIVILEGE_DESCRIPTORS_STILL_EXIST_NO_SUBCLASS:
  constant SQLSTATE_TYPE :="2B000";
DIAGNOSTICS_EXCEPTION_NO_SUBCLASS:
  constant SQLSTATE_TYPE :="0Z000";
DIAGNOSTICS_EXCEPTION_MAXIMUM_NUMBER_OF_DIAGNOSTICS_AREAS_EXCEEDED:
  constant SQLSTATE_TYPE :="0Z001";
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_DATETIME_INTERVAL_CODE:
  constant SQLSTATE_TYPE := "0700F";
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_DATA_TYPE_TRANSFORM_FUNCTION_VIOLATION:
  constant SQLSTATE_TYPE := "0700B";
DYNAMIC_SQL_ERROR_INVALID_DATA_TARGET:
  constant SQLSTATE_TYPE := "0700D";
DYNAMIC_SQL_ERROR_INVALID_LEVEL_VALUE:
  constant SQLSTATE_TYPE := "0700E";
DYNAMIC_SQL_ERROR_UNDEFINED_DATA_VALUE:
  constant SQLSTATE_TYPE := "0700C";
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";
EXTERNAL_ROUTINE_EXCEPTION_NO_SUBCLASS:
  constant SQLSTATE_TYPE :="38000";
EXTERNAL_ROUTINE_EXCEPTION_CONTAINING_SQL_NOT_PERMITTED:
  constant SQLSTATE_TYPE :="38001";
EXTERNAL_ROUTINE_EXCEPTION_MODIFYING_SQL_DATA_NOT_PERMITTED:
  constant SQLSTATE_TYPE :="38002";
EXTERNAL_ROUTINE_EXCEPTION_PROHIBITED_SQL_STATEMENT_ATTEMPTED:
  constant SQLSTATE_TYPE :="38003";
EXTERNAL_ROUTINE_EXCEPTION_READING_SQL_DATA_NOT_PERMITTED:
  constant SQLSTATE_TYPE :="38004";
EXTERNAL_ROUTINE_INVOCATION_EXCEPTION_NO_SUBCLASS:
  constant SQLSTATE_TYPE :="39000";
EXTERNAL_ROUTINE_INVOCATION_EXCEPTION_NULL_VALUE_NOT_ALLOWED:
  constant SQLSTATE_TYPE :="39004";
```

```
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";
INTEGRITY_CONSTRAINT_VIOLATION_RESTRICT_VIOLATION:
  constant SQLSTATE_TYPE :="23001";
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_COLLATION_NAME_NO_SUBCLASS:
  constant SQLSTATE_TYPE :="2H000";
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_GRANTOR_STATE_NO_SUBCLASS:
  constant SQLSTATE_TYPE :="0L000";
INVALID_ROLE_SPECIFICATION:
  constant SQLSTATE_TYPE :="0P000";
INVALID_SCHEMA_NAME_NO_SUBCLASS:
  constant SQLSTATE_TYPE :="3F000";
INVALID_SCHEMA_NAME_LIST_SPECIFICATION_NO_SUBCLASS:
  constant SQLSTATE_TYPE :="0E000";
INVALID_SQL_DESCRIPTOR_NAME_NO_SUBCLASS:
  constant SQLSTATE_TYPE :="33000";
INVALID_SQL_INVOKED_PROCEDURE_REFERENCE_NO_SUBCLASS:
  constant SQLSTATE_TYPE :="0M000";
INVALID_SQL_STATEMENT:
  constant SQLSTATE_TYPE :="30000";
INVALID_SQL_STATEMENT_IDENTIFIER_NO_SUBCLASS:
  constant SQLSTATE_TYPE :="30000";
INVALID_SQL_STATEMENT_NAME_NO_SUBCLASS:
  constant SQLSTATE_TYPE :="26000";
INVALID_TRANSFORM_GROUP_NAME_SPECIFICATION_NO_SUBCLASS:
  constant SQLSTATE_TYPE :="0S000";
INVALID_TRANSACTION_STATE_NO_SUBCLASS:
  constant SQLSTATE_TYPE :="25000";
INVALID_TRANSACTION_STATE_ACTIVE_SQL_TRANSACTION:
  constant SQLSTATE_TYPE :="25001";
INVALID_TRANSACTION_STATE_BRANCH_TRANSACTION_ALREADY_ACTIVE:
  constant SQLSTATE_TYPE :="25002";
INVALID_TRANSACTION_STATE_HELD_CURSOR_REQUIRES_SAME_ISOLATION_LEVEL:
  constant SQLSTATE_TYPE :="25008";
INVALID_TRANSACTION_STATE_INAPPROPRIATE_ACCESS_MODE_FOR_BRANCH_TRANSACTION:
  constant SQLSTATE_TYPE :="25003";
INVALID_TRANSACTION_STATE_INAPPROPRIATE_ISOLATION_LEVEL_FOR_BRANCH_TRANSACTION:
  constant SQLSTATE_TYPE :="25004";
INVALID_TRANSACTION_STATE_NO_ACTIVE_SQL_TRANSACTION_FOR_BRANCH_TRANSACTION:
  constant SQLSTATE_TYPE :="25005";
INVALID_TRANSACTION_STATE_READ_ONLY_SQL_TRANSACTION:
  constant SQLSTATE_TYPE :="25006";
INVALID_TRANSACTION_STATE_SCHEMA_AND_DATA_STATEMENT_MIXING_NOT_SUPPORTED:
  constant SQLSTATE_TYPE :="25007";
INVALID_TRANSACTION_INITIATION_NO_SUBCLASS:
  constant SQLSTATE_TYPE :="0B000";
INVALID_TRANSACTION_TERMINATION_NO_SUBCLASS:
  constant SQLSTATE_TYPE :="2D000";
LOCATOR_EXCEPTION_INVALID_SPECIFICATION:
  constant SQLSTATE_TYPE :="0F001";
LOCATOR_EXCEPTION_NO_SUBCLASS:
  constant SQLSTATE_TYPE :="0F000";
NO_DATA_NO_SUBCLASS:
  constant SQLSTATE_TYPE :="02000";
NO_DATA_NO_ADDITIONAL_DYNAMIC_RESULT_SETS_RETURNED:
  constant SQLSTATE_TYPE :="02001";
REMOTE_DATABASE_ACCESS_NO_SUBCLASS:
  constant SQLSTATE_TYPE :="HZ000";
SAVEPOINT_EXCEPTION_INVALID_SPECIFICATION:
  constant SQLSTATE_TYPE :="3B001";
SAVEPOINT_EXCEPTION_NO_SUBCLASS:
  constant SQLSTATE_TYPE :="3B000";
SAVEPOINT_EXCEPTION_TOO_MANY:
  constant SQLSTATE_TYPE :="3B002";
SQL_ROUTINE_EXCEPTION_NO_SUBCLASS:
  constant SQLSTATE_TYPE :="2F000";
SQL_ROUTINE_EXCEPTION_FUNCTION_EXECUTED_NO_RETURN_STATEMENT:
  constant SQLSTATE_TYPE :="2F005";
SQL_ROUTINE_EXCEPTION_MODIFYING_SQL_DATA_NOT_PERMITTED:
  constant SQLSTATE_TYPE :="2F002";
SQL_ROUTINE_EXCEPTION_PROHIBITED_SQL_STATEMENT_ATTEMPTED:
  constant SQLSTATE_TYPE :="2F003";
SQL_ROUTINE_EXCEPTION_READING_SQL_DATA_NOT_PERMITTED:
  constant SQLSTATE_TYPE :="2F004";
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";
TARGET_TABLE_DISAGREES_WITH_CURSOR_SPECIFICATION_NO_SUBCLASS:
  constant SQLSTATE_TYPE :="0T000";
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";
TRANSACTION_ROLLBACK_STATEMENT_COMPLETION_UNKNOWN:
  constant SQLSTATE_TYPE :="40003";
TRIGGERED_DATA_CHANGE_VIOLATION_NO_SUBCLASS:
      constant SQLSTATE_TYPE :="27000";
    WARNING_NO_SUBCLASS:
      constant SQLSTATE_TYPE :="01000";
    WARNING_ADDITIONAL_RESULT_SETS_RETURNED:
      constant SQLSTATE_TYPE :="0100D";
    WARNING_ARRAY_DATA_RIGHT_TRUNCATION:
      constant SQLSTATE_TYPE :="0102F";
    WARNING_ATTEMPT_TO_RETURN_TOO_MANY_RESULT_SETS:
      constant SQLSTATE_TYPE :="0100E";
    WARNING_CURSOR_OPERATION_CONFLICT:
      constant SQLSTATE_TYPE :="01001";
    WARNING_DEFAULT_VALUE_TOO_LONG_FOR_INFORMATION_SCHEMA:
      constant SQLSTATE_TYPE :="0100B";
    WARNING_DISCONNECT_ERROR:
      constant SQLSTATE_TYPE :="01002";
    WARNING_DYNAMIC_RESULT_SETS_RETURNED:
      constant SQLSTATE_TYPE :="0100C";
    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_QUERY_EXPRESSION_TOO_LONG_FOR_INFORMATION_SCHEMA:
      constant SQLSTATE_TYPE :="0100A";
    WARNING_SEARCH_CONDITION_TOO_LONG_FOR_INFORMATION_SCHEMA:
      constant SQLSTATE_TYPE :="01009";
    WARNING_STATEMENT_TOO_LONG_FOR_INFORMATION_SCHEMA:
      constant SQLSTATE_TYPE :="0100F";
    WARNING_STRING_DATA_RIGHT_TRUNCATION_WARNING:
      constant SQLSTATE_TYPE :="01004";
    WITH_CHECK_OPTION_VIOLATION_NO_SUBCLASS:
      constant SQLSTATE_TYPE :="44000";
  end SQLSTATE_CODES;
end Interfaces.SQL;
```

where *bs*, *ts*, *bi*, *ti*, *bb*, *tb*, *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.

NOTE 341 — The Ada identifier INVALID_SQL_STATEMENT appears for compatibility with earlier editions of ISO/IEC 9075. However, the intended symbol is INVALID_SQL_STATEMENT_IDENTIFIER_NO_SUBCLASS, which has been added in this edition of ISO/IEC 9075 to correspond correctly with the exception condition name.

f) The library unit package `Interfaces.SQL.Numerics` shall contain a sequence of decimal fixed point type declarations of the following form.

```
type Scale_s is delta 10.0 ** - s digits max_p;
```

where *s* is an integer ranging from 0 (zero) to an implementation-defined maximum value and *max_p* is an implementation-defined integer maximum precision.

g) The library unit package `Interfaces.SQL.Varying` shall contain type or subtype declarations with the defining identifiers CHAR and NCHAR.

h) Let *SQLcsn* be a <character set name> and let *Adacsn* be the result of replacing <period>'s in *SQLcsn* with <underscore>s. If *Adacsn* is a valid Ada identifier, then the library unit packages `Interfaces.SQL.`*`Adacsn`* and `Interfaces.SQL.`*`Adacsn`*`.Varying` shall contain a type or subtype declaration with defining identifier CHAR. If *Adacsn* is not a valid Ada identifier, then the names of these packages shall be implementation-defined.

i) `Interfaces.SQL` and its children may contain context clauses and representation items as needed. These packages may also contain declarations of Ada character types as needed to support the declarations of the types CHAR and NCHAR.

NOTE 342 — If the implementation-defined character set specification used by default with a fixed-length character string type is Latin1, then the declaration

```
subtype CHAR is String;
```

within `Interfaces.SQL` and the declaration

```
subtype CHAR is
    Ada.Strings.Unbounded.Unbounded_String;
```

within `Interfaces.SQL.Varying` (assuming the appropriate context clause) conform to the requirements of this paragraph of this Subclause. If the character set underlying NATIONAL CHARACTER is supported by an Ada package specification `Host_Char_Pkg` that declares a type `String_Type` that stores strings over the given character set, and furthermore the package specification `Host_Char_Pkg_Varying` (not necessary distinct from `Host_Char_Pkg`) declares a type `String_Type_Varying` that reproduces the functionality of `Ada.Strings.Unbounded.Unbounded_String` over the national character string type (rather than Latin1), then the declaration

```
subtype NCHAR is Host_Char_Pkg.String_Type;
```

within `Interfaces.SQL` and the declaration

```
subtype NCHAR is Host_Char_Pkg_Varying.String_Type_Varying;
```

within `Interfaces.SQL.Varying` conform to the requirements of this paragraph. Similar comments apply to other character sets and the packages `Interfaces.SQL.`*`Adacsn`* and `Interfaces.SQL.`*`Adacsn`*`.Varying`.

j) The library unit package `Interfaces.SQL` shall contain declarations of the following form:

```
package CHARACTER_SET renames Interfaces.SQL.Adacsn;
subtype CHARACTER_TYPE is CHARACTER_SET.cst;
```

where *cst* is a data type capable of storing a single character from the default character set. The package `Interfaces.SQL.`*`Adacsn`* shall contain the necessary declaration for *cst*.

NOTE 343 — If the default character set is Latin1, then a declaration of the form:

```
package CHARACTER_SET is
   subtype cst is Character;
end CHARACTER_SET;
```

may be substituted for the renaming declaration of CHARACTER_SET.

k) The base type of the SQLSTATE parameter shall be `Interfaces.SQL.SQLSTATE_TYPE`.

l) The Ada parameter mode of the SQLSTATE parameter is out.

m) If the *i*-th <host parameter declaration> specifies a <data type> that is:

i) CHARACTER(*L*) for some *L*, then the subtype mark in the *i*-th parameter declaration shall specify `Interfaces.SQL.CHAR`.

ii) CHARACTER VARYING(*L*) for some *L*, then the subtype mark in the *i*-th parameter declaration shall specify `Interfaces.SQL.VARYING.CHAR`.

iii) NATIONAL CHARACTER(*L*) for some *L*, then the subtype mark in the *i*-th parameter declaration shall specify `Interfaces.SQL.NCHAR`.

iv) NATIONAL CHARACTER VARYING(*L*) for some *L*, then the subtype mark in the *i*-th parameter declaration shall specify `Interfaces.SQL.VARYING.NCHAR`.

v) CHARACTER(*L*) CHARACTER SET *csn* for some *L* and some character set name *csn*, then the subtype mark in the *i*-th parameter declaration shall specify `Interfaces.SQL.`*`Adacsn`*`.CHAR`.

vi) CHARACTER VARYING(*L*) CHARACTER SET *csn* for some *L* and some character set name *csn*, then the subtype mark in the *i*-th parameter declaration shall specify `Interfaces.SQL.`*`Adacsn`*`.VARYING.CHAR`.

If *P* is an actual parameter associated with the *i*-th parameter in a call to the encompassing procedure, then *P* shall be sufficient to hold a character string of length L in the appropriate character set.

NOTE 344 — If a character set uses fixed length encodings then the definition of the subtype CHAR for fixed length strings may be an array type whose element type is an Ada character type. If that Ada character type is defined so as to use the number of bits per character used by the SQL encoding, then the restriction on P is precisely P'LENGTH = *L*. For variable length strings using fixed length encodings, if the definition of CHAR in the appropriate VARYING package is based on the type `Ada.Strings.Unbounded.Unbounded_String`, there is no restriction on *P*. Otherwise, a precise statement of the restriction on *P* is implementation-defined.

n) If the *i*-th <host parameter declaration> specifies a <data type> that is NUMERIC(*P*,*S*) for some <precision> *P* and <scale> *S*, then the Ada library unit package generated for the encompassing module shall contain a declaration equivalent to:

```
subtype Numeric_p_s is
  Interfaces.SQL.Numerics.Scale_s digits p;
```

The subtype mark in the *i*-th parameter specification shall specify this subtype.

o) If the *i*-th <host parameter declaration> specifies a <data type> that is SMALLINT, then the subtype mark in the *i*-th parameter declaration shall specify `Interfaces.SQL.SMALLINT`.

p) If the *i*-th <host parameter declaration> specifies a <data type> that is INTEGER, then the subtype mark in the *i*-th parameter declaration shall specify `Interfaces.SQL.INT`.

q) If the *i*-th <host parameter declaration> specifies a <data type> that is BIGINT, then the subtype mark in the *i*-th parameter declaration shall specify `Interfaces.SQL.BIGINT`.

r) If the *i*-th <host parameter declaration> specifies a <data type> that is REAL, then the subtype mark in the *i*-th parameter declaration shall specify `Interfaces.SQL.REAL`.

s) If the *i*-th <host parameter declaration> specifies a <data type> that is DOUBLE_PRECISION, then the subtype mark in the *i*-th parameter declaration shall specify `Interfaces.SQL.DOUBLE_PRECISION`.

t) For every parameter,

Case:

i) If the parameter is an input parameter but not an output parameter, then the Ada parameter mode is **in**.

ii) If the parameter is an output parameter but not an input parameter, then the Ada parameter mode is **out**.

iii) If the parameter is both an input parameter and an output parameter, then the Ada parameter mode is **in out**.

iv) Otherwise, the Ada parameter mode is **in**, **out**, or **in out**.

u) The following Ada library unit renaming declaration exists:

```
with Interfaces.SQL;
package SQL_Standard renames Interfaces.SQL.
```

3) If the caller language of the <externally-invoked procedure> is C, then:

a) The declared type of an SQLSTATE host parameter shall be C char with length 6.

b) For each i, 1 (one) $< i \leq n$, PDT_i shall not identify a data type listed in the "SQL data type" column of Table 17, "Data type correspondences for C", for which the corresponding row in the "C data type" column is 'None'.

c) For each i, 1 (one) $< i \leq n$, the type of the i-th host parameter shall be the data type listed in the "C data type" column of Table 17, "Data type correspondences for C", for which the corresponding row in the "SQL data type" column is PDT_i.

4) If the caller language of the <externally-invoked procedure> is COBOL, then:

a) The declared type of an SQLSTATE host parameter shall be COBOL PICTURE X(5).

b) For each i, 1 (one) $\leq i \leq n$, PDT_i shall not identify a data type listed in the "SQL data type" column of Table 18, "Data type correspondences for COBOL", for which the corresponding row in the "COBOL data type" column is 'None'.

c) For each i, 1 (one) $< i \leq n$, the type of the i-th host parameter shall be the data type listed in the "COBOL data type" column of Table 18, "Data type correspondences for COBOL", for which the corresponding row in the "SQL data type" column is PDT_i.

5) If the caller language of the <externally-invoked procedure> is FORTRAN, then:

a) The declared type of an SQLSTATE host parameter shall be Fortran CHARACTER with length 5.

b) For each i, 1 (one) $\leq i \leq n$, PDT_i shall not identify a data type listed in the "SQL data type" column of Table 19, "Data type correspondences for Fortran", for which the corresponding row in the "Fortran data type" column is 'None'.

c) For each i, 1 (one) $< i \leq n$, the type of the i-th host parameter shall be the data type listed in the "Fortran data type" column of Table 19, "Data type correspondences for Fortran", for which the corresponding row in the "SQL data type" column is PDT_i.

6) If the caller language of the <externally-invoked procedure> is M, then:

a) The declared type of an SQLSTATE host parameter shall be M character with maximum length greater than or equal to 5.

b) For each i, 1 (one) $\leq i \leq n$, PDT_i shall not identify a data type listed in the "SQL data type" column of Table 20, "Data type correspondences for M", for which the corresponding row in the "MUMPS data type" column is 'None'.

c) For each i, 1 (one) $< i \leq n$, the type of the i-th host parameter shall be the data type listed in the "MUMPS data type" column of Table 20, "Data type correspondences for M", for which the corresponding row in the "SQL data type" column is PDT_i.

7) If the caller language of the <externally-invoked procedure> is PASCAL, then:

a) The declared type of an SQLSTATE host parameter shall be Pascal PACKED ARRAY[1..5] OF CHAR.

b) For each i, 1 (one) $\leq i \leq n$, PDT_i shall not identify a data type listed in the "SQL data type" column of Table 21, "Data type correspondences for Pascal", for which the corresponding row in the "Pascal data type" column is 'None'.

c) For each i, 1 (one) $< i \leq n$, the type of the i-th host parameter shall be the data type listed in the "Pascal data type" column of Table 21, "Data type correspondences for Pascal", for which the corresponding row in the "SQL data type" column is PDT_i.

8) If the caller language of the <externally-invoked procedure> is PLI, then:

a) The declared type of an SQLSTATE host parameter shall be PL/I CHARACTER(5).

b) For each i, 1 (one) $\leq i \leq n$, PDT_i shall not identify a data type listed in the "SQL data type" column of Table 22, "Data type correspondences for PL/I", for which the corresponding row in the "PL/I data type" column is 'None'.

c) For each i, 1 (one) $< i \leq n$, the type of the i-th host parameter shall be the data type listed in the "PL/I data type" column of Table 22, "Data type correspondences for PL/I", for which the corresponding row in the "SQL data type" column is PDT_i.

Access Rules

None.

General Rules

1) Let *EP*, *PD*, *PN*, *DT*, and *PI* be a *PROC*, a *DECL*, a *NAME*, a *TYPE*, and an *ARG* specified in an application of the General Rules of this Subclause. Let *P* be the host parameter corresponding to *PD*.

2) If the General Rules of this Subclause are being applied for the evaluation of input parameters, and *P* is either an input host parameter or both an input host parameter and an output host parameter, then

   Case:

   a) If *DT* identifies a CHARACTER(*L*) or CHARACTER VARYING(*L*) data type and the caller language of *EP* is C, then a reference to *PN* is implicitly treated as a character string type value in the specified character set in which the octets of *PI* are the corresponding octets of that value.

      When such a reference is evaluated,

      Case:

      i) If *DT* identifies a CHARACTER(*L*) data type and some C character preceding the least significant C character of the value *PI* contains the implementation-defined null character that terminates a C character string, then the remaining characters of the value are set to <space>s.

      ii) If *DT* identifies a CHARACTER VARYING(*L*), then the length in characters of the value is set to the number of characters of PI_i that precede the implementation-defined null character that terminates a C character string.

      iii) If the least significant C character of the value *PI* does not contain the implementation-defined null character that terminates a C character string, then an exception condition is raised: *data exception — unterminated C string*; otherwise, that least significant C character does not correspond to any character in PI_i and is ignored.

   b) If *DT* identifies a CHARACTER(*L*) data type and the caller language of *EP* is either COBOL, FORTRAN, or PASCAL, or *DT* identifies a CHARACTER VARYING(*L*) data type and the caller language of *EP* is M, or *DT* identifies a CHARACTER(*L*) data type or CHARACTER VARYING(*L*) data type and the caller language of *EP* is PLI, then a reference to *PN* is implicitly treated as a character string type value in the specified character set in which the octets of *PI* are the corresponding octets of that value.

      NOTE 345 — In the preceding 2 Rules, the phrase "implementation-defined null character that terminates a C character string" implies one or more octets all of whose bits are zero and whose number is equal to the number of octets in the largest character of the character set of *DT*.

   c) If *DT* identifies INT, DEC, or REAL and the caller language of *EP* is M, then a reference to *PN* is implicitly treated as:

      ```
      CAST ( PI AS DT )
      ```

   d) If *DT* identifies a BOOLEAN type, then

      Case:

      i) If the caller language of *EP* is ADA, then if *PI* is False, then a reference to *PN* has the value <u>*False*</u>; otherwise, a reference to *PN* has the value <u>*True*</u>.

      ii) If the caller language of *EP* is C, then if *PI* is 0 (zero), then a reference to *PN* has the value <u>*False*</u>; otherwise, a reference to *PN* has the value <u>*True*</u>.

      iii) If the caller language of *EP* is COBOL, then if *PI* is 'F', then a reference to *PN* has the value <u>*False*</u>; otherwise, a reference to *PN* has the value <u>*True*</u>.

iv) If the caller language of *EP* is FORTRAN, then if *PI* is .FALSE., then a reference to *PN* has the value *False*; otherwise, a reference to *PN* has the value *True*.

v) If the caller language of *EP* is PLI, then if *PI* is '0'B, then a reference to *PN* has the value *False*; otherwise, a reference to *PN* has the value *True*.

NOTE 346 — Pascal has a Boolean-type whose values are *True* and *False*.

e) If *P* is a binary large object locator parameter, a character large object locator parameter, an array locator parameter, a multiset locator parameter, or a user-defined type locator parameter, then a reference to *PN* in a <general value specification> has the corresponding large object value, the large object character string value, the array value, the multiset value, or the user-defined type value, respectively, corresponding to *PI*.

f) If *DT* identifies a CHARACTER LARGE OBJECT or BINARY LARGE OBJECT type, then

Case:

i) If the caller language of *EP* is C, then a reference to *PN* is implicitly treated as

Case:

1) If *DT* identifies a CHARACTER LARGE OBJECT type, then a character string containing the *PN*.*PN*_length characters of *PN*.*PN*_data starting at character number 1 (one) in the same order that the characters appear in *PN*.*PN*_data.

2) If *DT* identifies a BINARY LARGE OBJECT type, then a binary large object string containing the *PN*.*PN*_length octets of *PN*.*PN*_data starting at octet number 1 (one) in the same order that the octets appear in *PN*.*PN*_data.

ii) If the caller language of *EP* is COBOL, then a reference to *PN* is implicitly treated as

Case:

1) If *DT* identifies a CHARACTER LARGE OBJECT type, then a character string containing the *PN*.*PN*_length characters of *PN*.*PN*_data starting at character number 1 (one) in the same order that the characters appear in *PN*.*PN*_data.

2) If *DT* identifies a BINARY LARGE OBJECT type, then a binary large object string containing the *PN*.*PN*_length octets of *PN*.*PN*_data starting at octet number 1 (one) in the same order that the octets appear in *PN*.*PN*_data.

iii) If the caller language of *EP* is FORTRAN, then a reference to *PN* is implicitly treated as

Case:

1) If *DT* identifies a CHARACTER LARGE OBJECT type, then a character string containing the *PN*.*PN*_length characters of *PN*.*PN*_data starting at character number 1 (one) in the same order that the characters appear in *PN*.*PN*_data.

2) If *DT* identifies a BINARY LARGE OBJECT type, then a binary large object string containing the *PN*.*PN*_length octets of *PN*.*PN*_data starting at octet number 1 (one) in the same order that the octets appear in *PN*.*PN*_data.

iv) If the caller language of *EP* is PLI, then a reference to *PN* is implicitly treated as

Case:

1) If *DT* identifies a CHARACTER LARGE OBJECT type, then a character string containing the *PN*.*PN*_length characters of *PN*.*PN*_data starting at character number 1 (one) in the same order that the characters appear in *PN*.*PN*_data.

2) If *DT* identifies a BINARY LARGE OBJECT type, then a binary large object string containing the *PN*.*PN*_length octets of *PN*.*PN*_data starting at octet number 1 (one) in the same order that the octets appear in *PN*.*PN*_data.

g) Otherwise, a reference to *PN* in a <general value specification> has the value *PI*.

3) If the General Rules of this Subclause are being applied for the evaluation of output parameters, and *P* is either an output host parameter or both an input host parameter and an output host parameter, then

Case:

a) If *DT* identifies CHARACTER(*L*) or CHARACTER VARYING(*L*) data types and the caller language of *EP* is C, then let *CL* be *k* greater than the maximum possible length in octets of *PN*, where *k* is the size in octets of the largest character in the character set of *DT*. A reference to *PN* that assigns some value *SV* to *PN* implicitly assigns a value that is an SQL CHARACTER(*CL*) data type in which octets of the value are the corresponding octets of SV_i, padded on the right with <space>s as necessary to reach the length *CL*, concatenated with a single implementation-defined null character that terminates a C character string.

b) If *DT* identifies a CHARACTER(*L*) data type and the caller language of *EP* is either COBOL, FORTRAN, or PASCAL, then let *CL* be the maximum possible length in octets of *PN*. A reference to *PN* that assigns some value *SV* to *PN* implicitly assigns a value that is an SQL CHARACTER(*CL*) data type in which octets of the value are the corresponding octets of *SV*, padded on the right with <space>s as necessary to reach the length *CL*.

c) If *DT* identifies a CHARACTER VARYING(*L*) data type and the caller language of *EP* is M, then a reference to *PN* that assigns some value *SV* to *PN* implicitly assigns a value that is an SQL CHARACTER VARYING(*ML*) data type in which octets of the value are the corresponding octets of *SV*, padded on the right with <space>s as necessary to reach the length *CL*. *ML* is the implementation-defined maximum length of variable-length character strings.

d) If *DT* identifies a CHARACTER(*L*) or CHARACTER VARYING(*L*) data types and the caller language of *EP* is PLI, then let *CL* be the maximum possible length in octets of *PN*. A reference to *PN* that assigns some value *SV* to *PN* implicitly assigns a value that is

Case:

i) If *DT* identifies CHARACTER(*L*), then an SQL CHARACTER(*CL*) data type.

ii) Otherwise, an SQL CHARACTER VARYING(*CL*) data type in which octets of the value are the corresponding octets of *SV*, padded on the right with <space>s as necessary to reach the length *CL*.

NOTE 347 — In the preceding 4 Rules, the phrase "implementation-defined null character that terminates a C character string" implies one or more octets all of whose bits are zero and whose number is equal to the number of octets in the largest character of the character set of *DT*.

e) If *DT* identifies INT, DEC, or REAL and the caller language of *EP* is M, then a reference to *PN* that assigns some value *SV* to *PN* implicitly assigns the value

```
CAST ( SV AS CHARACTER VARYING(ML) )
```

to *PI*, where *ML* is the implementation-defined maximum length of variable-length of character strings.

f) If *DT* identifies a BOOLEAN type, then

Case:

i) If the caller language of *EP* is ADA, then a reference to *PN* that assigns the value <u>*False*</u> to *PN* implicitly assigns the False to *PI*; a reference to *PN* that assigns the value <u>*True*</u> implicitly assigns the value True to *PI*.

ii) If the caller language of *EP* is C, then a reference to *PN* that assigns the value <u>*False*</u> to *PN* implicitly assigns the value 0 (zero) to *PI*; a reference to *PN* that assigns the value <u>*True*</u> implicitly assigns the value 1 (one) to *PI*.

iii) If the caller language of *EP* is COBOL, then a reference to *PN* that assigns the value <u>*False*</u> to *PN* implicitly assigns the value 'F' to *PI*; a reference to *PN* that assigns the value <u>*True*</u> implicitly assigns the value 'T' to *PI*.

iv) If the caller language of *EP* is FORTRAN, then a reference to *PN* that assigns the value <u>*False*</u> to *PN* implicitly assigns the value .FALSE. to *PI*; a reference to *PN* that assigns the value <u>*True*</u> implicitly assigns the value .TRUE. to *PI*.

v) If the caller language of *EP* is PLI, then a reference to *PN* that assigns the value <u>*False*</u> to *PN* implicitly assigns the value '0'B to *PI*; a reference to *PN* that assigns the value <u>*True*</u> implicitly assigns the value '1'B to *PI*.

NOTE 348 — Pascal has a Boolean-type, whose values are <u>*True*</u> and <u>*False*</u>.

g) If *P* is a binary large object locator parameter, a character large object locator parameter, an array locator parameter, a multiset locator parameter, or a user-defined type locator parameter, then a reference to *PN* that assigns some value *SV* to *PN* implicitly assigns the corresponding large object locator value, the character large object locator value, the array locator value, the multiset locator value, or the user-defined type locator value, respectively, that uniquely identifies *SV* to *PI*.

h) If *DT* identifies a CHARACTER LARGE OBJECT or BINARY LARGE OBJECT type, then

Case:

i) If the caller language of *EP* is C, then a reference to *PN* that assigns some value *SV* to *PN* implicitly assigns the value LENGTH(*SV*) to *PN*.*PN*_length and the value *SV* to *PN*.*PN*_data.

ii) If the caller language of *EP* is COBOL, then a reference to *PN* that assigns some value *SV* to *PN* implicitly assigns the value LENGTH(*SV*) to *PN*.*PN*-LENGTH and the value *SV* to *PN*.*PN*-DATA.

iii) If the caller language of *EP* is FORTRAN, then a reference to *PN* that assigns some value *SV* to *PN* implicitly assigns the value LENGTH(*SV*) to *PN*_LENGTH and the value *SV* to *PN*_DATA.

iv) If the caller language of *EP* is PLI, then a reference to *PN* that assigns some value *SV* to *PN* implicitly assigns the value LENGTH(*SV*) to *PN*.*PN*_length and the value *SV* to *PN*.*PN*_data.

i) Otherwise, a reference to *PN* that assigns some value *SV* to *PN* implicitly assigns the value *SV* to *PI*. If the caller language of *EP* is ADA and no value has been assigned to *PI*, then an implementation-dependent value is assigned to *PI*.

Conformance Rules

None.

13.5 <SQL procedure statement>

Function

Define all of the SQL-statements that are <SQL procedure statement>s.

Format

```
<SQL procedure statement> ::= <SQL executable statement>

<SQL executable statement> ::=
    <SQL schema statement>
  | <SQL data statement>
  | <SQL control statement>
  | <SQL transaction statement>
  | <SQL connection statement>
  | <SQL session statement>
  | <SQL diagnostics statement>
  | <SQL dynamic statement>

<SQL schema statement> ::=
    <SQL schema definition statement>
  | <SQL schema manipulation statement>

<SQL schema definition statement> ::=
    <schema definition>
  | <table definition>
  | <view definition>
  | <SQL-invoked routine>
  | <grant statement>
  | <role definition>
  | <domain definition>
  | <character set definition>
  | <collation definition>
  | <transliteration definition>
  | <assertion definition>
  | <trigger definition>
  | <user-defined type definition>
  | <user-defined cast definition>
  | <user-defined ordering definition>
  | <transform definition>
  | <sequence generator definition>

<SQL schema manipulation statement> ::=
    <drop schema statement>
  | <alter table statement>
  | <drop table statement>
  | <drop view statement>
  | <alter routine statement>
  | <drop routine statement>
  | <drop user-defined cast statement>
  | <revoke statement>
  | <drop role statement>
```

```
    | <alter domain statement>
    | <drop domain statement>
    | <drop character set statement>
    | <drop collation statement>
    | <drop transliteration statement>
    | <drop assertion statement>
    | <drop trigger statement>
    | <alter type statement>
    | <drop data type statement>
    | <drop user-defined ordering statement>
    | <alter transform statement>
    | <drop transform statement>
    | <alter sequence generator statement>
    | <drop sequence generator statement>

<SQL data statement> ::=
      <open statement>
    | <fetch statement>
    | <close statement>
    | <select statement: single row>
    | <free locator statement>
    | <hold locator statement>
    | <SQL data change statement>

<SQL data change statement> ::=
      <delete statement: positioned>
    | <delete statement: searched>
    | <insert statement>
    | <update statement: positioned>
    | <update statement: searched>
    | <merge statement>

<SQL control statement> ::=
      <call statement>
    | <return statement>

<SQL transaction statement> ::=
      <start transaction statement>
    | <set transaction statement>
    | <set constraints mode statement>
    | <savepoint statement>
    | <release savepoint statement>
    | <commit statement>
    | <rollback statement>

<SQL connection statement> ::=
      <connect statement>
    | <set connection statement>
    | <disconnect statement>

<SQL session statement> ::=
      <set session user identifier statement>
    | <set role statement>
    | <set local time zone statement>
    | <set session characteristics statement>
    | <set catalog statement>
    | <set schema statement>
    | <set names statement>
    | <set path statement>
    | <set transform group statement>
    | <set session collation statement>

<SQL diagnostics statement> ::= <get diagnostics statement>

<SQL dynamic statement> ::=
      <SQL descriptor statement>
    | <prepare statement>
    | <deallocate prepared statement>
    | <describe statement>
    | <execute statement>
    | <execute immediate statement>
    | <SQL dynamic data statement>

<SQL dynamic data statement> ::=
      <allocate cursor statement>
    | <dynamic open statement>
    | <dynamic fetch statement>
    | <dynamic close statement>
    | <dynamic delete statement: positioned>
    | <dynamic update statement: positioned>

<SQL descriptor statement> ::=
      <allocate descriptor statement>
    | <deallocate descriptor statement>
    | <set descriptor statement>
    | <get descriptor statement>
```

Syntax Rules

1) Let *S* be the <SQL procedure statement>.

2) An <SQL connection statement> shall not be generally contained in an <SQL control statement>.

3) The SQL-invoked routine specified by <SQL-invoked routine> shall be a schema-level routine.

   NOTE 349 — "schema-level routine" is defined in Subclause 11.50, "<SQL-invoked routine>".

4) *S* is *possibly non-deterministic* if and only if *S* is not an <SQL schema statement> and at least one of the following is satisfied:

   a) *S* is a <select statement: single row> that is possibly non-deterministic.

   b) *S* contains a <routine invocation> whose subject routine is an SQL-invoked routine that is possibly non-deterministic.

   c) *S* generally contains a <query specification> or a <query expression> that is possibly non-deterministic.

   d) *S* generally contains a <value expression> that is possibly non-deterministic.

Access Rules

None.

General Rules

1) Let *S* be the executing statement specified in an application of this Subclause.

   NOTE 350 — *S* is necessarily the innermost executing statement of the SQL-session as defined in Subclause 4.37, "SQL-sessions".

2) A *statement execution context NEWSEC* is established for the execution of *S*. Let *OLDSEC* be the most recent statement execution context. *NEWSEC* becomes the most recent statement execution context. *NEWSEC* is an atomic execution context, and therefore the most recent atomic execution context, if and only if *S* is an atomic SQL-statement.

3) If the non-dynamic or dynamic execution of an <SQL data statement>, <SQL dynamic data statement>, <dynamic select statement>, or <dynamic single row select statement> occurs within the same SQL-transaction as the non-dynamic or dynamic execution of an SQL-schema statement and this is not allowed by the SQL-implementation, then an exception condition is raised: *invalid transaction state — schema and data statement mixing not supported.*

4) Case:

   a) If *S* is immediately contained in an <externally-invoked procedure> *EP*, then let *n* be the number of <host parameter declaration>s specified in *EP*; let PD_i, 1 (one) ≤ *i* ≤ *n*, be the *i*-th such <host parameter declaration>; and let PN_i and DT_i be the <host parameter name> and <data type>, respectively, specified in PD_i. When *EP* is called by an SQL-agent, let PI_i be the *i*-th argument in the procedure call.

      Case:

      i) If *S* is an <SQL connection statement>, then:

         1) The SQL-client module that contains *S* is associated with the SQL-agent.

         2) The first diagnostics area is emptied.

         3) For each *i*, 1 (one) ≤ *i* ≤ *n*, the General Rules of Subclause 13.4, "Calls to an <externally-invoked procedure>", are evaluated for input parameters with *EP*, PD_i, PN_i, DT_i, and PI_i as *PROC*, *DECL*, *NAME*, *TYPE*, and *ARG*, respectively.

         4) The General Rules of *S* are evaluated.

         5) For each *i*, 1 (one) ≤ *i* ≤ *n*, the General Rules of Subclause 13.4, "Calls to an <externally-invoked procedure>", are evaluated for output parameters with *EP*, PD_i, PN_i, DT_i, and PI_i as *PROC*, *DECL*, *NAME*, *TYPE*, and *ARG*, respectively.

         6) If *S* successfully initiated or resumed an SQL-session, then subsequent calls to an <externally-invoked procedure> and subsequent invocations of <direct SQL statement>s by the SQL-agent are associated with that SQL-session until the SQL-agent terminates the SQL-session or makes it dormant.

      ii) If *S* is an <SQL diagnostics statement>, then:

         1) The SQL-client module that contains *S* is associated with the SQL-agent.

2) For each i, 1 (one) $\leq i \leq n$, the General Rules of Subclause 13.4, "Calls to an <externally-invoked procedure>", are evaluated for input parameters with EP, PD_i, PN_i, DT_i, and PI_i as $PROC$, $DECL$, $NAME$, $TYPE$, and ARG, respectively.

3) The General Rules of S are evaluated.

4) For each i, 1 (one) $\leq i \leq n$, the General Rules of Subclause 13.4, "Calls to an <externally-invoked procedure>", are evaluated for output parameters with EP, PD_i, PN_i, DT_i, and PI_i as $PROC$, $DECL$, $NAME$, $TYPE$, and ARG, respectively.

iii) Otherwise:

1) If no SQL-session is current for the SQL-agent, then

Case:

A) If the SQL-agent has not executed an <SQL connection statement> and there is no default SQL-session associated with the SQL-agent, then the following <connect statement> is effectively executed:

```
CONNECT TO DEFAULT
```

B) If the SQL-agent has not executed an <SQL connection statement> and there is a default SQL-session associated with the SQL-agent, then the following <set connection statement> is effectively executed:

```
SET CONNECTION DEFAULT
```

C) Otherwise, an exception condition is raised: *connection exception — connection does not exist.*

2) Subsequent calls to an <externally-invoked procedure> and subsequent invocations of <direct SQL statement>s by the SQL-agent are associated with the SQL-session until the SQL-agent terminates the SQL-session or makes it dormant.

3) If an SQL-transaction is active for the SQL-agent, then S is associated with that SQL-transaction.

4) If no SQL-transaction is active for the SQL-agent and S is a transaction-initiating SQL-statement, then

A) An SQL-transaction is effectively initiated and associated with this call and with subsequent calls of any <externally-invoked procedure> by that SQL-agent and with this and subsequent invocations of <direct SQL statement>s by that SQL-agent until the SQL-agent terminates that SQL-transaction.

B) If S is not a <start transaction statement>, then

Case:

I) If a <set transaction statement> has been executed since the termination of the last SQL-transaction in the SQL-session, then the access mode, constraint mode, and isolation level of the SQL-transaction are set as specified by the <set transac-

tion statement>. If a <set constraints mode statement> *SCM* has been executed since the termination of the last SQL-transaction in the SQL-session, then the constraint modes of constraints specified in *SCM* are set as specified in *SCM*.

II) If a <set session characteristics statement> has been executed in the current SQL-session, then:

1) If that <set session characteristics statement> set the enduring transaction characteristics of access mode, then the access mode of the SQL-transaction is set to the specified access mode.

2) If that <set session characteristics statement> set the enduring transaction characteristics of isolation level, then the isolation level of the SQL-transaction is set to the specified isolation level.

3) The constraint modes for all constraints in the SQL-transaction are set to their initial state.

III) Otherwise, the access mode of that SQL-transaction is read-write, the constraint mode for all constraints in that SQL-transaction is immediate, and the isolation level of that SQL-transaction is SERIALIZABLE.

C) The SQL-transaction is associated with the SQL-session.

D) The <SQL-client module definition> that contains *S* is associated with the SQL-transaction.

5) The SQL-client module that contains *S* is associated with the SQL-agent.

6) If *S* contains an <SQL schema statement> and the access mode of the current SQL-transaction is read-only, then an exception condition is raised: *invalid transaction state*.

7) The first diagnostics area is emptied.

8) For each i, 1 (one) $\leq i \leq n$, the General Rules of Subclause 13.4, "Calls to an <externally-invoked procedure>", are evaluated for input parameters with EP, PD_i, PN_i, DT_i, and PI_i as *PROC*, *DECL*, *NAME*, *TYPE*, and *ARG*, respectively.

9) If *S* does not conform to the Syntax Rules and Access Rules of an <SQL procedure statement>, then an exception condition is raised: *syntax error or access rule violation*.

10) The General Rules of *S* are evaluated.

11) For each i, 1 (one) $\leq i \leq n$, the General Rules of Subclause 13.4, "Calls to an <externally-invoked procedure>", are evaluated for output parameters with EP, PD_i, PN_i, DT_i, and PI_i as *PROC*, *DECL*, *NAME*, *TYPE*, and *ARG*, respectively.

12) If *S* is a <select statement: single row> or a <fetch statement> and a completion condition is raised: *no data*, or an exception condition is raised, then the value of each PI_i for which PN_i is referenced in a <target specification> in *S* is implementation-dependent.

b) Otherwise:

i) If an SQL-transaction is active for the SQL-agent, then *S* is associated with that SQL-transaction.

ii) If no SQL-transaction is active for the SQL-agent and *S* is a transaction-initiating SQL-statement, then

1) An SQL-transaction is effectively initiated as follows.

Case:

A) If a <set transaction statement> has been executed since the termination of the last SQL-transaction in the SQL-session, then the access mode, constraint mode, and isolation level of the SQL-transaction are set as specified by the <set transaction statement>.

B) Otherwise, the access mode of that SQL-transaction is read-write, the constraint mode for all constraints in that SQL-transaction is immediate, and the isolation level of that SQL-transaction is SERIALIZABLE.

2) The SQL-transaction is associated with the SQL-session.

iii) If *S* is an <SQL schema statement> and the access mode of the current SQL-transaction is read-only, then an exception condition is raised: *invalid transaction state*.

iv) If *S* is not an <SQL diagnostics statement>, then the first diagnostics area is emptied.

5) Case:

a) If *S* is immediately contained in an <externally-invoked procedure>, then

Case:

i) If *S* executed successfully, then either a completion condition is raised: *successful completion*, or a completion condition is raised: *warning*, or a completion condition is raised: *no data*, as determined by the General Rules in this and other Subclauses of ISO/IEC 9075.

ii) If *S* did not execute successfully, then:

1) The status parameter is set to the value specified for the condition in Clause 23, "Status codes".

2) If *S* is not an <SQL control statement>, then all changes made to SQL-data or schemas by the execution of *S* are canceled.

b) Otherwise, the General Rules for *S* are evaluated.

Case:

i) If *S* executed successfully, then either a completion condition is raised: *successful completion*, or a completion condition is raised: *warning*, or a completion condition is raised: *no data*, as determined by the General Rules in this and other Subclauses of ISO/IEC 9075.

ii) Otherwise:

1) If *S* is not an <SQL control statement>, then all changes made to SQL-data or schemas by the execution of *S* are canceled.

2) The exception condition with which the execution of *S* completed is raised.

6) If *S* is not an <SQL diagnostics statement>, then diagnostics information resulting from the execution of *S* is placed into the first diagnostics area, causing the first condition area in the first diagnostics area to become occupied. Whether any other condition areas become occupied is implementation-defined.

7) If *NEWSEC* is atomic, then all savepoints established during its existence are destroyed.

8) *NEWSEC* ceases to exist and *OLDSEC* becomes the most recent statement execution context.

9) *S* ceases to be an executing statement.

NOTE 351 — The innermost executing statement, if any, is now the one that was the innermost executing statement that caused *S* to be executed.

Conformance Rules

None.

13.6 Data type correspondences

Function

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

NOTE 352 — These tables are referenced in Subclause 11.50, "<SQL-invoked routine>", for the definitions of external routines and in Subclause 10.4, "<routine invocation>", for the invocation of external routines.

In the following tables, let *P* be <precision>, *S* be <scale>, *L* be <length>, *T* be <time fractional seconds precision>, *Q* be <interval qualifier>, and *N* be the implementation-defined size of a structured type reference.

Tables

Table 16 — Data type correspondences for Ada

SQL Data Type	Ada Data Type
SQLSTATE	SQL_STANDARD.SQLSTATE_TYPE
CHARACTER (*L*)	SQL_STANDARD.CHAR, with P'LENGTH of *L*
CHARACTER VARYING (*L*)	*None*
CHARACTER LARGE OBJECT(*L*)	*None*
BINARY LARGE OBJECT(*L*)	*None*
NUMERIC(*P*,*S*)	*None*
DECIMAL(*P*,*S*)	*None*
SMALLINT	SQL_STANDARD.SMALLINT
INTEGER	SQL_STANDARD.INT
BIGINT	SQL_STANDARD.BIGINT
FLOAT(*P*)	*None*
REAL	SQL_STANDARD.REAL
DOUBLE PRECISION	SQL_STANDARD.DOUBLE_PRECISION
BOOLEAN	SQL_STANDARD.BOOLEAN
DATE	*None*

SQL Data Type	Ada Data Type
TIME(*T*)	*None*
TIMESTAMP(*T*)	*None*
INTERVAL(*Q*)	*None*
user-defined type	*None*
REF	SQL_STANDARD.CHAR with P'LENGTH of *N*
ROW	*None*
ARRAY	*None*
MULTISET	*None*

Table 17 — Data type correspondences for C

SQL Data Type	C Data Type
SQLSTATE	`char`, with length 6
CHARACTER (*L*)[3]	`char`, with length (*L*+1)**k*[1]
CHARACTER VARYING (*L*)[3]	`char`, with length (*L*+1)**k*[1]
CHARACTER LARGE OBJECT(*L*)	`struct { long hvn_reserved unsigned long hvn_length char hvn_data[L]; } hvn`[2] [3]
BINARY LARGE OBJECT(*L*)	`struct { long hvn_reserved unsigned long hvn_length char hvn_data[L]; } hvn`[2]
NUMERIC(*P*,*S*)	*None*
DECIMAL(*P*,*S*)	*None*
SMALLINT	pointer to `short`
INTEGER	pointer to `long`

SQL Data Type	C Data Type
BIGINT	pointer to `long long`
FLOAT(*P*)	*None*
REAL	pointer to `float`
DOUBLE PRECISION	pointer to `double`
BOOLEAN	pointer to `long`
DATE	*None*
TIME(*T*)	*None*
TIMESTAMP(*T*)	*None*
INTERVAL(*Q*)	*None*
user-defined type	*None*
REF	`char`, with length *N*
ROW	*None*
ARRAY	*None*
MULTISET	*None*

[1] For character set UTF16, as well as other implementation-defined character sets in which a code unit occupies two octets, *k* is the length in units of C **unsigned short** of the character encoded using the greatest number of such units in the character set; for character set UTF32, as well as other implementation-defined character sets in which a code unit occupies four octets, *k* is four; for other character sets, *k* is the length in units of C **char** of the character encoded using the greatest number of such units in the character set.

[2] *hvn* is the name of the host variable defined to correspond to the SQL data type

[3] For character set UTF16, as well as other implementation-defined character sets in which a code unit occupies two octets, **char** or **unsigned char** should be replaced with **unsigned short**; for character set UTF32, as well as other implementation-defined character sets in which a code unit occupies four octets, **char** or **unsigned char** should be replaced with **unsigned int**. Otherwise, **char** or **unsigned char** should be used.

Table 18 — Data type correspondences for COBOL

SQL Data Type	COBOL Data Type
SQLSTATE	PICTURE X(5)
CHARACTER (*L*)	PICTURE X(*L*)[3]

SQL Data Type	**COBOL Data Type**
CHARACTER VARYING (*L*)	*None*

CHARACTER LARGE OBJECT(*L*)	`01 hvn.`
	`   49 hvn-RESERVED PIC S9(9) USAGE IS BINARY.`
	`   49 hvn-LENGTH PIC S9(9) USAGE IS BINARY.`
	`   49 hvn-DATA PIC X(L)`[2] [3]`.`

BINARY LARGE OBJECT (*L*)	`01 hvn.`
	`   49 hvn-RESERVED PIC S9(9) USAGE IS BINARY.`
	`   49 hvn-LENGTH PIC S9(9) USAGE IS BINARY.`
	`   49 hvn-DATA PIC X(L)`[2]`.`

NUMERIC(*P*,*S*)	USAGE DISPLAY SIGN LEADING SEPARATE, with PICTURE as specified[1]
DECIMAL(*P*,*S*)	*None*
SMALLINT	PICTURE S9(*SPI*) USAGE BINARY, where *SPI* is implementation-defined
INTEGER	PICTURE S9(*PI*) USAGE BINARY, where *PI* is implementation-defined
BIGINT	PICTURE S9(*BPI*) USAGE BINARY, where *BPI* is implementation-defined
FLOAT(*P*)	*None*
REAL	*None*
DOUBLE PRECISION	*None*
BOOLEAN	PICTURE X
DATE	*None*
TIME(*T*)	*None*
TIMESTAMP(*T*)	*None*
INTERVAL(*Q*)	*None*
user-defined type	*None*
REF	alphanumeric with length *N*
ROW	*None*

SQL Data Type	COBOL Data Type
ARRAY	*None*
MULTISET	*None*

[1] 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 *P*--*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'.

[2] *hvn* is the name of the host variable defined to correspond to the SQL data type

[3] For character set UTF16, as well as other implementation-defined character sets in which a code unit occupies two octets, "PICTURE X(*L*)" should be replaced with "PICTURE N(*L*)". For character set UTF32, as well as other implementation-defined character sets in which a code unit occupies four octets, "PICTURE X(*L*)" should be replaced with "PICTURE (*????*)". Otherwise, "PICTURE X(*L*)" should be used.

Table 19 — Data type correspondences for Fortran

SQL Data Type	Fortran Data Type
SQLSTATE	CHARACTER, with length 5
CHARACTER (*L*)	CHARACTER[2], with length *L*
CHARACTER VARYING (*L*)	*None*

CHARACTER LARGE OBJECT(*L*)	`CHARACTER hvn(L+8)`
	`    INTEGER*4 hvn_RESERVED`
	`    INTEGER*4 hvn_LENGTH`
	`    CHARACTER hvn_DATA`
	`    EQUIVALENCE(hvn(5), hvn_LENGTH)`
	`    EQUIVALENCE(hvn(9), hvn_DATA)` [1] [2]

BINARY LARGE OBJECT(*L*)	`CHARACTER hvn(L+8)`
	`    INTEGER*4 hvn_RESERVED`
	`    INTEGER*4 hvn_LENGTH`
	`    CHARACTER hvn_DATA`
	`    EQUIVALENCE(hvn(5), hvn_LENGTH)`
	`    EQUIVALENCE(hvn(9), hvn_DATA)` [1]

NUMERIC(*P*,*S*)	*None*
DECIMAL(*P*,*S*)	*None*
SMALLINT	*None*
INTEGER	INTEGER

SQL Data Type	Fortran Data Type
BIGINT	*None*
FLOAT(*P*)	*None*
REAL	REAL
DOUBLE PRECISION	DOUBLE PRECISION
BOOLEAN	LOGICAL
DATE	*None*
TIME(*T*)	*None*
TIMESTAMP(*T*)	*None*
INTERVAL(*Q*)	*None*
user-defined type	*None*
REF	CHARACTER with length *N*
ROW	*None*
ARRAY	*None*
MULTISET	*None*

[1] *hvn* is the name of the host variable defined to correspond to the SQL data type
[2] For character set UTF16, as well as other implementation-defined character sets in which a code unit occupies more than one octet, "CHARACTER KIND=*n*" should be used; in this case, the value of *n* that corresponds to a given character set is implementation-defined. Otherwise, "CHARACTER" (without "KIND=*n*") should be used.

Table 20 — Data type correspondences for M

SQL Data Type	M Data Type
SQLSTATE	character, with maximum length at least 5
CHARACTER (*L*)	*None*
CHARACTER VARYING (*L*)	character with maximum length *L*
CHARACTER LARGE OBJECT(*L*)	*None*
BINARY LARGE OBJECT(*L*)	*None*

SQL Data Type	M Data Type
NUMERIC(*P*,*S*)	character
DECIMAL(*P*,*S*)	character
SMALLINT	*None*
INTEGER	character
BIGINT	*None*
FLOAT(*P*)	*None*
REAL	character
DOUBLE PRECISION	*None*
BOOLEAN	*None*
DATE	*None*
TIME(*T*)	*None*
TIMESTAMP(*T*)	*None*
INTERVAL(*Q*)	*None*
user-defined type	*None*
REF	character
ROW	*None*
ARRAY	*None*
MULTISET	*None*

Table 21 — Data type correspondences for Pascal

SQL Data Type	Pascal Data Type
SQLSTATE	PACKED ARRAY [1..5] OF CHAR
CHARACTER(1)	CHAR
CHARACTER (*L*), *L*>1	PACKED ARRAY [1..*L*] OF CHAR
CHARACTER VARYING (*L*)	*None*

SQL Data Type	**Pascal Data Type**
CHARACTER LARGE OBJECT(*L*)	*None*
BINARY LARGE OBJECT(*L*)	*None*
NUMERIC(*P*,*S*)	*None*
DECIMAL(*P*,*S*)	*None*
SMALLINT	*None*
INTEGER	INTEGER
BIGINT	*None*
FLOAT(*P*)	*None*
REAL	REAL
DOUBLE PRECISION	*None*
BOOLEAN	BOOLEAN
DATE	*None*
TIME(*T*)	*None*
TIMESTAMP(*T*)	*None*
INTERVAL(*Q*)	*None*
user-defined type	*None*
REF	PACKED ARRAY[1..*N*] OF CHAR
ROW	*None*
ARRAY	*None*
MULTISET	*None*

Table 22 — Data type correspondences for PL/I

SQL Data Type	**PL/I Data Type**
SQLSTATE	CHARACTER(5)

SQL Data Type	**PL/I Data Type**
CHARACTER (*L*)	CHARACTER(*L*)
CHARACTER VARYING (*L*)	CHARACTER VARYING(*L*)

CHARACTER LARGE OBJECT(*L*)	`DCL 01 hvn`
	`    49 hvn_reserved FIXED BINARY (31)`
	`    49 hvn_length FIXED BINARY (31)`
	`    49 hvn_data CHAR (n)`[1]`;`

BINARY LARGE OBJECT (*L*)	`DCL 01 hvn`
	`    49 hvn_reserved FIXED BINARY (31)`
	`    49 hvn_length FIXED BINARY (31)`
	`    49 hvn_data CHAR (n)`[1]`;`

NUMERIC(*P*,*S*)	*None*
DECIMAL(*P*,*S*)	FIXED DECIMAL (*P*,*S*)
SMALLINT	FIXED BINARY(*SPI*), where *SPI* is implementation-defined
INTEGER	FIXED BINARY(*PI*), where *PI* is implementation-defined
BIGINT	FIXED BINARY(*BPI*), where *BPI* is implementation-defined
FLOAT(*P*)	FLOAT BINARY (*P*)
REAL	*None*
DOUBLE PRECISION	*None*
BOOLEAN	BIT(1)
DATE	*None*
TIME(*T*)	*None*
TIMESTAMP(*T*)	*None*
INTERVAL(*Q*)	*None*
user-defined type	*None*
REF	CHARACTER VARYING(*N*)
ROW	*None*
ARRAY	*None*

SQL Data Type	PL/I Data Type
MULTISET	*None*
[1] *hvn* is the name of the host variable defined to correspond to the SQL data type	

Conformance Rules

None.

This page intentionally left blank.

14 Data manipulation

14.1 <declare cursor>

Function

Define a cursor.

Format

```
<declare cursor> ::=
    DECLARE <cursor name> [ <cursor sensitivity> ] [ <cursor scrollability> ] CURSOR
    [ <cursor holdability> ]
    [ <cursor returnability> ]
    FOR <cursor specification>

<cursor sensitivity> ::=
    SENSITIVE
  | INSENSITIVE
  | ASENSITIVE

<cursor scrollability> ::=
    SCROLL
  | NO SCROLL

<cursor holdability> ::=
    WITH HOLD
  | WITHOUT HOLD

<cursor returnability> ::=
    WITH RETURN
  | WITHOUT RETURN

<cursor specification> ::=
    <query expression> [ <order by clause> ] [ <updatability clause> ]

<updatability clause> ::=
    FOR { READ ONLY | UPDATE [ OF <column name list> ] }

<order by clause> ::= ORDER BY <sort specification list>
```

Syntax Rules

1) If a <declare cursor> is contained in an <SQL-client module definition> *M*, then:

   a) The <cursor name> shall not be equivalent to the <cursor name> of any other <declare cursor> in *M*.

   b) Any <host parameter name> contained in the <cursor specification> shall be defined in a <host parameter declaration> in the <externally-invoked procedure> that contains an <open statement> that specifies the <cursor name>.

   NOTE 353 — See the Syntax Rules of Subclause 13.1, "<SQL-client module definition>".

2) When <cursor name> is referenced in an <update statement: positioned>, no <object column> in the <set clause> shall identify a column that is specified in a <sort specification> of an <order by clause>.

3) Let *T* be the result of evaluating the <query expression> *QE* immediately contained in the <cursor specification>.

4) Let *CS* be the cursor specified by the <declare cursor>.

5) If <cursor sensitivity> is not specified, then ASENSITIVE is implicit.

6) *CS* is *sensitive* if SENSITIVE is specified, *insensitive* if INSENSITIVE is specified, and *asensitive* if ASENSITIVE is specified or implied.

7) If <cursor scrollability> is not specified, then NO SCROLL is implicit.

8) If <cursor holdability> is not specified, then WITHOUT HOLD is implicit.

9) If <cursor returnability> is not specified, then WITHOUT RETURN is implicit.

10) If <updatability clause> is not specified, then

   Case:

   a) If either INSENSITIVE, SCROLL, or ORDER BY is specified, or if *QE* is not a simply updatable table, then an <updatability clause> of READ ONLY is implicit.

   b) Otherwise, an <updatability clause> of FOR UPDATE without a <column name list> is implicit.

11) If an <updatability clause> of FOR UPDATE with or without a <column name list> is specified, then INSENSITIVE shall not be specified and *QE* shall be updatable.

12) If an <updatability clause> specifying FOR UPDATE is specified or implicit, then *CS* is *updatable*; otherwise, *CS* is *not updatable*.

13) If *CS* is updatable, then let *LUTN* be a <table name> that references the leaf underlying table *LUT* of *QE*. *LUTN* is an exposed <table or query name> whose scope is <updatability clause>.

14) If an <order by clause> is specified, then the cursor specified by the <cursor specification> is said to be an *ordered cursor*.

15) If WITH HOLD is specified, then the cursor specified by the <cursor specification> is said to be a *holdable cursor*.

16) If WITH RETURN is specified, then the cursor specified by the <cursor specification> is said to be a *result set cursor*.

   NOTE 354 — "result set cursor" is defined in Subclause 4.32, "Cursors".

17) *QE* is the *simply underlying table* of *CS*.

18) If an <order by clause> is specified, then:

a) Let *OBC* be the <order by clause>. Let *NSK* be the number of <sort specification>s in *OBC*. For each *i* between 1 (one) and *NSK*, let K_i be the <sort key> contained in the *i*-th <sort specification> in *OBC*.

b) Each K_i shall contain a <column reference> and shall not contain a <subquery> or a <set function specification>.

c) If *QE* is a <query expression body> that is a <query term> that is a <query primary> that is a <simple table> that is a <query specification>, then the <cursor specification> is said to be a *simple table query*.

d) Case:

   i) If <sort specification list> contains any <sort key> K_i that contains a column reference to a column that is not a column of *T*, then:

      1) The <cursor specification> shall be a simple table query.

      2) Let *TE* be the <table expression> immediately contained in the <query specification> *QS* contained in *QE*.

      3) Let *SL* be the <select list> of *QS*. Let *SLT* be obtained from *SL* by replacing each <column reference> with its fully qualified equivalent.

      4) Let *OBCT* be obtained from *OBC* by replacing each <column reference> that references a column of *TE* with its fully qualified equivalent; in the case of common column names, each common column name is regarded as fully qualified.

      5) For each *i* between 1 (one) and *NSK*, let KT_i be the <sort key> contained in the *i*-th <sort specification> contained in *OBCT*.

      6) For each *i* between 1 (one) and *NSK*, if KT_i has the same left normal form derivation as the <value expression> immediately contained in some <derived column> *DC* of *SLT*, then:

         NOTE 355 — "Left normal form derivation" is defined in Subclause 6.2, "Notation provided in this International Standard", in ISO/IEC 9075-1.

         A) Case:

            I) If *DC* simply contains an <as clause>, then let *CN* be the <column name> contained in the <as clause>.

            II) Otherwise, let *CN* be an implementation-dependent <column name> that is not equivalent to the explicit or implicit <column name> of any other <derived column> contained in *SLT*. Let *VE* be the <value expression> simply contained in *DC*. *DC* is replaced in *SLT* by

```
VE AS CN
```

         B) KT_i is replaced in *OBCT* by

```
CN
```

      7) Let *SCR* be the set of <column reference>s to columns of *TE* that remain in *OBCT* after the preceding transformation.

      8) Let *NSCR* be the number of <column reference>s contained in *SCR*. For each *j* between 1 (one) and *NCR*, let C_j be an enumeration of these <column reference>s.

9) Case:

A) If *NSCR* is 0 (zero), then let *SKL* be the zero-length string.

B) Otherwise:

I) *T* shall not be a grouped table.

II) *QS* shall not specify the <set quantifier> DISTINCT or directly contain one or more <set function specification>s.

III) Let *SKL* be the comma-separated list of <derived column>s:

$, C_1, C_2, \ldots, C_{NCR}$

The columns C_j are said to be *extended sort key columns*.

10) Let *ST* be the result of evaluating the <query specification>:

```
SELECT SLT SKL TE
```

11) Let *EOBC* be *OBCT*.

ii) Otherwise, let *ST* be *T* and let *EOBC* be *OBC*.

e) *ST* is said to be a *sort table*.

19) If an <updatability clause> of FOR UPDATE without a <column name list> is specified or implicit, then a <column name list> that consists of the <column name> of every column of *LUT* is implicit.

20) If an <updatability clause> of FOR UPDATE with a <column name list> is specified, then each <column name> in the <column name list> shall be the <column name> of a column of *LUT*.

Access Rules

None.

General Rules

1) If an <order by clause> is not specified, then the table specified by the <cursor specification> is *T* and the ordering of rows in *T* is implementation-dependent.

2) If an <order by clause> is specified, then the ordering of rows of the result is determined by the <sort specification list>. The result table specified by the <cursor specification> is *TS* with all extended sort key columns (if any) removed.

a) Let *TS* be the sort table.

b) Each <sort specification> contained in *EOBC* specifies the sort direction for the corresponding sort key EK_i. If DESC is not specified in the *i*-th <sort specification>, then the sort direction for EK_i is ascending and the applicable <comp op> is the <less than operator>. Otherwise, the sort direction for EK_i is descending and the applicable <comp op> is the <greater than operator>.

c) Let P be any row of TS and let Q be any other row of TS, and let PV_i and QV_i be the values of EK_i in these rows, respectively. The relative position of rows P and Q in the result is determined by comparing PV_i and QV_i according to the rules of Subclause 8.2, "<comparison predicate>", where the <comp op> is the applicable <comp op> for EK_i, with the following special treatment of null values. A sort key value that is null is considered equal to another sort key value that is null. Whether a sort key value that is null is considered greater or less than a non-null value is implementation-defined, but all sort key values that are null shall either be considered greater than all non-null values or be considered less than all non-null values. PV_i is said to *precede* QV_i if the value of the <comparison predicate> "PV_i <comp op> QV_i" is *True* for the applicable <comp op>. If PV_i and QV_i are not null and the result of "PV_i <comp op> QV_i" is *Unknown*, then the relative ordering of PV_i and QV_i is implementation-dependent.

d) In TS, the relative position of row P is before row Q if PV_n precedes QV_n for some n greater than 0 (zero) and less than or equal to the number of <sort specification>s and $PV_i = QV_i$ for all $i < n$. The relative order of two rows that are not distinct with respect to the <sort specification>s are implementation-dependent.

e) The result table specified by the <cursor specification> is TS with all extended sort key columns (if any) removed.

3) If WITH HOLD is specified and the cursor is in an open state when an SQL-transaction is terminated with a <commit statement>, then the cursor is not closed and remains open into the next SQL-transaction.

NOTE 356 — A holdable cursor that has been held open retains its position when the new SQL-transaction is initiated. However, even if the cursor is currently positioned on a row when the SQL-transaction is terminated, before either an <update statement: positioned> or a <delete statement: positioned> is permitted to reference that cursor in the new SQL-transaction, a <fetch statement> shall be issued against the cursor.

Conformance Rules

1) Without Feature T231, "Sensitive cursors", conforming SQL language shall not contain a <cursor sensitivity> that immediately contains SENSITIVE.

2) Without Feature F791, "Insensitive cursors", conforming SQL language shall not contain a <cursor sensitivity> that immediately contains INSENSITIVE.

3) Without Feature F791, "Insensitive cursors", or Feature T231, "Sensitive cursors", conforming SQL language shall not contain a <cursor sensitivity> that immediately contains ASENSITIVE.

4) Without Feature F431, "Read-only scrollable cursors", conforming SQL language shall not contain a <cursor scrollability>.

5) Without Feature T471, "Result sets return value", conforming SQL language shall not contain a <cursor returnability>.

6) Without Feature F831, "Full cursor update", conforming SQL language shall not contain an <updatability clause> that contains FOR UPDATE and that contains a <cursor scrollability>.

7) Without Feature F831, "Full cursor update", conforming SQL language shall not contain an <updatability clause> that specifies FOR UPDATE and that contains an <order by clause>.

8) Without Feature T551, “Optional key words for default syntax”, conforming SQL language shall not contain a <cursor holdability> that immediately contains WITHOUT HOLD.

14.2 <open statement>

Function

Open a cursor.

Format

```
<open statement> ::= OPEN <cursor name>
```

Syntax Rules

1) The containing <SQL-client module definition> shall contain a <declare cursor> *DC* whose <cursor name> is equivalent to the <cursor name> contained in the <open statement>. Let *CR* be the cursor specified by *DC.*

Access Rules

1) The Access Rules for the <query expression> simply contained in the <declare cursor> identified by the <cursor name> are applied.

General Rules

1) If *CR* is not in the closed state, then an exception condition is raised: *invalid cursor state.*
2) Let *S* be the <cursor specification> of cursor *CR.*
3) Cursor *CR* is opened in the following steps:
   a) A copy of *S* is effectively created in which:
      i) Each <target specification> is replaced by the value of the target.
      ii) Each <value specification> generally contained in *S* that is CURRENT_USER, CURRENT_ROLE, SESSION_USER, SYSTEM_USER, CURRENT_PATH, CURRENT_DEFAULT_TRANSFORM_GROUP, or CURRENT_TRANSFORM_GROUP_FOR_TYPE <path-resolved user-defined type name> is replaced by the value resulting from evaluation of CURRENT_USER, CURRENT_ROLE, SESSION_USER, SYSTEM_USER, CURRENT_PATH, CURRENT_DEFAULT_TRANSFORM_GROUP, or CURRENT_TRANSFORM_GROUP_FOR_TYPE <path-resolved user-defined type name>, respectively, with all such evaluations effectively done at the same instant in time.
      iii) Each <datetime value function> generally contained in *S* is replaced by the value resulting from evaluation of that <datetime value function>, with all such evaluations effectively done at the same instant in time.
   b) Let *T* be the table specified by the copy of *S.*

c) A table descriptor for *T* is effectively created.

d) The General Rules of Subclause 14.1, "<declare cursor>", are applied.

e) Case:

   i) If *S* specifies INSENSITIVE, then a copy of *T* is effectively created and cursor *CR* is placed in the open state and its position is before the first row of the copy of *T*.

   ii) Otherwise, cursor *CR* is placed in the open state and its position is before the first row of *T*.

4) If *CR* specifies INSENSITIVE, and the SQL-implementation is unable to guarantee that significant changes will be invisible through *CR* during the SQL-transaction in which *CR* is opened and every subsequent SQL-transaction during which it may be held open, then an exception condition is raised: *cursor sensitivity exception — request rejected.*

5) If *CR* specifies SENSITIVE, and the SQL-implementation is unable to guarantee that significant changes will be visible through *CR* during the SQL-transaction in which *CR* is opened, then an exception condition is raised: *cursor sensitivity exception — request rejected.*

NOTE 357 — The visibility of significant changes through a sensitive holdable cursor during a subsequent SQL-transaction is implementation-defined.

6) Whether an SQL-implementation is able to disallow significant changes that would not be visible through a currently open cursor is implementation-defined.

Conformance Rules

None.

14.3 <fetch statement>

Function

Position a cursor on a specified row of a table and retrieve values from that row.

Format

```
<fetch statement> ::=
    FETCH [ [ <fetch orientation> ] FROM ] <cursor name> INTO <fetch target list>

<fetch orientation> ::=
    NEXT
  | PRIOR
  | FIRST
  | LAST
  | { ABSOLUTE | RELATIVE } <simple value specification>

<fetch target list> ::=
    <target specification> [ { <comma> <target specification> }... ]
```

Syntax Rules

1) <fetch target list> shall not contain a <target specification> that specifies a <column reference>.

2) If the <fetch orientation> is omitted, then NEXT is implicit.

3) Let *DC* be the <declare cursor> denoted by the <cursor name> and let *T* be the table defined by the <cursor specification> of *DC*. Let *CR* be the cursor specified by *DC*.

4) If the implicit or explicit <fetch orientation> is not NEXT, then *DC* shall specify SCROLL.

5) If a <fetch orientation> that contains a <simple value specification> is specified, then the declared type of that <simple value specification> shall be exact numeric with a scale of 0 (zero).

6) Case:

   a) If the <fetch target list> contains a single <target specification> *TS* and the degree of table *T* is greater than 1 (one), then the declared type of *TS* shall be a row type.

      Case:

      i) If *TS* is an <SQL parameter reference>, then the Syntax Rules of Subclause 9.2, "Store assignment", apply to *TS* and the row type of table *T* as *TARGET* and *VALUE*, respectively.

      ii) Otherwise, the Syntax Rules of Subclause 9.1, "Retrieval assignment", apply to *TS* and the row type of table *T* as *TARGET* and *VALUE*, respectively.

   b) Otherwise:

i) The number of <target specification>s *NTS* in the <fetch target list> shall be the same as the degree of table *T*. The *i*-th <target specification>, 1 (one) ≤ *i*, ≤ *NTS*, in the <fetch target list> corresponds with the *i*-th column of table *T*.

ii) For *i* varying from 1 (one) to *NTS*, let $TS1_i$ be the *i*-th <target specification> in the <fetch target list> that is either an <SQL parameter reference> or a <target array element specification>, and let CS_i be the *i*-th column of table *T* that corresponds with the <target specification> in the <fetch target list>.

   Case:

   1) If $TS1_i$ contains a <simple value specification>, then the Syntax Rules of Subclause 9.2, "Store assignment", apply to an arbitrary site whose declared type is the declared type of $TS1_i$ and CS_i as *TARGET* and *VALUE*, respectively.

   2) Otherwise, the Syntax Rules of Subclause 9.2, "Store assignment", apply to $TS1_i$ and the corresponding column of table *T* as *TARGET* and *VALUE*, respectively.

iii) For each <target specification> $TS2_i$, 1 (one) ≤ *i*, ≤ *NTS*, that is a <host parameter specification>, the Syntax Rules of Subclause 9.1, "Retrieval assignment", apply to $TS2_i$ and the corresponding column of table *T*, as *TARGET* and *VALUE*, respectively.

iv) For each <target specification> $TS2_i$, 1 (one) ≤ *i*, ≤ *NTS*, that is an <embedded variable specification>, the Syntax Rules of Subclause 9.1, "Retrieval assignment", apply to $TS2_i$ and the corresponding column of table *T*, as *TARGET* and *VALUE*, respectively.

Access Rules

None.

General Rules

1) If cursor *CR* is not in the open state, then an exception condition is raised: *invalid cursor state*.

2) Case:

   a) If the <fetch orientation> contains a <simple value specification>, then let *J* be the value of that <simple value specification>.

   b) If the <fetch orientation> specifies NEXT or FIRST, then let *J* be +1.

   c) If the <fetch orientation> specifies PRIOR or LAST, then let *J* be –1.

3) Let T_t be a table of the same degree as *T*.

   Case:

   a) If the <fetch orientation> specifies ABSOLUTE, FIRST, or LAST, then let T_t contain all rows of *T*, preserving their order in *T*.

b) If the <fetch orientation> specifies NEXT or specifies RELATIVE with a positive value of *J*, then:

   i) If the table *T* identified by cursor *CR* is empty or if the position of *CR* is on or after the last row of *T*, then let T_t be a table of no rows.

   ii) If the position of *CR* is on a row *R* that is other than the last row of *T*, then let T_t contain all rows of *T* ordered after row *R*, preserving their order in *T*.

   iii) If the position of *CR* is before a row *R*, then let T_t contain row *R* and all rows of *T* ordered after row *R*, preserving their order in *T*.

c) If the <fetch orientation> specifies PRIOR or specifies RELATIVE with a negative value of *J*, then:

   i) If the table *T* identified by cursor *CR* is empty or if the position of *CR* is on or before the first row of *T*, then let T_t be a table of no rows.

   ii) If the position of *CR* is on a row *R* that is other than the first row of *T*, then let T_t contain all rows of *T* ordered before row *R*, preserving their order in *T*.

   iii) If the position of *CR* is before a row *R* that is not the first row of *T*, then let T_t contain row all rows of *T* ordered before row *R*, preserving their order in *T*.

   iv) If the position of *CR* is after the last row of *T*, then let T_t contain all rows of *T*, preserving their order in *T*.

d) If RELATIVE is specified with a zero value of *J*, then

   Case:

   i) If the position of *CR* is on a row of *T*, then let T_t be a table comprising that one row.

   ii) Otherwise, let T_t be an empty table.

4) Let *N* be the number of rows in T_t. If *J* is positive, then let *K* be *J*. If *J* is negative, then let *K* be *N*+*J*+1. If *J* is zero and ABSOLUTE is specified, then let *K* be zero; if *J* is zero and RELATIVE is specified, then let *K* be 1.

5) Case:

   a) If *K* is greater than 0 (zero) and not greater than *N*, then *CR* is positioned on the *K*-th row of T_t and the corresponding row of *T*. That row becomes the current row of *CR*.

   b) Otherwise, no SQL-data values are assigned to any targets in the <fetch target list>, and a completion condition is raised: *no data*.

      Case:

      i) If the <fetch orientation> specifies RELATIVE with *J* equal to zero, then the position of *CR* is unchanged.

      ii) If the <fetch orientation> implicitly or explicitly specifies NEXT, specifies ABSOLUTE or RELATIVE with *K* greater than *N*, or specifies LAST, then *CR* is positioned after the last row.

iii) Otherwise, the <fetch orientation> specifies PRIOR, FIRST, or ABSOLUTE or RELATIVE with *K* not greater than *N* and *CR* is positioned before the first row.

6) If a completion condition *no data* has been raised, then no further General Rules of this Subclause are applied.

7) Case:

a) If the <fetch target list> contains a single <target specification> *TS* and the degree of table *T* is greater than 1 (one), then the current row is assigned to *TS* and

Case:

i) If *TS* is an <SQL parameter reference>, then the General Rules of Subclause 9.2, "Store assignment", apply to *TS* and the current row as *TARGET* and *VALUE*, respectively.

ii) Otherwise, the General Rules of Subclause 9.1, "Retrieval assignment", are applied to *TS* and the current row as *TARGET* and *VALUE*, respectively.

b) Otherwise, if the <fetch target list> contains more than one <target specification>, then values from the current row are assigned to their corresponding targets identified by the <fetch target list>. The assignments are made in an implementation-dependent order. Let *TV* be a target and let *SV* denote its corresponding value in the current row of *CR*.

Case:

i) If *TV* is either an <SQL parameter reference> or a <target array element specification>, then for each <target specification> in the <fetch target list>, let TV_i be the *i*-th <target specification> in the <fetch target list> and let SV_i denote the *i*-th corresponding value in the current row of *CR*.

Case:

1) If <target array element specification> is specified, then

Case:

A) If the value of TV_i, denoted by *C*, is null, then an exception condition is raised: *data exception — null value in array target.*

B) Otherwise:

I) Let *N* be the maximum cardinality of *C*.

II) Let *M* be the cardinality of the value of *C*.

III) Let *I* be the value of the <simple value specification> immediately contained in TV_i.

IV) Let *EDT* be the element type of *C*.

V) Case:

1) If *I* is greater than zero and less than or equal to *M*, then the value of *C* is replaced by an array *A* with element type *EDT* and cardinality *M* derived as follows:

a) For *j* varying from 1 (one) to *I*–1 and from *I*+1 to *M*, the *j*-th element in *A* is the value of the *j*-th element in *C*.

b) The *I*-th element of *A* is set to the value of *SV*, denoted by SV_i, by applying the General Rules of Subclause 9.2, "Store assignment", to the *I*-th element of *A* and SV_i as *TARGET* and *VALUE*, respectively.

2) If *I* is greater than *M* and less than or equal to *N*, then the value of *C* is replaced by an array *A* with element type *EDT* and cardinality *I* derived as follows:

a) For *j* varying from 1 (one) to *M*, the *j*-th element in *A* is the value of the *j*-th element in *C*.

b) For *j* varying from *M*+1 to *I*, the *j*-th element in *A* is the null value.

c) The *I*-th element of *A* is set to the value of *SV*, denoted by SV_i, by applying the General Rules of Subclause 9.2, "Store assignment", to the *I*-th element of *A* and SV_i as *TARGET* and *VALUE*, respectively.

3) Otherwise, an exception condition is raised: *data exception — array element error.*

2) Otherwise, the General Rules of Subclause 9.2, "Store assignment", apply to TV_i and SV_i as *TARGET* and *VALUE*, respectively.

ii) If *TV* is a <host parameter name>, then the General Rules of Subclause 9.1, "Retrieval assignment", are applied to *TV* and *SV* as *TARGET* and *VALUE*, respectively.

iii) If *TV* is an <embedded variable specification>, then the General Rules of Subclause 9.1, "Retrieval assignment", are applied to *TV* and *SV* as *TARGET* and *VALUE*, respectively.

NOTE 358 — SQL parameters cannot have as their data types any row type.

8) If an exception condition occurs during the assignment of a value to a target, then the values of all targets are implementation-dependent and *CR* remains positioned on the current row.

NOTE 359 — It is implementation-dependent whether *CR* remains positioned on the current row when an exception condition is raised during the derivation of any <derived column>.

Conformance Rules

1) Without Feature F431, "Read-only scrollable cursors", a <fetch statement> shall not contain a <fetch orientation>.

14.4 <close statement>

Function

Close a cursor.

Format

```
<close statement> ::= CLOSE <cursor name>
```

Syntax Rules

None.

Access Rules

None.

General Rules

1) Let *CR* be the cursor identified by the <cursor name> immediately contained in the <close statement>.
2) If cursor *CR* is not in the open state, then an exception condition is raised: *invalid cursor state.*
3) Let *RS* be the result set of *CR*.
4) Cursor *CR* is placed in the closed state and the copy of the <cursor specification> of the <declare cursor> that specified *CR* is destroyed.
5) If *RS* was one of an ordered set of result sets *RRS* returned from an SQL-invoked procedure *SIP*, then:
   a) Let *RTN* be the number of result sets returned by *SIP*.
   b) Let *RSN* be the ordinal position of *RS* within *RRS*.
   c) Case:
      i) If *RSN* = *RTN*, then a completion condition is raised: *no data — no additional dynamic result sets returned.*
      ii) Otherwise:
         1) *CR* is opened on *RS* in ordinal position *RSN* + 1 and *CR* is positioned before the first row of *RS*.
         2) A completion condition is raised: *warning — additional result sets returned.*

Conformance Rules

None.

14.5 <select statement: single row>

Function

Retrieve values from a specified row of a table.

Format

```
<select statement: single row> ::=
    SELECT [ <set quantifier> ] <select list>
    INTO <select target list>
    <table expression>

<select target list> ::=
    <target specification> [ { <comma> <target specification> }... ]
```

Syntax Rules

1) <select target list> shall not contain a <target specification> that specifies a <column reference>.

2) Let *T* be the table defined by the <table expression>.

3) Case:

   a) If the <select target list> contains a single <target specification> *TS* and the degree of table *T* is greater than 1 (one), then the declared type of *TS* shall be a row type.

      Case:

      i) If *TS* is an <SQL parameter reference>, then the Syntax Rules of Subclause 9.2, "Store assignment", apply to *TS* and the row type of table *T* as *TARGET* and *VALUE*, respectively.

      ii) Otherwise, the Syntax Rules of Subclause 9.1, "Retrieval assignment", apply to *TS* and the row type of table *T* as *TARGET* and *VALUE*, respectively.

   b) Otherwise:

      i) The number of elements *NOE* in the <select list> shall be the same as the number of elements in the <select target list>. The *i*-th <target specification>, 1 (one) ≤ *i* ≤ *NOE*, in the <select target list> corresponds with the *i*-th element of the <select list>.

      ii) For *i* varying from 1 (one) to *NOE*, let TS_i be the *i*-th <target specification> in the <select target list> that is either an <SQL parameter reference> or a <target array element specification>, and let SL_i be the *i*-th element of the <select list> that corresponds to the <target specification> in the <select target list>.

         Case:

         1) If <target array element specification> is specified, then the Syntax Rules of Subclause 9.2, "Store assignment", apply to an arbitrary site whose declared type is the declared type of TS_i and SL_i as *TARGET* and *VALUE*, respectively.

2) Otherwise, the Syntax Rules of Subclause 9.2, "Store assignment", apply to TS_i and the corresponding element of the <select list>, as *TARGET* and *VALUE*, respectively.

iii) For each <target specification> *TS* that is a <host parameter specification>, the Syntax Rules of Subclause 9.1, "Retrieval assignment", apply to *TS* and the corresponding element of the <select list>, as *TARGET* and *VALUE*, respectively.

iv) For each <target specification> *TS* that is an <embedded variable specification>, the Syntax Rules of Subclause 9.1, "Retrieval assignment", apply to *TS* and the corresponding element of the <select list>, as *TARGET* and *VALUE*, respectively.

4) Let *S* be a <query specification> whose <select list> and <table expression> are those specified in the <select statement: single row> and that specifies the <set quantifier> if it is specified in the <select statement: single row>. *S* shall be a valid <query specification>.

5) A column in the result of the <select statement: single row> is known not null if the corresponding column in the result of *S* is known not null.

6) The <select statement: single row> is *possibly non-deterministic* if *S* is possibly non-deterministic.

Access Rules

None.

General Rules

1) Let *Q* be the result of <query specification> *S*.

2) Case:

   a) If the cardinality of *Q* is greater than 1 (one), then an exception condition is raised: *cardinality violation*. It is implementation-dependent whether or not SQL-data values are assigned to the targets identified by the <select target list>.

   b) If *Q* is empty, then no SQL-data values are assigned to any targets identified by the <select target list>, and a completion condition is raised: *no data*.

   c) Otherwise, values in the row of *Q* are assigned to their corresponding targets.

3) If a completion condition *no data* has been raised, then no further General Rules of this Subclause are applied.

4) Case:

   a) If the <select target list> contains a single <target specification> *TS* and the degree of table *T* is greater than 1 (one), then the current row is assigned to *TS* and

   Case:

   i) If *TS* is an <SQL parameter reference>, then the General Rules of Subclause 9.2, "Store assignment", apply to *TS* and the current row as *TARGET* and *VALUE*, respectively.

ii) Otherwise, the General Rules of Subclause 9.1, “Retrieval assignment”, are applied to *TS* and the current row as *TARGET* and *VALUE*, respectively.

b) Otherwise:

i) Let *NOE* be the number of elements in the <select list>.

ii) For *i* varying from 1 (one) to *NOE*, let TS_i be the *i*-th <target specification> in the <select target list> that is either an <SQL parameter reference> or a <target array element specification>, and let SL_i denote the corresponding (*i*-th) value in the row of *Q*. The assignment of values to targets in the <select target list> is in an implementation-dependent order.

Case:

1) If <target array element specification> is specified, then

Case:

A) If the value of TS_i, denoted by *C*, is null, then an exception condition is raised: *data exception — null value in array target.*

B) Otherwise:

I) Let *N* be the maximum cardinality of *C*.

II) Let *M* be the cardinality of the value of *C*.

III) Let *I* be the value of the <simple value specification> immediately contained in TS_i.

IV) Let *EDT* be the element type of *C*.

V) Case:

1) If *I* is greater than zero and less than or equal to *M*, then the value of *C* is replaced by an array *A* with element type *EDT* and cardinality *M* derived as follows:

a) For *j* varying from 1 (one) to *I*–1 and from *I*+1 to *M*, the *j*-th element in *A* is the value of the *j*-th element in *C*.

b) The *I*-th element of *A* is set to the value of *SL*, denoted by SL_i, by applying the General Rules of Subclause 9.2, “Store assignment”, to the *I*-th element of *A* and SL_i as *TARGET* and *VALUE*, respectively.

2) If *I* is greater than *M* and less than or equal to *N*, then the value of *C* is replaced by an array *A* with element type *EDT* and cardinality *I* derived as follows:

a) For *j* varying from 1 (one) to *M*, the *j*-th element in *A* is the value of the *j*-th element in *C*.

b) For *j* varying from *M*+1 to *I*, the *j*-th element in *A* is the null value.

c) The *I*-th element of *A* is set to the value of *SL*, denoted by SL_i, by applying the General Rules of Subclause 9.2, "Store assignment", to the *I*-th element of *A* and SL_i as *TARGET* and *VALUE*, respectively.

3) Otherwise, an exception condition is raised: *data exception — array element error*.

2) Otherwise, the corresponding value SL_i in the row of *Q* is assigned to TS_i according to the General Rules of Subclause 9.2, "Store assignment", as *VALUE* and *TARGET*, respectively.

iii) For each <target specification> *TS* that is a <host parameter specification>, the corresponding value in the row of *Q* is assigned to *TS* according to the General Rules of Subclause 9.1, "Retrieval assignment", as *VALUE* and *TARGET*, respectively. The assignment of values to targets in the <select target list> is in an implementation-dependent order.

iv) For each <target specification> *TS* that is an <embedded variable specification>, the corresponding value in the row of *Q* is assigned to *TS* according to the General Rules of Subclause 9.1, "Retrieval assignment", as *VALUE* and *TARGET*, respectively. The assignment of values to targets in the <select target list> is in an implementation-dependent order.

5) If an exception condition is raised during the assignment of a value to a target, then the values of all targets are implementation-dependent.

Conformance Rules

None.

14.6 <delete statement: positioned>

Function

Delete a row of a table.

Format

```
<delete statement: positioned> ::=
    DELETE FROM <target table> [ [ AS ] <correlation name> ]
    WHERE CURRENT OF <cursor name>

<target table> ::=
    <table name>
  | ONLY <left paren> <table name> <right paren>
```

Syntax Rules

1) Let *CR* be the cursor denoted by the <cursor name>.

2) Let *TN* be the <table name> contained in <target table>.

3) If <target table> *TT* immediately contains ONLY and the table identified by *TN* is not a typed table, then *TT* is equivalent to *TN*.

4) Let *T* be the simply underlying table of *CR*. *T* is the *subject table* of the <delete statement: positioned>. *T* shall have exactly one leaf underlying table *LUT*.

5) The subject table of a <delete statement: positioned> shall not identify an old transition table or a new transition table.

6) *CR* shall be an updatable cursor.

7) *TN* shall identify *LUT*.

8) <target table> shall specify ONLY if and only if the <table reference> contained in *T* that references *LUT* specifies ONLY.

9) The schema identified by the explicit or implicit qualifier of *TN* shall include the descriptor of *LUT*.

10) Case:

   a) If <correlation name> is specified, then let *CN* be that <correlation name>.

   b) Otherwise, let *CN* be the <table name> contained in <target table>. *CN* is an exposed <table or query name>.

   NOTE 360 — *CN* has no scope.

Access Rules

1) Case:

a) If <delete statement: positioned> is contained, without an intervening <SQL routine spec> that specifies SQL SECURITY INVOKER, in an <SQL schema statement>, then the applicable privileges for the owner of that schema shall include DELETE for *TN*.

b) Otherwise, the current privileges shall include DELETE for *TN*.

NOTE 361 — "current privileges" and "applicable privileges" are defined in Subclause 12.3, "<privileges>".

General Rules

1) If the access mode of the current SQL-transaction or the access mode of the branch of the current SQL-transaction at the current SQL-connection is read-only, and not every leaf generally underlying table of *CR* is a temporary table, then an exception condition is raised: *invalid transaction state — read-only SQL-transaction.*

2) If there is any sensitive cursor *SCR*, other than *CR*, that is currently open in the SQL-transaction in which this SQL-statement is being executed, then

   Case:

   a) If *SCR* has not been held into a subsequent SQL-transaction, then either the change resulting from the successful execution of this statement is made visible to *SCR* or an exception condition is raised: *cursor sensitivity exception — request failed.*

   b) Otherwise, whether the change resulting from the successful execution of this SQL-statement is made visible to *SCR* is implementation-defined.

3) If there is any insensitive cursor *ICR*, other than *CR*, that is currently open, then either the change resulting from the successful execution of this statement is invisible to *ICR*, or an exception condition is raised: *cursor sensitivity exception — request failed.*

4) The extent to which an SQL-implementation may disallow independent changes that are not significant is implementation-defined.

5) If *CR* is not positioned on a row, then an exception condition is raised: *invalid cursor state.*

6) If *CR* is a holdable cursor and a <fetch statement> has not been issued against *CR* within the current SQL-transaction, then an exception condition is raised: *invalid cursor state.*

7) Let *R* be the current row of *CR*. Exactly one row *R1* in *LUT* such that each field in *R* is identical to the corresponding field in *R1* is identified for deletion from *LUT.*

   NOTE 362 — In case more than one row *R1* satisfies the stated condition, it is implementation-dependent which one is identified for deletion.

   NOTE 363 — Identifying a row for deletion is an implementation-dependent mechanism.

8) The effect on *CR* is implementation-defined.

9) Case:

   a) If *LUT* is a base table, then

      Case:

      i) If <target table> specifies ONLY, then *LUT* is *identified for deletion processing without subtables.*

ii) Otherwise, *LUT* is *identified for deletion processing with subtables*.

NOTE 364 — Identifying a base table for deletion processing, with or without subtables, is an implementation-dependent mechanism.

b) If *LUT* is a viewed table, then the General Rules of Subclause 14.18, "Effect of deleting some rows from a viewed table", are applied with <target table> as *VIEW NAME*.

10) The General Rules of Subclause 14.16, "Effect of deleting rows from base tables", are applied.

11) If, while *CR* is open, the row from which the current row of *CR* is derived has been marked for deletion by any <delete statement: searched>, marked for deletion by any <delete statement: positioned> that identifies any cursor other than *CR*, updated by any <update statement: searched>, updated by any <update statement: positioned>, or updated by any <merge statement> that identifies any cursor other than *CR*, then a completion condition is raised: *warning — cursor operation conflict*.

12) If the <delete statement: positioned> deleted the last row of *CR*, then the position of *CR* is after the last row; otherwise, the position of *CR* is before the next row.

Conformance Rules

1) Without Feature S111, "ONLY in query expressions", conforming SQL language shall not contain a <target table> that contains ONLY.

14.7 <delete statement: searched>

Function

Delete rows of a table.

Format

```
<delete statement: searched> ::=
    DELETE FROM <target table> [ [ AS ] <correlation name> ]
    [ WHERE <search condition> ]
```

Syntax Rules

1) Let *TN* be the <table name> contained in the <target table>. Let *T* be the table identified by *TN*.
2) *T* shall be an updatable table.
3) If the <delete statement: searched> is contained in a <triggered SQL statement>, then the <search condition> shall not contain a <value specification> that specifies a parameter reference.
4) *T* is the *subject table* of the <delete statement: searched>.
5) *TN* shall not identify an old transition table or a new transition table.
6) Case:
   a) If <correlation name> is specified, then let *CN* be that <correlation name>.
   b) Otherwise, let *CN* be the <table name> contained in <target table>. *CN* is an exposed <table or query name>.
7) The scope of *CN* is <search condition>.
8) If WHERE <search condition> is not specified, then WHERE TRUE is implicit.
9) The <search condition> shall not generally contain a <routine invocation> whose subject routine is an SQL-invoked routine that possibly modifies SQL-data.

Access Rules

1) Case:
   a) If <delete statement: searched> is contained, without an intervening <SQL routine spec> that specifies SQL SECURITY INVOKER, in an <SQL schema statement>, then let *A* be the <authorization identifier> that owns that schema.
      i) The applicable privileges for *A* shall include DELETE for *TN*.
      ii) If <target table> immediately contains ONLY, then the applicable privileges for *A* shall include SELECT WITH HIERARCHY OPTION on at least one supertable of *T*.

b) Otherwise,

i) The current privileges shall include DELETE for *TN*.

ii) If <target table> immediately contains ONLY, then the current privileges shall include SELECT WITH HIERARCHY OPTION on at least one supertable of *T*.

NOTE 365 — "current privileges" and "applicable privileges" are defined in Subclause 12.3, "<privileges>".

General Rules

1) If the access mode of the current SQL-transaction or the access mode of the branch of the current SQL-transaction at the current SQL-connection is read-only, and *T* is not a temporary table, then an exception condition is raised: *invalid transaction state — read-only SQL-transaction*.

2) If there is any sensitive cursor *CR* that is currently open in the SQL-transaction in which this SQL-statement is being executed, then

Case:

a) If *CR* has not been held into a subsequent SQL-transaction, then either the change resulting from the successful execution of this statement shall be made visible to *CR* or an exception condition is raised: *cursor sensitivity exception — request failed.*

b) Otherwise, whether the change resulting from the successful execution of this SQL-statement is made visible to *CR* is implementation-defined.

3) If there is any cursor *CR* that is currently open and whose <declare cursor> contained INSENSITIVE, then either the change resulting from the successful execution of this statement shall be invisible to *CR*, or an exception condition is raised: *cursor sensitivity exception — request failed.*

4) The extent to which an SQL-implementation may disallow independent changes that are not significant is implementation-defined.

5) The <search condition> is applied to each row of *T* with the exposed <correlation name>s or <table or query name>s of the <table reference> bound to that row.

6) Case:

a) If <target table> contains ONLY, then the rows for which the result of the <search condition> is *True* and for which there is no subrow in a proper subtable of *T* are identified for deletion from *T*.

b) Otherwise, the rows for which the result of the <search condition> is *True* are identified for deletion from *T*.

NOTE 366 — Identifying a row for deletion is an implementation-dependent mechanism.

7) Case:

a) If *T* is a base table, then

Case:

i) If <target table> specifies ONLY, then *T* is *identified for deletion processing without subtables*.

ii) Otherwise, *T* is *identified for deletion processing with subtables*.

NOTE 367 — Identifying a base table for deletion processing, with or without subtables, is an implementation-dependent mechanism.

b) If *T* is a viewed table, then the General Rules of Subclause 14.18, "Effect of deleting some rows from a viewed table", are applied with <target table> as *VIEW NAME*.

8) The General Rules of Subclause 14.16, "Effect of deleting rows from base tables", are applied.

9) Each <subquery> in the <search condition> is effectively executed for each row of *T* and the results are used in the application of the <search condition> to the given row of *T*. If any executed <subquery> contains an outer reference to a column of *T*, then the reference is to the value of that column in the given row of *T*.

NOTE 368 — "outer reference" is defined in Subclause 6.7, "<column reference>".

10) If any row that is marked for deletion by the <delete statement: searched> has been marked for deletion by any <delete statement: positioned> that identifies some cursor *CR* that is still open or updated by any <update statement: positioned> that identifies some cursor *CR* that is still open, then a completion condition is raised: *warning — cursor operation conflict.*

11) All rows that are marked for deletion are effectively deleted at the end of the <delete statement: searched>, prior to the checking of any integrity constraints.

12) If <search condition> is specified, then the <search condition> is evaluated for each row of *T* prior to the invocation of any <triggered action> caused by the imminent or actual deletion of any row of *T*.

13) If no row is deleted, then a completion condition is raised: *no data*.

Conformance Rules

1) Without Feature F781, "Self-referencing operations", conforming SQL language shall not contain a <delete statement: searched> in which a leaf generally underlying table of *T* is an underlying table of any <query expression> generally contained in the <search condition>.

14.8 <insert statement>

Function

Create new rows in a table.

Format

```
<insert statement> ::=
    INSERT INTO <insertion target> <insert columns and source>

<insertion target> ::= <table name>

<insert columns and source> ::=
    <from subquery>
  | <from constructor>
  | <from default>

<from subquery> ::=
    [ <left paren> <insert column list> <right paren> ]
    [ <override clause> ]
    <query expression>

<from constructor> ::=
    [ <left paren> <insert column list> <right paren> ]
    [ <override clause> ]
    <contextually typed table value constructor>

<override clause> ::=
    OVERRIDING USER VALUE
  | OVERRIDING SYSTEM VALUE

<from default> ::= DEFAULT VALUES

<insert column list> ::= <column name list>
```

Syntax Rules

1) Let *TN* be the <table name>; let *T* be the table identified by *TN*. If *T* is a view, then <target table> is effectively replaced by:

   ```
   ONLY ( TN )
   ```

2) *T* shall be insertable-into.
3) For each leaf generally underlying table of *T* whose descriptor includes a user-defined type name *UDTN*, the data type descriptor of the user-defined type *UDT* identified by *UDTN* shall indicate that *UDT* is instantiable.
4) A column identified by the <insert column list> is an object column.
5) *T* shall be an updatable table; each object column of *T* shall be an updatable column.

NOTE 369 — The notion of updatable columns of base tables is defined in Subclause 4.14, "Tables". The notion of updatable columns of viewed tables is defined in Subclause 11.22, "<view definition>".

6) *T* is the *subject table* of the <insert statement>.

7) *TN* shall not identify an old transition table or a new transition table.

8) An <insert columns and source> that specifies DEFAULT VALUES is implicitly replaced by an <insert columns and source> that specifies a <contextually typed table value constructor> of the form

```
VALUES (DEFAULT, DEFAULT, ..., DEFAULT)
```

where the number of "DEFAULT" entries is equal to the number of columns of *T*.

9) Each <column name> in the <insert column list> shall identify an updatable column of *T*. No <column name> of *T* shall be identified more than once. If the <insert column list> is omitted, then an <insert column list> that identifies all columns of *T* in the ascending sequence of their ordinal positions within *T* is implicit.

10) If <contextually typed table value constructor> *CTTVC* is specified, then every <contextually typed row value constructor element> simply contained in *CTTVC* whose positionally corresponding <column name> in <insert column list> references a column of which some underlying column is a generated column shall be a <default specification>.

11) Case:

   a) If some underlying column of a column referenced by a <column name> contained in <insert column list> is a system-generated self-referencing column or a derived self-referencing column, then <override clause> shall be specified.

   b) If for some *n*, some underlying column of the column referenced by the <column name> *CN* contained in the *n*-th ordinal position in <insert column list> is an identity column, then

      Case:

      i) If <from subquery> is specified, then <override clause> shall be specified.

      ii) If any <contextually typed row value expression> simply contained in the <contextually typed table value constructor> is a <row value special case>, then <override clause> shall be specified.

      iii) If the *n*-th <contextually typed row value constructor element> simply contained in any <contextually typed row value constructor> simply contained in the <contextually typed table value constructor> is not a <default specification>, then <override clause> shall be specified.

      NOTE 370 — The preceding subrules do not cover all possibilities of their parent subrule. The remaining possibilities are where <default clause> is specified for every identity column, in which case it is immaterial whether <override clause> is specified or not.

   c) Otherwise, <override clause> shall not be specified.

12) If <contextually typed table value constructor> *CVC* is specified, then the data type of every <contextually typed value specification> *CVS* specified in every <contextually typed row value expression> *CRVS* contained in *CVC* is the data type *DT* indicated in the column descriptor for the positionally corresponding column in the explicit or implicit <insert column list>. If *CVS* is an <empty specification> that specifies ARRAY, then *DT* shall be an array type. If *CVS* is an <empty specification> that specifies MULTISET, then *DT* shall be a multiset type.

13) Let *QT* be the table specified by the <query expression> or <contextually typed table value constructor>. The degree of *QT* shall be equal to the number of <column name>s in the <insert column list>. The column of table *T* identified by the *i*-th <column name> in the <insert column list> corresponds with the *i*-th column of *QT*.

14) The Syntax Rules of Subclause 9.2, "Store assignment", apply to corresponding columns of *T* and *QT* as *TARGET* and *VALUE*, respectively.

15) If the <insert statement> is contained in a <triggered SQL statement>, then the insert value shall not contain a <value specification> that specifies a parameter reference.

16) A <query expression> simply contained in a <from subquery> shall not be a <table value constructor>.

NOTE 371 — This rule removes a syntactic ambiguity; otherwise, "VALUES (1)" could be parsed either as

```
<insert columns and source> ::=
   <from subquery> ::=
   <query expression> ::=
   <table value constructor> ::=
   VALUES (1)
```

or

```
<insert columns and source> ::=
   <from constructor> ::=
   <contextually typed table value constructor> ::=
   VALUES (1)
```

Access Rules

1) Case:

   a) If <insert statement> is contained in, without an intervening <SQL routine spec> that specifies SQL SECURITY INVOKER, an <SQL schema statement>, then let *A* be the <authorization identifier> that owns that schema. The applicable privileges for *A* for *TN* shall include INSERT for each object column.

   b) Otherwise, the current privileges for *TN* shall include INSERT for each object column.

NOTE 372 — "current privileges" and "applicable privileges" are defined in Subclause 12.3, "<privileges>".

General Rules

1) If the access mode of the current SQL-transaction or the access mode of the branch of the current SQL-transaction at the current SQL-connection is read-only, and *T* is not a temporary table, then an exception condition is raised: *invalid transaction state — read-only SQL-transaction*.

2) If there is any sensitive cursor *CR* that is currently open in the SQL-transaction in which this SQL-statement is being executed, then

   Case:

   a) If *CR* has not been held into a subsequent SQL-transaction, then either the change resulting from the successful execution of this statement shall be made visible to *CR* or an exception condition is raised: *cursor sensitivity exception — request failed*.

b) Otherwise, whether the change resulting from the successful execution of this SQL-statement is made visible to *CR* is implementation-defined.

3) If there is any cursor *CR* that is currently open and whose <declare cursor> contained INSENSITIVE, then either the change resulting from the successful execution of this statement shall be invisible to *CR*, or an exception condition is raised: *cursor sensitivity exception — request failed.*

4) The extent to which an SQL-implementation may disallow independent changes that are not significant is implementation-defined.

5) *QT* is effectively evaluated before insertion of any rows into *T*.

6) Let *Q* be the result of evaluating *QT*.

7) For each row *R* of *Q*:

a) A candidate row of *T* is effectively created in which the value of each column is its default value, as specified in the General Rules of Subclause 11.5, "<default clause>". The candidate row consists of every column of *T*.

b) If *T* has a column *RC* of which some underlying column is a self-referencing column, then

Case:

i) If *RC* is a system-generated self-referencing column, then the value of *RC* is effectively replaced by the REF value of the candidate row.

ii) If *RC* is a derived self-referencing column, then the value of *RC* is effectively replaced by a value derived from the columns in the candidate row that correspond to the list of attributes of the derived representation of the reference type of *RC* in an implementation-dependent manner.

c) For each object column in the candidate row, let C_i be the object column identified by the *i*-th <column name> in the <insert column list> and let SV_i be the *i*-th value of *R*.

d) For every C_i for which one of the following conditions is true:

i) C_i is not marked as unassigned and no underlying column of C_i is a self-referencing column.

ii) Some underlying column of C_i is a user-generated self-referencing column.

iii) Some underlying column of C_i is a self-referencing column and OVERRIDING SYSTEM VALUE is specified.

iv) Some underlying column of C_i is an identity column and OVERRIDING SYSTEM VALUE is specified.

the General Rules of Subclause 9.2, "Store assignment", are applied with C_i and SV_i as *TARGET* and *SOURCE*, respectively.

NOTE 373 — The data values allowable in the candidate row may be constrained by a WITH CHECK OPTION constraint. The effect of a WITH CHECK OPTION constraint is defined in the General Rules of Subclause 14.21, "Effect of inserting a table into a viewed table".

8) Let *S* be the table consisting of the candidate rows.

Case:

**

a) If *T* is a base table, then *T* is *identified for insertion of source table S.*

NOTE 374 — Identifying a base table for insertion of a source table is an implementation-dependent operation.

b) If *T* is a viewed table, then the General Rules of Subclause 14.21, "Effect of inserting a table into a viewed table", are applied with *S* as *SOURCE* and *T* as *TARGET.*

9) The General Rules of Subclause 14.19, "Effect of inserting tables into base tables", are applied.

10) If *Q* is empty, then a completion condition is raised: *no data.*

Conformance Rules

1) Without Feature F781, "Self-referencing operations", conforming SQL language shall not contain an <insert statement> in which the <table name> of a leaf generally underlying table of *T* is generally contained in the <from subquery> except as the table name of a qualifying table of a column reference.

2) Without Feature F222, "INSERT statement: DEFAULT VALUES clause", conforming SQL language shall not contain a <from default>.

3) Without Feature S024, "Enhanced structured types", in conforming SQL language, for each column *C* identified in the explicit or implicit <insert column list>, if the declared type of *C* is a structured type *TY*, then the declared type of the corresponding column of the <query expression> or <contextually typed table value constructor> shall be *TY.*

4) Without Feature S043, "Enhanced reference types", conforming SQL language shall not contain an <override clause>.

14.9 <merge statement>

Function

Conditionally update rows of a table, or insert new rows into a table, or both.

Format

```
<merge statement> ::=
    MERGE INTO <target table> [ [ AS ] <merge correlation name> ]
    USING <table reference>
    ON <search condition> <merge operation specification>

<merge correlation name> ::= <correlation name>

<merge operation specification> ::= <merge when clause>...

<merge when clause> ::=
    <merge when matched clause>
  | <merge when not matched clause>

<merge when matched clause> ::=
    WHEN MATCHED THEN <merge update specification>

<merge when not matched clause> ::=
    WHEN NOT MATCHED THEN <merge insert specification>

<merge update specification> ::= UPDATE SET <set clause list>

<merge insert specification> ::=
    INSERT [ <left paren> <insert column list> <right paren> ]
    [ <override clause> ]
    VALUES <merge insert value list>

<merge insert value list> ::=
    <left paren>
    <merge insert value element> [ { <comma> <merge insert value element> }... ]
    <right paren>

<merge insert value element> ::=
    <value expression>
  | <contextually typed value specification>
```

Syntax Rules

1) Neither <merge when matched clause> nor <merge when not matched clause> shall be specified more than once.

2) Let *TN* be the <table name> contained in <target table> and let *T* be the table identified by *TN*. *T* is the *subject table* of the <merge statement>.

3) *T* shall be both updatable and insertable-into.

4) *T* shall not be an old transition table or a new transition table.

5) For each leaf generally underlying table of *T* whose descriptor includes a user-defined type name *UDTN*, the data type descriptor of the user-defined type *UDT* identified by *UDTN* shall indicate that *UDT* is instantiable.

6) If *T* is a view, then <target table> is effectively replaced by:

```
ONLY ( TN )
```

7) Case:

   a) If <merge correlation name> is specified, then let *CN* be the <correlation name> contained in <merge correlation name>.

   b) Otherwise, let *CN* be the <table name> contained in <target table>.

8) The scope of *CN* is <search condition> and <set clause list>.

9) Let *TR* be the <table reference> immediately contained in <merge statement>. *TR* shall not directly contain a <joined table>.

10) The <correlation name> or exposed <table name> that is exposed by *TR* shall not be equivalent to *CN*.

11) If the <insert column list> is omitted, then an <insert column list> that identifies all columns of *T* in the ascending sequence of their ordinal position within *T* is implicit.

12) Case:

   a) If *T* is a referenceable table or a table having an identity column whose descriptor includes an indication that values are always generated, then:

      i) Let *C* be the self-referencing column or identity column of *T*.

      ii) If *C* is an identity column, a system-generated self-referencing column or a derived self-referencing column and *C* is contained in <insert column list>, then <override clause> shall be specified; otherwise, <override clause> shall not be specified.

   b) Otherwise, <override clause> shall not be specified.

13) The <search condition> shall not generally contain a <routine invocation> whose subject routine is an SQL-invoked routine that possibly modifies SQL-data.

14) Each column identified by an <object column> in the <set clause list> is an *update object column*. Each column identified by a <column name> in the implicit or explicit <insert column list> is an *insert object column*. Each update object column and each insert object column is an *object column*.

15) Every object column shall identify an updatable column of *T*.

NOTE 375 — The notion of updatable columns of base tables is defined in Subclause 4.14, "Tables". The notion of updatable columns of viewed tables is defined in Subclause 11.22, "<view definition>".

16) No <column name> of *T* shall be identified more than once in in an <insert column list>.

17) Let *NI* be the number of <merge insert value element>s contained in <merge insert value list>. Let EXP_1, EXP_2, ... , EXP_{NI} be those <merge insert value element>s.

18) The number of <column name>s in the <insert column list> shall be equal to *NI*.

19) The declared type of every <contextually typed value specification> *CVS* in a <merge insert value list> is the data type *DT* indicated in the column descriptor for the positionally corresponding column in the explicit or implicit <insert column list>. If *CVS* is an <empty specification> that specifies ARRAY, then *DT* shall be an array type. If *CVS* is an <empty specification> that specifies MULTISET, then *DT* shall be a multiset type.

20) Every <merge insert value element> whose positionally corresponding <column name> in <insert column list> references a column of which some underlying column is a generated column shall be a <default specification>.

21) For 1 (one) ≤ *i* *NI*, the Syntax Rules of Subclause 9.2, "Store assignment", apply to the column of table *T* identified by the *i*-th <column name> in the <insert column list> and EXP_i as *TARGET* and *VALUE*, respectively.

Access Rules

1) Case:

   a) If <merge statement> is contained, without an intervening <SQL routine spec> that specifies SQL SECURITY INVOKER, in an <SQL schema statement>, then let *A* be the <authorization identifier> that owns that schema.

      i) The applicable privileges for *A* shall include UPDATE for each update object column.

      ii) The applicable privileges for *A* shall include INSERT for each insert object column.

      iii) If <target table> immediately contains ONLY, then the applicable privileges for *A* shall include SELECT WITH HIERARCHY OPTION on at least one supertable of *T*.

   b) Otherwise,

      i) The current privileges shall include UPDATE for each update object column.

      ii) The current privileges shall include INSERT for each insert object column.

      iii) If <target table> immediately contains ONLY, then the current privileges shall include SELECT WITH HIERARCHY OPTION on at least one supertable of *T*.

      NOTE 376 — "current privileges" and "applicable privileges" are defined in Subclause 12.3, "<privileges>".

General Rules

1) If the access mode of the current SQL-transaction or the access mode of the branch of the current SQL-transaction at the current SQL-connection is read-only, and *T* is not a temporary table, then an exception condition is raised: *invalid transaction state — read-only SQL-transaction*.

2) If there is any sensitive cursor *CR* that is currently open in the SQL-transaction in which this SQL-statement is being executed, then

   Case:

a) If *CR* has not been held into a subsequent SQL-transaction, then either the change resulting from the successful execution of this statement shall be made visible to *CR* or an exception condition is raised: *cursor sensitivity exception — request failed.*

b) Otherwise, whether the change resulting from the successful execution of this SQL-statement is made visible to *CR* is implementation-defined.

3) If there is any cursor *CR* that is currently open and whose <declare cursor> contained INSENSITIVE, then either the change resulting from the successful execution of this statement shall be invisible to *CR*, or an exception condition is raised: *cursor sensitivity exception — request failed.*

4) The extent to which an SQL-implementation may disallow independent changes that are not significant is implementation-defined.

5) Let *QT* be the table specified by the <table reference>. *QT* is effectively evaluated before update or insertion of any rows in *T*. Let *Q* be the result of evaluating *QT*.

6) For each <merge when clause>,

Case:

a) If <merge when matched clause> is specified, then:

i) For each row *R1* of *T*:

1) The <search condition> is applied to *R1* with the exposed <table name> of the <target table> bound to *R1* and to each row of *Q* with the exposed <correlation name>s or <table or query name>s of the <table reference> bound to that row. The <search condition> is effectively evaluated for *R1* before updating any row of *T* and prior to the invocation of any <triggered action> caused by the update of any row of *T* and before inserting any rows into *T* and prior to the invocation of any <triggered action> caused by the insert of any row of *T*. Each <subquery> in the <search condition> is effectively executed for *R1* and for each row of *Q* and the results used in the application of the <search condition> to *R1* and the given row of *Q*. If any executed <subquery> contains an outer reference to a column of *T*, then the reference is to the value of that column in the given row of *T*.

Case:

A) If <target table> contains ONLY, then *R1* is a subject row if *R1* has no subrow in a proper subtable of *T* and the result of the <search condition> is *True* for some row *R2* of *Q*. *R2* is the matching row.

B) Otherwise, *R1* is a subject row if the result of the <search condition> is *True* for some row *R2* of *Q*. *R2* is the matching row.

NOTE 377 — "outer reference" is defined in Subclause 6.7, "<column reference>".

2) If *R1* is a subject row, then:

A) Let *M* be the number of matching rows in *Q* for *R1*.

B) If *M* is greater than 1 (one), then an exception condition is raised: *cardinality violation.*

C) The <update source> of each <set clause> is effectively evaluated for *R1* before any row of *T* is updated and prior to the invocation of any <triggered action> caused by the update of any row of *T*. The resulting value is the update value.

D) A candidate new row is constructed by copying the subject row and updating it as specified by each <set clause> by applying the General Rules of Subclause 14.12, "<set clause list>".

ii) If *T* is a base table, then each subject row is also an object row; otherwise, an object row is any row of a leaf generally underlying table of *T* from which a subject row is derived.

NOTE 378 — The data values allowable in the object rows may be constrained by a WITH CHECK OPTION constraint. The effect of a WITH CHECK OPTION constraint is defined in the General Rules of Subclause 14.24, "Effect of replacing some rows in a viewed table".

iii) If any row in the set of object rows has been marked for deletion by any <delete statement: positioned> that identifies some cursor *CR* that is still open or updated by any <update statement: positioned> that identifies some cursor *CR* that is still open, then a completion condition is raised: *warning — cursor operation conflict.*

iv) Let *CL* be the columns of *T* identified by the <object column>s contained in the <set clause list>.

v) Each subject row *SR* is identified for replacement, by its corresponding candidate new row *CNR*, in *T*. The set of (*SR*, *CNR*) pairs is the replacement set for *T*.

NOTE 379 — Identifying a row for replacement, associating a replacement row with an identified row, and associating a replacement set with a table are implementation-dependent operations.

vi) Case:

1) If *T* is a base table, then

Case:

A) If <target table> specifies ONLY, then *T* is identified for replacement processing without subtables with respect to object columns *CL*.

B) Otherwise, *T* is identified for replacement processing with subtables with respect to object columns *CL*.

NOTE 380 — Identifying a base table for replacement processing, with or without subtables, is an implementation-dependent mechanism. In general, though not here, the list of object columns can be empty.

2) If *T* is a viewed table, then the General Rules of Subclause 14.24, "Effect of replacing some rows in a viewed table", are applied with <target table> as *VIEW NAME*.

vii) The General Rules of Subclause 14.22, "Effect of replacing rows in base tables", are applied.

b) If <merge when not matched clause> is specified, then:

i) Let *TR1* be the <target table> immediately contained in <merge statement>, let *TR2* be the <table reference> immediately contained in <merge statement>, and let *SC1* be the <search condition> immediately contained in <merge statement>. If <merge correlation name> is specified, let *MCN* be "AS <merge correlation name>"; otherwise, let *MCN* be a zero-length string. Let *S1* be the result of

```
SELECT *
FROM TR1 MCN, TR2
WHERE SC1
```

ii) Let *S2* be the collection of rows of *Q* for which there exists in *S1* some row that is the concatenation of some row *R1* of *T* and some row *R2* of *Q*.

iii) Let *S3* be the collection of rows of *Q* that are not in *S2*. Let *SN3* be the effective distinct name for *S3*. Let *EN* be the exposed <correlation name> or <table or query name> of *TR2*.

iv) Let *S4* be the result of:

```
SELECT EXP_1, EXP_2, ... , EXP_NI
FROM SN3 AS EN
```

v) *S4* is effectively evaluated before insertion of any rows into or update of any rows in *T*.

vi) For each row *R* of *S4*:

1) A candidate row of *T* is effectively created in which the value of each column is its default value, as specified in the General Rules of Subclause 11.5, "<default clause>". The candidate row consists of every column of *T*.

2) If *T* has a column *RC* of which some underlying column is a self-referencing column, then

Case:

A) If *RC* is a system-generated self-referencing column, then the value of *RC* is effectively replaced by the REF value of the candidate row.

B) If *RC* is a derived self-referencing column, then the value of *RC* is effectively replaced by a value derived from the columns in the candidate row that correspond to the list of attributes of the derived representation of the reference type of *RC* in an implementation-dependent manner.

3) For each object column in the candidate row, let C_i be the object column identified by the *i*-th <column name> in the <insert column list> and let SV_i be the *i*-th value of *R*.

4) For every C_i for which one of the following conditions is true:

A) C_i is not marked as unassigned and no underlying column of C_i is a self-referencing column.

B) Some underlying column of C_i is a user-generated self-referencing column.

C) Some underlying column of C_i is a self-referencing column and OVERRIDING SYSTEM VALUE is specified.

D) Some underlying column of C_i is an identity column and OVERRIDING SYSTEM VALUE is specified.

the General Rules of Subclause 9.2, "Store assignment", are applied to C_i and SV_i as *TARGET* and *SOURCE*, respectively.

NOTE 381 — The data values allowable in the candidate row may be constrained by a WITH CHECK OPTION constraint. The effect of a WITH CHECK OPTION constraint is defined in the General Rules of Subclause 14.21, "Effect of inserting a table into a viewed table".

vii) Let *S* be the table consisting of the candidate rows.

Case:

1) If *T* is a base table, then *T* is *identified for insertion of source table S*.

   NOTE 382 — Identifying a base table for insertion of a source table is an implementation-dependent operation.

2) If *T* is a viewed table, then the General Rules of Subclause 14.21, "Effect of inserting a table into a viewed table", are applied with *S* as *SOURCE* and *T* as *TARGET*.

viii) The General Rules of Subclause 14.19, "Effect of inserting tables into base tables", are applied.

7) If *Q* is empty, then a completion condition is raised: *no data*.

Conformance Rules

1) Without Feature F781, "Self-referencing operations", conforming SQL language shall not contain a <merge statement> in which a leaf generally underlying table of *T* is generally contained in a <query expression> immediately contained in the <table reference> except as the <table or query name> or <correlation name> of a column reference.

2) Without Feature F781, "Self-referencing operations", conforming SQL language shall not contain a <merge statement> in which a leaf generally underlying table of *T* is an underlying table of any <query expression> generally contained in the <search condition>.

3) Without Feature S024, "Enhanced structured types", conforming SQL language shall not contain a <merge statement> that does not satisfy the condition: for each column *C* identified in the explicit or implicit <insert column list>, if the declared type of *C* is a structured type *TY*, then the declared type of the corresponding column of the <query expression> or <contextually typed table value constructor> is *TY*.

4) Without Feature F312, "MERGE statement", conforming SQL language shall not contain a <merge statement>.

14.10 <update statement: positioned>

Function

Update a row of a table.

Format

```
<update statement: positioned> ::=
    UPDATE <target table> [ [ AS ] <correlation name> ]
    SET <set clause list>
    WHERE CURRENT OF <cursor name>
```

Syntax Rules

1) Let *CR* be the cursor denoted by the <cursor name>.

2) Let *TU* be the simply underlying table of *CR*. *TU* is the *subject table* of the <update statement: positioned>. *TU* shall have exactly one leaf underlying table *LUT*.

   NOTE 383 — The "simply underlying table" of a <cursor specification> is defined in Subclause 14.1, "<declare cursor>".

3) Let *TN* be the <table name> contained in <target table>. *TN* shall identify *LUT*.

4) <target table> shall specify ONLY if and only if the <table reference> contained in *TU* that references *LUT* specifies ONLY.

5) *TN* shall not identify an old transition table or a new transition table.

6) *CR* shall be an updatable cursor.

7) Let *T* be the table identified by *TN*.

8) Case:

   a) If <correlation name> is specified, then let *CN* be that <correlation name>.

   b) Otherwise, let *CN* be the <table name> contained in <target table>. *CN* is an exposed <table or query name>.

9) The scope of *CN* is <set clause list>.

10) If *CR* is an ordered cursor, then for each <object column> *OC* contained in <set clause list>, the <order by clause> of the defining <cursor specification> for *CR* shall not generally contain a <column reference> that references *OC* or an underlying column of the column identified by *OC*.

11) If the cursor identified by <cursor name> was specified using an explicit or implicit <updatability clause> of FOR UPDATE, then each <column name> specified as an <object column> shall identify a column in the explicit or implicit <column name list> associated with the <updatability clause>.

Access Rules

1) Case:

   a) If <update statement: positioned> is contained, without an intervening <SQL routine spec> that specifies SQL SECURITY INVOKER, in an <SQL schema statement>, then let *A* be the <authorization identifier> that owns that schema. The applicable privileges for *A* shall include UPDATE for each <object column>.

   b) Otherwise, the current privileges shall include UPDATE for each <object column>.

   NOTE 384 — "current privileges" and "applicable privileges" are defined in Subclause 12.3, "<privileges>".

General Rules

1) If the access mode of the current SQL-transaction or the access mode of the branch of the current SQL-transaction at the current SQL-connection is read-only and not every leaf generally underlying table of *CR* is a temporary table, then an exception condition is raised: *invalid transaction state — read-only SQL-transaction.*

2) If there is any sensitive cursor *SCR*, other than *CR*, that is currently open in the SQL-transaction in which this SQL-statement is being executed, then

   Case:

   a) If *SCR* has not been held into a subsequent SQL-transaction, then either the change resulting from the successful execution of this statement is made visible to *CR* or an exception condition is raised: *cursor sensitivity exception — request failed.*

   b) Otherwise, whether the change resulting from the successful execution of this SQL-statement is made visible to *SCR* is implementation-defined.

3) If there is any insensitive cursor *ICR*, other than *CR*, that is currently open, then either the change resulting from the successful execution of this statement is invisible to *CR*, or an exception condition is raised: *cursor sensitivity exception — request failed.*

4) The extent to which an SQL-implementation may disallow independent changes that are not significant is implementation-defined.

5) If *CR* is not positioned on a row, then an exception condition is raised: *invalid cursor state.*

6) If *CR* is a holdable cursor and a <fetch statement> has not been issued against *CR* within the current SQL-transaction, then an exception condition is raised: *invalid cursor state.*

7) An object row is any row of a base table from which the current row of *CR* is derived.

8) If, while *CR* is open, an object row has been marked for deletion by any <delete statement: searched>, marked for deletion by any <delete statement: positioned> that identifies any cursor other than *CR*, updated by any <update statement: searched>, updated by any <update statement: positioned>, or updated by any <merge statement> that identifies any cursor other than *CR*, then a completion condition is raised: *warning — cursor operation conflict.*

9) The value associated with DEFAULT is the default value for the <object column> in the containing <set clause>, as indicated in the General Rules of Subclause 11.5, "<default clause>".

10) Each <update source> is effectively evaluated for the current row before any of the current row's object rows is updated.

11) *CR* remains positioned on its current row, even if an exception condition is raised during evaluation of any <update source>.

12) A candidate new row is constructed by copying the current row of *CR* and updating it as specified by each <set clause> by applying the General Rules of Subclause 14.12, "<set clause list>".

    NOTE 385 — The data values allowable in an object row may be constrainted by a WITH CHECK OPTION constraint. The effect of a WITH CHECK OPTION constraint is defined in the General Rules of Subclause 14.24, "Effect of replacing some rows in a viewed table".

13) Let *CL* be the columns of *T* identified by the <object column>s contained in the <set clause list>.

14) Let *R1* be the candidate new row and let *R* be the current row of *CR*. Exactly one row *TR* in *T* such that each field in *R* is identical to the corresponding field in *TR* is identified for replacement in *T*. The current row *R* of *CR* is replaced by *R1*. Let *TR1* be a row consisting of the fields of *R1* and the fields of *TR* that have no corresponding fields in *R1*, ordered according to the order of their corresponding columns in *T*. *TR1* is the replacement row for *TR* and `{ ( TR, TR1 ) }` is the replacement set for *T*.

    NOTE 386 — In case more than one row *R1* satisfies the stated condition, it is implementation-dependent which one is identified for replacement.

    NOTE 387 — Identifying a row for replacement, associating a replacement row with an identified row, and associating a replacement set with a table are implementation-dependent mechanisms.

15) Case:

    a) If *LUT* is a base table, then

       Case:

       i) If <target table> specifies ONLY, then *LUT* is *identified for replacement processing without subtables* with respect to object columns *CL*.

       ii) Otherwise, *LUT* is *identified for replacement processing with subtables* with respect to object columns *CL*.

       NOTE 388 — Identifying a base table for replacement processing, with or without subtables, is an implementation-dependent mechanism. In general, though not here, the list of object columns can be empty.

    b) If *LUT* is a viewed table, then the General Rules of Subclause 14.24, "Effect of replacing some rows in a viewed table", are applied with <target table> as *VIEW NAME*.

16) The General Rules of Subclause 14.22, "Effect of replacing rows in base tables", are applied.

Conformance Rules

1) Without Feature F831, "Full cursor update", conforming SQL language shall not contain an <update statement: positioned> in which *CR* identifies an ordered cursor.

14.11 <update statement: searched>

Function

Update rows of a table.

Format

```
<update statement: searched> ::=
    UPDATE <target table> [ [ AS ] <correlation name> ]
    SET <set clause list>
    [ WHERE <search condition> ]
```

Syntax Rules

1) Let *TN* be the <table name> contained in <target table>; let *T* be the table identified by *TN*. *T* shall be an updatable table.

2) *T* is the *subject table* of the <update statement: searched>.

3) *TN* shall not identify an old transition table or a new transition table.

4) Case:

   a) If <correlation name> is specified, then let *CN* be that <correlation name>.

   b) Otherwise, let *CN* be the <table name> contained in <target table>. *CN* is an exposed <table or query name>.

5) The scope of *CN* is <set clause list> and <search condition>.

6) If the <update statement: searched> is contained in a <triggered SQL statement>, then the <search condition> shall not contain a <value specification> that specifies a parameter reference.

7) The <search condition> shall not generally contain a <routine invocation> whose subject routine is an SQL-invoked routine that possibly modifies SQL-data.

Access Rules

1) Case:

   a) If <update statement: searched> is contained, without an intervening <SQL routine spec> that specifies SQL SECURITY INVOKER, in an <SQL schema statement>, then let *A* be the <authorization identifier> that owns that schema.

      i) The applicable privileges for *A* for *TN* shall include UPDATE for each <object column>.

      ii) If <target table> immediately contains ONLY, then the applicable privileges for *A* shall include SELECT WITH HIERARCHY OPTION on at least one supertable of *T*.

   b) Otherwise,

i) The current privileges for *TN* shall include UPDATE for each <object column>.

ii) If <target table> immediately contains ONLY, then the current privileges shall include SELECT WITH HIERARCHY OPTION on at least one supertable of *T*.

NOTE 389 — "current privileges" and "applicable privileges" are defined in Subclause 12.3, "<privileges>".

General Rules

1) If the access mode of the current SQL-transaction or the access mode of the branch of the current SQL-transaction at the current SQL-connection is read-only and *T* is not a temporary table, then an exception condition is raised: *invalid transaction state — read-only SQL-transaction.*

2) If there is any sensitive cursor *CR* that is currently open in the SQL-transaction in which this SQL-statement is being executed, then

   Case:

   a) If *CR* has not been held into a subsequent SQL-transaction, then either the change resulting from the successful execution of this statement shall be made visible to *CR* or an exception condition is raised: *cursor sensitivity exception — request failed.*

   b) Otherwise, whether the change resulting from the successful execution of this SQL-statement is made visible to *CR* is implementation-defined.

3) If there is any cursor *CR* that is currently open and whose <declare cursor> contained INSENSITIVE, then either the change resulting from the successful execution of this statement shall be invisible to *CR*, or an exception condition is raised: *cursor sensitivity exception — request failed.*

4) The extent to which an SQL-implementation may disallow independent changes that are not significant is implementation-defined.

5) Case:

   a) If <target table> contains ONLY, then

      Case:

      i) If a <search condition> is not specified, then all rows of *T* for which there is no subrow in a proper subtable of *T* are the subject rows.

      ii) If a <search condition> is specified, then it is applied to each row of *T* with the exposed <correlation name>s or <table or query name>s of the <table reference> bound to that row, and the subject rows are those rows for which the result of the <search condition> is *True* and for which there is no subrow in a proper subtable of *T*. The <search condition> is effectively evaluated for each row of *T* before updating any row of *T*.

      Each <subquery> in the <search condition> is effectively executed for each row of *T* and the results used in the application of the <search condition> to the given row of *T*. If any executed <subquery> contains an outer reference to a column of *T*, then the reference is to the value of that column in the given row of *T*.

   b) Otherwise,

      Case:

i) If a <search condition> is not specified, then all rows of *T* are the subject rows.

ii) If a <search condition> is specified, then it is applied to each row of *T* with the exposed <table name> of the <target table> bound to that row, and the subject rows are those rows for which the result of the <search condition> is *True*. The <search condition> is effectively evaluated for each row of *T* before any row of *T* is updated.

Each <subquery> in the <search condition> is effectively executed for each row of *T* and the results used in the application of the <search condition> to the given row of *T*. If any executed <subquery> contains an outer reference to a column of *T*, then the reference is to the value of that column in the given row of *T*.

NOTE 390 — *outer reference* is defined in Subclause 6.7, "<column reference>".

6) If *T* is a base table, then each subject row is also an *object row*; otherwise, an *object row* is any row of a leaf generally underlying table of *T* from which a subject row is derived.

7) If any row in the set of object rows has been marked for deletion by any <delete statement: positioned> that identifies some cursor *CR* that is still open or updated by any <update statement: positioned> that identifies some cursor *CR* that is still open, then a completion condition is raised: *warning — cursor operation conflict*.

8) If a <search condition> is specified, then the <search condition> is evaluated for each row of *T* prior to the invocation of any <triggered action> caused by the update of any row of *T*.

9) The <update source> of each <set clause> is effectively evaluated for each row of *T* before any row of *T* is updated.

10) For each subject row, a candidate new row is constructed by copying the subject row and updating it as specified by each <set clause> by applying the General Rules of Subclause 14.12, "<set clause list>".

NOTE 391 — The data values allowable in the object rows may be constrained by a WITH CHECK OPTION constraint. The effect of a WITH CHECK OPTION constraint is defined in the General Rules of Subclause 14.24, "Effect of replacing some rows in a viewed table".

11) Let *CL* be the columns of *T* identified by the <object column>s contained in the <set clause list>.

12) Each subject row *SR* is identified for replacement, by its corresponding candidate new row *CNR*, in *T*. The set of `(SR, CNR)` pairs is the *replacement set* for *T*.

NOTE 392 — Identifying a row for replacement, associating a replacement row with an identified row, and associating a replacement set with a table are implementation-dependent operations.

13) Case:

a) If *T* is a base table, then

Case:

i) If <target table> specifies ONLY, then *T* is *identified for replacement processing without subtables* with respect to object columns *CL*.

ii) Otherwise, *T* is *identified for replacement processing with subtables* with respect to object columns *CL*.

NOTE 393 — Identifying a base table for replacement processing, with or without subtables, is an implementation-dependent mechanism. In general, though not here, the list of object columns can be empty.

b) If *T* is a viewed table, then the General Rules of Subclause 14.24, "Effect of replacing some rows in a viewed table", are applied with <target table> as *VIEW NAME*.

14) The General Rules of Subclause 14.22, "Effect of replacing rows in base tables", are applied.

15) If the set of object rows is empty, then a completion condition is raised: *no data*.

Conformance Rules

1) Without Feature F781, "Self-referencing operations", conforming SQL language shall not contain an <update statement: positioned> in which a leaf generally underlying table of *T* is an underlying table of any <query expression> generally contained in the <search condition>.

14.12 <set clause list>

Function

Specify a list of updates.

Format

```
<set clause list> ::= <set clause> [ { <comma> <set clause> }... ]

<set clause> ::=
    <multiple column assignment>
  | <set target> <equals operator> <update source>

<set target> ::=
    <update target>
  | <mutated set clause>

<multiple column assignment> ::=
    <set target list> <equals operator> <assigned row>

<set target list> ::=
    <left paren> <set target> [ { <comma> <set target> }... ] <right paren>

<assigned row> ::= <contextually typed row value expression>

<update target> ::=
    <object column>
  | <object column>
    <left bracket or trigraph> <simple value specification> <right bracket or trigraph>

<object column> ::= <column name>

<mutated set clause> ::= <mutated target> <period> <method name>

<mutated target> ::=
    <object column>
  | <mutated set clause>

<update source> ::=
    <value expression>
  | <contextually typed value specification>
```

Syntax Rules

1) Let *T* be the table identified by the <target table> contained in the containing <update statement: positioned>, <update statement: searched>, or <merge statement>.

2) Each <column name> specified as an <object column> shall identify an updatable column of *T*.

   NOTE 394 — The notion of updatable columns of base tables is defined in Subclause 4.14, "Tables". The notion of updatable columns of viewed tables is defined in Subclause 11.22, "<view definition>".

3) Each <set clause> *SC* that immediately contains a <multiple column assignment> is effectively replaced by a <set clause list> *MSCL* as follows:

   a) Let *STN* be the number of <set target>s contained in <set target list>.

   b) *STN* shall be equal to the degree of the <assigned row> *AR* contained in *SC*.

   c) Let ST_i, 1 (one) ≤ *i* ≤ *STN*, be the *i*-th <set target> contained in the <set target list> of *SC* and let DT_i be the declared type of the *i*-th field of *AR*. The *i*-th <set clause> in *MSCL* is:

   ST_i = CAST (*AR* AS ROW (F1 DT_1, F2 DT_2, ..., F*STN* DT_{STN})).F*i*

   NOTE 395 — "F*n*" here stands for the <field name> consisting of the letter "F" followed, with no intervening <separator> by the decimal <digit> or <digit>s comprising a <literal> corresponding to the value *n*.

4) If <set clause> *SC* specifies an <object column> that references a column of which some underlying column is either a generated column or an identity column whose descriptor indicates that values are always generated, then the <update source> specified in *SC* shall consist of a <default specification>.

5) A <value expression> simply contained in an <update source> in a <set clause> shall not directly contain a <set function specification>.

6) If the <set clause list> *OSCL* contains one or more <set clause>s that contain a <mutated set clause>, then:

   a) Let *N* be the number of <set clause>s in *OSCL* that contain a <mutated set clause>.

   b) For 1 (one) ≤ *i* ≤ *N*:

      i) Let SC_i be the *i*-th <set clause> that contains a <mutated set clause>.

      ii) Let $RCVE_i$ be the <update source> immediately contained in SC_i.

      iii) Let MSC_i be the <mutated set clause> immediately contained in the <set target> immediately contained in SC_i.

      iv) Let OC_i be the <object column> contained in MSC_i. The declared type of the column identified by OC_i shall be a structured type.

      v) Let M_i be the number of <method name>s contained in MSC_i.

      vi) For 1 (one) ≤ *j* ≤ M_i:

         1) If *j* = 1 (one), then

            Case:

            A) Let $MT_{i,1}$ be the <mutated target> immediately contained in MSC_i.

            B) Let $MN_{i,1}$ be the <method name> immediately contained in MSC_i.

            C) Let $V_{i,1}$ be:

            $MT_{i,1}$. $MN_{i,1}$ ($RCVE_i$)

         2) Otherwise:

A) Let $MT_{i,j}$ be the <mutated target> immediately contained in the <mutated set clause> immediately contained in $MT_{i,j-1}$.

B) Let $MN_{i,j}$ be the <method name> immediately contained in the <mutated set clause> immediately contained in $MT_{i,j-1}$.

C) Let $V_{i,j}$ be

```
MTi,j . MNi,j ( Vi,j-1 )
```

c) *OSCL* is equivalent to a <set clause list> *NSCL* derived as follows:

i) Let *NSCL* be a <set clause list> derived from *OSCL* by replacing every <set clause> SC_a, 1 (one) $\leq a \leq N$, that contains a <mutated set clause> with:

```
MTa,Ma = Va,Ma
```

ii) For 1 (one) $\leq b \leq N$, if there exists a c such that $c < b$ and OC_c is equivalent to OC_b, then:

1) Every occurrence of OC_b in $V_{b,Mb}$ is replaced by $V_{c,Mc}$.

2) SC_c is deleted from *NSCL*.

7) Equivalent <object column>s shall not appear more than once in a <set clause list>.

NOTE 396 — Multiple occurrences of equivalent <object column>s within <mutated set clause>s are eliminated by the preceding Syntax Rule of this Subclause.

8) If the <update source> of <set clause> *SC* specifies a <contextually typed value specification> *CVS*, then the data type of *CVS* is the data type *DT* of the <update target> or <mutated set clause> specified in *SC*.

9) If *CVS* is an <empty specification>, then *DT* shall be a collection type. If *CVS* specifies ARRAY, then *DT* shall be an array type. If *CVS* specifies MULTISET, then *DT* shall be a multiset type.

10) For every <object column> in a <set clause>,

Case:

a) If the <update target> immediately contains <simple value specification>, then the declared type of the column of *T* identified by the <object column> shall be an array type. The Syntax Rules of Subclause 9.2, "Store assignment", apply to an arbitrary site whose declared type is the element type of the column of *T* identified by the <object column> and the <update source> of the <set clause> as *TARGET* and *VALUE*, respectively.

b) Otherwise, the Syntax Rules of Subclause 9.2, "Store assignment", apply to the column of *T* identified by the <object column> and the <update source> of the <set clause> as *TARGET* and *VALUE*, respectively.

Access Rules

None.

General Rules

1) A <set clause> specifies one or more object columns and an update value. An *object column* is a column identified by an <object column> in the <set clause>. The *update value* is the value specified by the <update source> contained in the <set clause>.

2) The value of the *i*-th object column denoted by *C*, is replaced as follows:

   Case:

   a) If the *i*-th <set clause> contains an <update target> that immediately contains a <simple value specification>, then

      Case:

      i) If the value of *C* is null, then an exception condition is raised: *data exception — null value in array target.*

      ii) Otherwise:

         1) Let *N* be the maximum cardinality of *C*.

         2) Let *M* be the cardinality of the value of *C*.

         3) Let *I* be the value of the <simple value specification> immediately contained in <update target>.

         4) Let *EDT* be the element type of *C*.

         5) Case:

            A) If *I* is greater than zero and less than or equal to *M*, then the value of *C* is replaced by an array *A* with element type *EDT* and cardinality *M* derived as follows:

               I) For *j* varying from 1 (one) to *I*–1 and from *I*+1 to *M*, the *j*-th element in *A* is the value of the *j*-th element in *C*.

               II) The *I*-th element of *A* is set to the *i*-th update value, denoted by *SV*, by applying the General Rules of Subclause 9.2, "Store assignment", to the *I*-th element of *A* and *SV* as *TARGET* and *VALUE*, respectively.

            B) If *I* is greater than *M* and less than or equal to *N*, then the value of *C* is replaced by an array *A* with element type *EDT* and cardinality *I* derived as follows:

               I) For *j* varying from 1 (one) to *M*, the *j*-th element in *A* is the value of the *j*-th element in *C*.

               II) For *j* varying from *M*+1 to *I*–1, the *j*-th element in *A* is the null value.

               III) The *I*-th element of *A* is set to the *i*-th update value, denoted by *SV*, by applying the General Rules of Subclause 9.2, "Store assignment", to the *I*-th element of *A* and *SV* as *TARGET* and *VALUE*, respectively.

            C) Otherwise, an exception condition is raised: *data exception — array element error.*

b) Otherwise, the value of *C* is replaced by the *i*-th update value, denoted by *SV*. The General Rules of Subclause 9.2, "Store assignment", are applied to *C* and *SV* as *TARGET* and *VALUE*, respectively.

Conformance Rules

1) Without Feature F781, "Self-referencing operations", conforming SQL language shall not contain a <set clause> in which a leaf generally underlying table of *T* is an underlying table of any <query expression> generally contained in any <value expression> simply contained in an <update source> or <assigned row> immediately contained in the <set clause>.
2) Without Feature S091, "Basic array support", conforming SQL language shall not contain an <update target> that immediately contains a <simple value specification>.
3) Without Feature S024, "Enhanced structured types", conforming SQL language shall not contain a <set clause> in which the declared type of the <update target> in the <set clause> is a structured type *TY* and the declared type of the <update source> or corresponding field of the <assigned row> contained in the <set clause> is not *TY*.
4) Without Feature S024, "Enhanced structured types", conforming SQL language shall not contain a <set clause> that contains a <mutated set clause> and in which the declared type of the last <method name> identifies a structured type *TY*, and the declared type of the <update source> contained in the <set clause> is not *TY*.
5) Without Feature T641, "Multiple column assignment", conforming SQL language shall not contain a <multiple column assignment>.

14.13 <temporary table declaration>

Function

Declare a declared local temporary table.

Format

```
<temporary table declaration> ::=
    DECLARE LOCAL TEMPORARY TABLE <table name> <table element list>
    [ ON COMMIT <table commit action> ROWS ]
```

Syntax Rules

1) Let *TN* be the <table name> of a <temporary table declaration> *TTD*, and let *T* be the <qualified identifier> of *TN*.

2) *TTD* shall be contained in an <SQL-client module definition>.

3) Case:

   a) If *TN* contains a <local or schema qualifier> *LSQ*, then *LSQ* shall be "MODULE".

   b) If *TN* does not contain a <local or schema qualifier>, then "MODULE" is implicit.

4) If a <temporary table declaration> is contained in an <SQL-client module definition> *M*, then the <qualified identifier> of *TN* shall not be equivalent to the <qualified identifier> of the <table name> of any other <temporary table declaration> that is contained in *M*.

5) The descriptor of the table defined by a <temporary table declaration> includes *TN* and the column descriptor specified by each <column definition>. The *i*-th column descriptor is given by the *i*-th <column definition>.

6) A <temporary table declaration> shall contain at least one <column definition>.

7) If ON COMMIT is not specified, then ON COMMIT DELETE ROWS is implicit.

Access Rules

None.

General Rules

1) Let *U* be the implementation-dependent <schema name> that is effectively derived from the implementation-dependent SQL-session identifier associated with the SQL-session and an implementation-dependent name associated with the SQL-client module that contains the <temporary table declaration>.

2) Let *UI* be the current user identifier and let *R* be the current role name.

Case:

a) If *UI* is not the null value, then let *A* be *UI*.

b) Otherwise, let *A* be *R*.

3) The definition of *T* within an SQL-client module is effectively equivalent to the definition of a persistent base table *U.T*. Within the SQL-client module, any reference to MODULE.*T* is equivalent to a reference to *U.T*.

4) A set of privilege descriptors is created that define the privileges INSERT, SELECT, UPDATE, DELETE, and REFERENCES on this table and INSERT, SELECT, UPDATE, and REFERENCES for every <column definition> in the table definition to *A*. These privileges are not grantable. The grantor for each of these privilege descriptors is set to the special grantor value "_SYSTEM". The grantee is "PUBLIC".

5) The definition of a temporary table persists for the duration of the SQL-session. The termination of the SQL-session is effectively followed by the execution of the following <drop table statement> with the current authorization identifier *A* and current <schema name> *U* without further Access Rule checking:

```
DROP TABLE T CASCADE
```

6) The definition of a declared local temporary table does not appear in any view of the Information Schema.

NOTE 397 — The Information Schema is defined in ISO/IEC 9075-11.

Conformance Rules

1) Without Feature F531, "Temporary tables", conforming SQL language shall not contain a <temporary table declaration>.

14.14 <free locator statement>

Function

Remove the association between a locator variable and the value that is represented by that locator.

Format

```
<free locator statement> ::=
    FREE LOCATOR <locator reference> [ { <comma> <locator reference> }... ]

<locator reference> ::=
    <host parameter name>
  | <embedded variable name>
  | <dynamic parameter specification>
```

Syntax Rules

1) Each host parameter identified by <host parameter name> immediately contained in <locator reference> shall be a binary large object locator parameter, a character large object locator parameter, an array locator parameter, a multiset locator parameter, or a user-defined type locator parameter.

2) Each host variable identified by the <embedded variable name> immediately contained in <locator reference> shall be a binary object locator variable, a character large object locator variable, an array locator variable, a multiset locator parameter, or a user-defined type locator variable.

Access Rules

None.

General Rules

1) For every <locator reference> *LR* immediately contained in <free locator statement>, let *L* be the value of *LR*.

   Case:

   a) If *L* is not a valid locator value, then an exception condition is raised: *locator exception — invalid specification.*

   b) Otherwise, *L* is marked invalid.

Conformance Rules

1) Without Feature T561, "Holdable locators", conforming SQL language shall not contain a <free locator statement>.

14.15 <hold locator statement>

Function

Mark a locator variable as being holdable.

Format

```
<hold locator statement> ::=
    HOLD LOCATOR <locator reference> [ { <comma> <locator reference> }... ]
```

Syntax Rules

1) Each host parameter identified by <host parameter name> immediately contained in <locator reference> shall be a binary large object locator parameter, a character large object locator parameter, an array locator parameter, a multiset locator parameter, or a user-defined type locator parameter.

Access Rules

None.

General Rules

1) For every <locator reference> *LR* immediately contained in <hold locator statement>, let *L* be the value of *LR*.

   Case:

   a) If *L* is not a valid locator value, then an exception condition is raised: *locator exception — invalid specification.*

   b) Otherwise, *L* is marked *holdable.*

Conformance Rules

1) Without Feature T561, "Holdable locators", conforming SQL language shall not contain a <hold locator statement>.

14.16 Effect of deleting rows from base tables

Function

Specify the effect of deleting rows from one or more base tables.

Syntax Rules

None.

Access Rules

None.

General Rules

1) Let *TT* be the set consisting of every base table that is identified for deletion processing. Let *S* be the set consisting of every row identified for deletion in some table in *TT*.
2) For every row *R* in *S*, every row *SR* that is a subrow or a superrow of *R* is identified for deletion from the base table *BT* containing *SR*, and *BT* is identified for deletion processing.
3) The current trigger execution context *CTEC*, if any, is preserved, and new trigger execution context *NTEC* is created with an empty set of state changes *SSC*.
4) For every table *T* in *TT*, for every table *ST* that is a supertable of *T* or, unless *T* is identified for deletion processing without subtables, a subtable of *T*, a state change *SC* is added to *SSC* as follows:
   a) The set of transitions of *SC* consists of one copy each of every row of *ST* that is a subrow or superrow of a member of *S*.
   b) The trigger event of *SC* is DELETE.
   c) The subject table of *SC* is *ST*.
   d) The column list of *SC* is empty.
   e) The set of statement-level triggers for which *SC* is considered as executed is empty.
   f) The set of row-level triggers consists of each row-level trigger that is activated by *SC*, paired with the empty set (of rows considered as executed).
5) The Syntax Rules and General Rules of Subclause 14.25, "Execution of BEFORE triggers", are applied with *SSC* as the *SET OF STATE CHANGES*.
6) Every row that is identified for deletion in some table identified for deletion processing is marked for deletion. These rows are no longer identified for deletion, nor are their containing tables identified for deletion processing.

   NOTE 398 — "Marking for deletion" is an implementation-dependent mechanism.

7) For every referential constraint descriptor of a constraint whose mode is immediate, the General Rules of Subclause 11.8, "<referential constraint definition>", are applied.

8) For every table *T* that is the subject table of some state change in *SSC*, each row that is marked for deletion from *T* is deleted from *T*.

   NOTE 399 — See Subclause 4.14.6, "Operations involving tables", for the effect of deleting a row from a table.

9) The Syntax Rules and General Rules of Subclause 14.26, "Execution of AFTER triggers", are applied with *SSC* as the *SET OF STATE CHANGES*.

   NOTE 400 — All constraints have already been checked for the deletion of the deleted rows of the subject table, including all referential constraints.

10) *NTEC*, together with all of its contents, is destroyed and *CTEC*, if present, is restored to become the current trigger execution context.

Conformance Rules

None.

14.17 Effect of deleting some rows from a derived table

Function

Specify the effect of deleting some rows from a derived table.

Syntax Rules

None.

Access Rules

None.

General Rules

1) Let *QE* be *TABLE* in the application of this Subclause and let *T* be the result of evaluating *QE*.
2) Case:
   a) If *QE* simply contains a <query primary> that immediately contains a <query expression body>, then let *QEB* be that <query expression body>. Apply the General Rules of Subclause 14.17, "Effect of deleting some rows from a derived table", with the table identified by *QEB* as *TABLE*.
   b) If *QE* simply contains a <query expression body> *QEB* that specifies UNION ALL, then let *LO* and *RO* be the <query expression body> and the <query term>, respectively, that are immediately contained in *QEB*. Let *T1* and *T2* be the tables identified by *LO* and *RO*, respectively.
      i) For every row *R* in *T* that has been identified for deletion, let *RD* be the row in either *T1* or *T2* from which *R* has been derived and let *TD* be that table. Identify *RD* for deletion.
      ii) The General Rules of Subclause 14.17, "Effect of deleting some rows from a derived table", are applied with *T1* as *TABLE*.
      iii) The General Rules of Subclause 14.17, "Effect of deleting some rows from a derived table", are applied with *T2* as *TABLE*.
   c) Otherwise, let *QS* be the <query specification> simply contained in *QE*. Let *TE* be the <table expression> immediately contained in *QS*, and *TREF* be the <table reference>s simply contained in the <from clause> of *TE*.
      i) Case:
         1) If *TREF* contains only one <table reference>, then let TR_1 be that <table reference>, and let *m* be 1 (one).
         2) Otherwise, let *m* be the number of <table reference>s that identify tables with respect to which *QS* is one-to-one. Let TR_i, 1 (one) $\leq i \leq m$, be those <table reference>s.

NOTE 401 — The notion of one-to-one <query specification>s is defined in Subclause 7.12, "<query specification>".

ii) Let TT_i, 1 (one) $\leq i \leq m$, be the table identified by TR_i.

iii) For every row *R* of *T* that has been identified for deletion, and for *i* ranging from 1 (one) to *m*, let *RD* be the row in TT_i from which *R* has been derived. Identify that *RD* for deletion.

iv) For *i* ranging from 1 (one) to *m*,

Case:

1) If TT_i is a base table, then

Case:

A) If TR_i specifies ONLY, then TT_i is identified for deletion processing without subtables.

B) Otherwise, TT_i is identified for deletion processing with subtables.

2) If TT_i is a viewed table, then the General Rules of Subclause 14.18, "Effect of deleting some rows from a viewed table", are applied with TR_i as *VIEW NAME*.

3) Otherwise, the General Rules of Subclause 14.17, "Effect of deleting some rows from a derived table", are applied with TR_i as *TABLE*.

Conformance Rules

None.

14.18 Effect of deleting some rows from a viewed table

Function

Specify the effect of deleting some rows from a viewed table.

Syntax Rules

None.

Access Rules

None.

General Rules

1) Let *VN* be *VIEW NAME* in the application of this Subclause.
2) If *VN* specifies ONLY, then let *QE* be the original <query expression> included in the descriptor of the view *V* identified by *VN*; otherwise, let *QE* be the <query expression> contained in that descriptor. Let *T* be the result of evaluating *QE*.
3) For each row *R* of *V* that has been identified for deletion, let *RD* be the row in *T* from which *R* has been derived; identify that row for deletion.
4) The General Rules of Subclause 14.17, "Effect of deleting some rows from a derived table", are applied with *QE* as *TABLE*.

Conformance Rules

None.

14.19 Effect of inserting tables into base tables

Function

Specify the effect of inserting each of one or more given tables into its associated base table.

Syntax Rules

None.

Access Rules

None.

General Rules

1) The current trigger execution context *CTEC*, if any, is preserved, and a new trigger execution context *NTEC* is created with an empty set of state changes *SSC*.

2) For each base table *T* that is identified for insertion, let *S* be the source table for *T*.

   a) If some column *IC* of *T* is the identity column of *T*, then for each row in *S* whose site *ICS* corresponding to *IC* is marked as *unassigned*:

      i) *ICS* is no longer marked as unassigned.

      ii) Let *NV* be the result of applying the General Rules of Subclause 9.21, "Generation of the next value of a sequence generator", with the sequence descriptor included in the column descriptor of *IC* as *SEQUENCE*.

         Case:

         1) If the declared type of *IC* is a distinct type *DIST*, then let *ICNV* be *DIST*(*NV*).

         2) Otherwise, let *ICNV* be *NV*.

      iii) The General Rules of Subclause 9.2, "Store assignment", are applied with *ICS* as *TARGET* and *ICNV* as *VALUE*.

   b) Every proper supertable *ST* of *T* is identified for insertion. A source table for insertion into each *ST* is constructed as follows:

      i) Let *S* be the source table for the insertion into *T*. Let *TVC* be some <table value constructor> whose value is *S*.

      ii) Let *n* be the number of column descriptors included in the table descriptor of *ST* and let CD_i, 1 (one) ≤ *i* ≤ *n*, be those column descriptors. Let *SL* be a <select list> containing *n* <select sublist>s such that, for *i* ranging from 1 (one) to *n*, the *i*-th <select sublist> consists of the column name included in CD_i.

iii) The source table for insertion into *ST* consists of the rows in the result of the <query expression>:

```
SELECT SL FROM TVC
```

3) For every base table *BT* that is identified for insertion, a state change *SC* is added to *SSC* as follows:
   a) The set of transitions of *SC* consists of the rows in the source table for *BT*.
   b) The trigger event of *SC* is INSERT.
   c) The subject table of *SC* is *BT*.
   d) The column list of *SC* is empty.
   e) The set of statement-level triggers for which *SC* is considered as executed is empty.
   f) The set of row-level triggers consists of each row-level trigger that is activated by *SC*, paired with the empty set (of rows considered as executed).
4) The Syntax Rules and General Rules of Subclause 14.25, "Execution of BEFORE triggers", are applied with *SSC* as the *SET OF STATE CHANGES*.
5) For every state change *SC* in *SSC*, let *SOT* be the set of transitions in *SC* and let *BT* be the subject table of *SC*.
   a) In each row *R* in *SOT*, for each site *GCS* in *R* corresponding to a generated column *GC*, let *GCR* be the result of evaluating, for R, the generation expression included in the column descriptor of *GC*. The General Rules of Subclause 9.2, "Store assignment", are applied with *GCS* as *TARGET* and *GCR* as *VALUE*.
   b) Every row in *SOT* is inserted into *BT* and *BT* is no longer identified for insertion.

      NOTE 402 — See Subclause 4.14.6, "Operations involving tables", for the effect of inserting a row into a table.
   c) For every referential constraint descriptor of a constraint whose mode is immediate, the General Rules of Subclause 11.8, "<referential constraint definition>", are applied.
6) The Syntax Rules and General Rules of Subclause 14.26, "Execution of AFTER triggers", are applied with *SSC* as the *SET OF STATE CHANGES*.

   NOTE 403 — All constraints have already been checked for the insertion of the inserted rows of the subject table, including all referential constraints.
7) *NTEC*, together with all of its contents, is destroyed and *CTEC*, if present, is restored to become the current trigger execution context.

Conformance Rules

None.

14.20 Effect of inserting a table into a derived table

Function

Specify the effect of inserting a table into a derived table.

Syntax Rules

None.

Access Rules

None.

General Rules

1) Let *Q* and *T* be the *SOURCE* and *TARGET*, respectively, in the application of this Subclause.

2) Let *QE* be the <query expression> included in the descriptor of *T*.

   Case:

   a) If *QE* simply contains a <query primary> that immediately contains a <query expression body>, then let *QEB* be that <query expression body>. Apply the General Rules of Subclause 14.20, "Effect of inserting a table into a derived table", with *Q* as *SOURCE* and the result of *QEB* as *TARGET*.

   b) Otherwise, let *QS* be the <query specification> simply contained in *QE*. Let *TE* be the <table expression> immediately contained in *QS*, and *TREF* be the <table reference>s simply contained in the <from clause> of *TE*. Let *SL* be the <select list> immediately contained in *QS*, and *n* the number of <value expression>s VE_j, 1 (one) $\leq j \leq n$, simply contained in *SL*.

      i) Case:

         1) If *TREF* contains only one <table reference>, then let TR_1 be that <table reference>, and let *m* be 1 (one).

         2) Otherwise, let *m* be the number of <table reference>s that identify tables with respect to which *QS* is one-to-one. Let TR_i, 1 (one) $\leq i \leq m$, be those <table reference>s.

      ii) Let TT_i, 1 (one) $\leq i \leq m$, be the table identified by TR_i, and let S_i be an initially empty table of candidate rows for TT_i.

      iii) For every row *R* of *Q*, and for *i* ranging from 1 (one) to *m*:

         1) A candidate row of TT_i is effectively created in which the value of each column is its default value, as specified the General Rules of Subclause 11.5, "<default clause>". The candidate row includes every column of TT_i.

2) For j ranging from 1 (one) to n, let C be a column of some candidate row identified by VE_j, and let SV be the j-th value of R. The General Rules of Subclause 9.2, "Store assignment", are applied to C and SV as *TARGET* and *SOURCE*, respectively.

3) The candidate row is added to the corresponding S_i.

iv) For i ranging from 1 (one) to m,

Case:

1) If TT_i is a base table, then TT_i is identified for insertion of source table S_i.

2) If TT_i is a viewed table, the General Rules of Subclause 14.21, "Effect of inserting a table into a viewed table", are applied with S_i as *SOURCE* and TT_i as *TARGET*.

3) Otherwise, the General Rules of Subclause 14.20, "Effect of inserting a table into a derived table", are applied with S_i as *SOURCE* and TT_i as *TARGET*.

Conformance Rules

None.

14.21 Effect of inserting a table into a viewed table

Function

Specify the effect of inserting a table into a viewed table.

Syntax Rules

None.

Access Rules

None.

General Rules

1) Let *S* and *T* be the *SOURCE* and *TARGET*, respectively, in application of this Subclause. Let *TD* be the view descriptor of *T*. Let *QE* be the original <query expression> included in *TD*.

2) If *TD* indicates WITH CHECK OPTION, then:

   a) Case:

      i) If *TD* specifies LOCAL, then let *VD* be a view descriptor derived from *TD* as follows:

         1) The WITH CHECK OPTION indication is removed.

         2) Every reference contained in *QE* to a leaf underlying table *LUT* of *QE* is replaced by a reference to a temporary table consisting of a copy of *LUT*.

      ii) Otherwise, let *VD* be a view descriptor derived from *TD* as follows:

         1) The WITH CHECK OPTION indication is removed.

         2) Every reference contained in *QE* to an underlying table *UV* of *QE* that is a viewed table is replaced by a reference to a view whose descriptor is identical to that of *UV* except that WITH CASCADED CHECK OPTION is indicated.

         3) Every reference contained in *QE* to a leaf underlying table *LUT* of *QE* that is a base table is replaced by a reference to a temporary table consisting of a copy of *LUT*.

   b) The General Rules of this Subclause are applied with *S* as *SOURCE* and the view *V* described by *VD* as *TARGET*.

   c) If the result of

```
EXISTS ( SELECT * FROM S
         EXCEPT ALL
         SELECT * FROM V )
```

   is <u>*True*</u>, then an exception condition is raised: *with check option violation*.

3) The General Rules of Subclause 14.20, “Effect of inserting a table into a derived table”, are applied, with *S* as *SOURCE* and *QE* as *TARGET*.

Conformance Rules

None.

14.22 Effect of replacing rows in base tables

Function

Specify the effect of replacing some of the rows in one or more base tables.

Syntax Rules

None.

Access Rules

None.

General Rules

1) Let *TT* be the set consisting of every base table that is identified for replacement processing. Let *S* be the set consisting of every row identified for replacement in every table in *TT*.

2) For every base table *T* in *TT*, let *OC* be the set consisting of every object column with respect to which *T* is identified for replacement processing and every generated column of *T* that depends on at least one of these object columns. Every table *ST* that is a subtable or supertable of *T* is identified for replacement processing with respect to the intersection (possibly empty) of *OC* and the columns of *ST*.

3) For every row *R* that is identified for replacement in some table *T* in *TT*, every row *SR* that is a subrow or a superrow of *R* is identified for replacement in the base table *ST* that contains *SR*. The replacement set *RST* for *ST* is derived from the replacement set *RR* for *T* as follows.

   Case:

   a) If *ST* is a subtable of *T*, each replacement row in *RST* is the corresponding replacement row in *RR* extended with those fields of the corresponding identified row in *ST* that have no corresponding column in *T*.

   b) If *ST* is a supertable of *T*, each replacement row in *RST* is the corresponding replacement row in *RR* minus those fields that have no corresponding column in *ST*.

4) The current trigger execution context *CTEC*, if any, is preserved and a new trigger execution context *NTEC* is created with an empty set of state changes *SSC*.

5) For every table *T* in *TT*, for every table *ST* that is a supertable of *T* or, unless *T* is identified for replacement processing without subtables, a subtable of *T*, let *TL* be the set consisting of the names of the columns of *ST*. For every subset *STL* of *TL* such that either *STL* is empty or the intersection of *STL* and *OC* is not empty:

   a) If some column *IC* of *T* is the identity column of *ST*, then for each row identified for replacement in *ST* whose site *ICS* corresponding to *IC* is marked as *unassigned*:

i) Let *NV* be the result of applying the General Rules of Subclause 9.21, "Generation of the next value of a sequence generator", with the sequence descriptor included in the column descriptor of *IC* as *SEQUENCE*.

Case:

1) If the declared type of *IC* is a distinct type *DIST*, then let *ICNV* be *DIST*(*NV*).

2) Otherwise, let *ICNV* be *NV*.

ii) The General Rules of Subclause 9.2, "Store assignment", are applied with *ICS* as *TARGET* and *ICNV* as *VALUE*.

b) All sites in *ST* that are marked as unassigned cease to be so marked.

c) A state change *SC* is added to *SSC* as follows:

i) The set of transitions of *SC* consists of row pairs formed by pairing each row identified for replacement in *ST* with its corresponding replacement row.

ii) The trigger event of *SC* is UPDATE.

iii) The subject table of *SC* is *ST*.

iv) The column list of *SC* is *STL*.

v) The set of statement-level triggers for which *SC* is considered as executed is empty.

vi) The set of row-level triggers consists of each row-level trigger that is activated by *SC*, paired with the empty set (of rows considered as executed).

6) The Syntax Rules and General Rules of Subclause 14.25, "Execution of BEFORE triggers", are applied with *SSC* as the *SET OF STATE CHANGES*.

7) For each set of transitions *RST* in each state change *SC* in *SSC*, in each row *R* in *RST*, for each site *GCS* in *R* corresponding to a generated column *GC* in the subject table of *SC*, let *GCR* be the result of evaluating, for *R*, the generation expression included in the column descriptor of *GC*. The General Rules of Subclause 9.2, "Store assignment", are applied with *GCS* as *TARGET* and *GCR* as *VALUE*.

8) For every table *T* in *TT*, for every table *ST* that is a supertable or a subtable of *T*, for every row *R* that is identified for replacement in *ST*, *R* is replaced by its new transition variable. *R* is no longer identified for replacement. *ST* is no longer identified for replacement processing.

9) For every referential constraint descriptor of a constraint whose mode is immediate, the General Rules of Subclause 11.8, "<referential constraint definition>", are applied.

10) The Syntax Rules and General Rules of Subclause 14.26, "Execution of AFTER triggers", are applied with *SSC* as the *SET OF STATE CHANGES*.

NOTE 404 — All constraints have already been checked for the update of the replaced rows of the identified tables, including all referential constraints.

11) *NTEC*, along with all of its contents, is destroyed and *CTEC*, if present, is restored to become the current trigger execution context.

Conformance Rules

None.

14.23 Effect of replacing some rows in a derived table

Function

Specify the effect of replacing some rows in a derived table.

Syntax Rules

None.

Access Rules

None.

General Rules

1) Let *QE* be the *TABLE* and *RS* the replacement for *TABLE* in the application of this Subclause.
2) Let *T* be the result of evaluating *QE*. Let *CL* be the object columns of *QE*.
3) Case:
   a) If *QE* simply contains a <query primary> that immediately contains a <query expression body>, then let *QEB* be that <query expression body>. Apply the General Rules of Subclause 14.23, "Effect of replacing some rows in a derived table", with *TR* as the table identified by *QEB*, and with *RS* as the replacement set for *TR*.
   b) If *QE* simply contains a <query expression body> *QEB* that specifies UNION ALL, let *LO* and *RO* be the <query expression body> and the <query term>, respectively, that are immediately contained in *QEB*. Let *T1* and *T2* be the tables identified by *LO* and *RO*, respectively. Let the columns of *T1* and *T2* that are underlying columns of the object columns of *CL* be the object columns *CL1* and *CL2*, respectively. Let *RS1* and *RS2* be the initially empty replacement sets for *T1* and *T2*, respectively.
      i) For every pair (*SR*, *CNR*) of *RS*:

         Case:
         1) If *SR* has been derived from a row of *T1*, then identify that row *SR1* for replacement by *CNR*; the pair (*SR1*, *CNR*) is effectively added to *RS1*.
         2) Otherwise, let *SR2* be the row of *T2* from which *SR* has been derived; identify that row for replacement by *CNR*; the pair (*SR2*, *CNR*) is effectively added to *RS2*.
      ii) The General Rules of Subclause 14.23, "Effect of replacing some rows in a derived table", are applied with *T1* as *TABLE*.
      iii) The General rules of Subclause 14.23, "Effect of replacing some rows in a derived table", are applied with *T2* as *TABLE*.

c) Otherwise, let *QS* be the <query specification> simply contained in *QE*. Let *TE* be the <table expression> immediately contained in *QS*, and let *TREF* be the <table reference>s simply contained in the <from clause> of *TE*. Let *SL* be the <select list> immediately contained in *QS*, and let *n* be the number of <value expression>s VE_j, 1 (one) $\leq j \leq n$, simply contained in *SL*.

   i) Case:

      1) If *TREF* contains only one <table reference>, then let TR_1 be that <table reference>, and let *m* be 1 (one).

      2) Otherwise, let *m* be the number of <table reference>s that identify tables with respect to which *QS* is one-to-one. Let TR_i, 1 (one) $\leq i \leq m$, be those <table reference>s.

   ii) Let TT_i, 1 (one) $\leq i \leq m$, be the table identified by TR_i, let RS_i be an initially empty replacement set for TT_i, and let CL_i be the object column list of TT_i, such that every column of CL_i is an underlying column of *CL*.

   iii) For every pair (*SR*, *CNR*) of *RS*, and for *i* ranging from 1 (one) to *m*:

      1) Let *SRTI* be the row of TT_i from which *SR* has been derived.

      2) A candidate row *CNRI* of TT_i is effectively created in which the value of each column is its default value, as specified the General Rules of Subclause 11.5, "<default clause>". The candidate row includes every column of TT_i.

      3) For *j* ranging from 1 (one) to *n*, let *C* be a column of some candidate row identified by VE_j, and let *SV* be the *j*-th value of *R*. The General Rules of Subclause 9.2, "Store assignment", are applied to *C* and *SV* as *TARGET* and *SOURCE*, respectively.

      4) Identify *SRTI* for replacement by *CNRI*; the pair (*SRTI*, *CNRI*) is effectively added to SR_i.

   iv) For *i* ranging from 1 (one) to *m*

      Case:

      1) If TT_i is a base table, then

         Case:

         A) If TR_i specifies ONLY, then TT_i is identified for replacement processing without subtables with respect to the object columns CL_i.

         B) Otherwise, TT_i is identified for replacement processing with subtables with respect to the object columns CL_i.

      2) If TT_i is a viewed table, then the General rules of Subclause 14.24, "Effect of replacing some rows in a viewed table", are applied with TR_i as *VIEW NAME*.

      3) If TT_i is a derived table, then the General rules of Subclause 14.23, "Effect of replacing some rows in a derived table", are applied with TR_i as *TABLE*.

Conformance Rules

None.

14.24 Effect of replacing some rows in a viewed table

Function

Specify the effect of replacing some rows in a viewed table.

Syntax Rules

None.

Access Rules

None.

General Rules

1) Let *T* be the *VIEW NAME* and *RS* the replacement set for *VIEW NAME* in application of this Subclause. Let *TD* be the view descriptor of *T*. If *VN* specifies ONLY, then let *QE* be the original <query expression> included in *TD*; otherwise, let *QE* be the <query expression> included in *TD*.

2) If *TD* indicates WITH CHECK OPTION, then:

   a) Case:

      i) If *TD* specifies LOCAL, then let *VD* be a view descriptor derived from *TD* as follows:

         1) The WITH CHECK OPTION indication is removed.

         2) Every reference contained in *QE* to a leaf underlying table *LUT* of *QE* is replaced by a reference to a temporary table consisting of a copy of *LUT*.

      ii) Otherwise, let *VD* be a view descriptor derived from *TD* as follows.

         1) The WITH CHECK OPTION indication is removed.

         2) Every reference contained in *QE* to an underlying table *UV* of *QE* that is a viewed table is replaced by a reference to a view whose descriptor is identical to that of *UV* except that WITH CASCADED CHECK OPTION is indicated.

         3) Every reference contained in *QE* to a leaf underlying table *LUT* of *T* that is a base table is replaced by a reference to a temporary table consisting of a copy of *LUT*.

   b) The General Rules of this Subclause are applied with the view *V* described by *VD* as *VIEW NAME* and *RS* as the replacement set for *V*.

   c) Let *S* be the table consisting of the candidate new rows of *RS*. If the result of

```
EXISTS ( SELECT * FROM S
         EXCEPT ALL
         SELECT * FROM V )
```

is *True*, then an exception condition is raised: *with check option violation*.

3) The General Rules of Subclause 14.23, “Effect of replacing some rows in a derived table”, are applied with *QE* as *TABLE* and *RS* as the replacement set for *QE*.

Conformance Rules

None.

14.25 Execution of BEFORE triggers

Function

Define the execution of BEFORE triggers.

Syntax Rules

1) Let *SSC* be the *SET OF STATE CHANGES* specified in an application of this Subclause.

2) Let *BT* be the set of BEFORE triggers that are activated by some state change in *SSC*.

   NOTE 405 — Activation of triggers is defined in Subclause 4.38, "Triggers".

3) Let *NT* be the number of triggers in *BT* and let TR_k be the *k*-th such trigger, ordered according to their order of execution. Let SC_k be the state change in *SSC* that activated TR_k.

   NOTE 406 — Ordering of triggers is defined in Subclause 4.38, "Triggers".

Access Rules

None.

General Rules

1) For *k* ranging from 1 (one) to *NT*, apply the General Rules of Subclause 14.27, "Execution of triggers", with TR_k as *TRIGGER* and SC_k as *STATE CHANGE*, respectively.

Conformance Rules

None.

14.26 Execution of AFTER triggers

Function

Define the execution of AFTER triggers.

Syntax Rules

1) Let *SSC* be the *SET OF STATE CHANGES* specified in an application of this Subclause.
2) Let *AT* be the set of AFTER triggers that are activated by some state change in *SSC*.

   NOTE 407 — Activation of triggers is defined in Subclause 4.38, "Triggers".
3) Let *NT* be the number of triggers in *AT* and let TR_k be the *k*-th such trigger, ordered according to their order of execution. Let SC_k be the state change in *SSC* that activated TR_k.

   NOTE 408 — Ordering of triggers is defined in Subclause 4.38, "Triggers".

Access Rules

None.

General Rules

1) For *k* ranging from 1 (one) to *NT*, apply the General Rules of Subclause 14.27, "Execution of triggers", with TR_k as *TRIGGER* and SC_k as *STATE CHANGE*, respectively.

Conformance Rules

None.

14.27 Execution of triggers

Function

Define the execution of triggers.

Syntax Rules

None.

Access Rules

None.

General Rules

1) Let *TR* and *SC* be respectively a *TRIGGER* and a *STATE CHANGE* in an application of this Subclause.

2) Let *TA* be the triggered action included in the trigger descriptor of *TR*. Let *TSS* be the <triggered SQL statement> contained in *TA*. Let *TE* be the trigger event of *SC*. Let *ST* be the set of transitions in *SC*.

3) *TR* is executed as follows.

   Case:

   a) If *TR* is a row-level trigger, then, for each transition *T* in *SC* for which *TR* is not considered as executed, *TA* is invoked and *TR* is considered as executed for *T*. The order in which the transitions in *SC* are taken is implementation-dependent.

   b) If *TR* is not considered as executed for *SC*, then *TA* is invoked once and *TR* is considered as executed for *SC*.

4) When *TA* is invoked:

   a) Case:

      i) If *TE* is DELETE, then the old transition table for the invocation of *TA* is *ST*. If *TR* is a row-level trigger, then the value of the old transition variable for the execution of *TSS* is *T*.

      ii) If *TE* is INSERT, then the new transition table for the invocation of *TA* is *ST*. If *TR* is a row-level trigger, then the value of the new transition variable for the invocation of *TA* is *T*.

      iii) If *TE* is UPDATE, then the old transition table for the invocation of *TA* is the multiset formed by taking the old rows of the transitions in in *ST* and the new transition table for the invocation of *TA* is the multiset formed by taking the new rows of the transitions in *ST*. If *TR* is a row-level trigger, then the value of the old transition variable for the invocation of *TA* is the old row of *T* and the new transition variable for the invocation of *TA* is the new row of *T*.

   b) Case:

   i) If *TA* contains a <search condition> *TASC* and the result of evaluating *TASC* is *true*, then *TSS* is executed.

   ii) If *TA* does not contain a <search condition>, then *TSS* is executed.

5) When *TSS* is executed:

   a) The General Rules of Subclause 22.2, "Pushing and popping the diagnostics area stack", are applied with "PUSH" as *OPERATION* and the diagnostics area stack as *STACK*.

   b) The authorization identifier of the owner of the schema that includes the trigger descriptor of *TR* is pushed onto the authorization stack.

   c) A new savepoint level is established.

   d) Let *N* be the number of <SQL procedure statement>s simply contained in *TSS*. For *i* ranging from 1 (one) to *N*:

      i) Let S_i be the *i*-th such <SQL procedure statement>.

      ii) The General Rules of Subclause 13.5, "<SQL procedure statement>", are evaluated with S_i as the executing statement.

   e) The <SQL procedure statement>s simply contained in *TSS* are effectively executed in the order in which they are specified in *TSS*.

   f) If, before the completion of the execution of any <SQL procedure statement> simply contained in *TSS*, an attempt is made to execute an SQL-schema statement, an SQL-dynamic statement, or an SQL-session statement then an exception condition is raised: *prohibited statement encountered during trigger execution*.

   g) If *TR* is a BEFORE trigger and if, before the completion of the execution of any <SQL procedure statement> simply contained in *TSS*, an attempt is made to execute an SQL-data change statement or an SQL-invoked routine that possibly modifies SQL-data, then an exception condition is raised: *prohibited statement encountered during trigger execution*.

   h) The current savepoint level is destroyed.

   NOTE 409 — Destroying a savepoint level destroys all existing savepoints that are established at that level.

   i) The General Rules of Subclause 22.2, "Pushing and popping the diagnostics area stack", are applied with "POP" as *OPERATION* and the diagnostics area stack as *STACK*.

   j) The top cell in the authorization stack is removed.

   k) If the execution of *TSS* is not successful, then an exception condition is raised: *triggered action exception*. The exception condition that caused *TSS* to fail is raised.

   NOTE 410 — Raising the exception condition that caused *TSS* to fail enters the exception information into the diagnostics area that was pushed prior to the execution of *TSS*.

Conformance Rules

None.

15 Control statements

15.1 <call statement>

Function

Invoke an SQL-invoked routine.

Format

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

Syntax Rules

1) Let *RI* be the <routine invocation> immediately contained in the <call statement>.
2) Let *SR* be the subject routine specified by applying the Syntax Rules of Subclause 10.4, "<routine invocation>", to *RI*.
3) *SR* shall be an SQL-invoked procedure.

Access Rules

None.

General Rules

1) *SR* is effectively invoked according to the General Rules of Subclause 10.4, "<routine invocation>", with *RI* and *SR* as the <routine invocation> and the subject routine, respectively.

Conformance Rules

None.

15.2 <return statement>

Function

Return a value from an SQL function.

Format

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

<return value> ::=
    <value expression>
  | NULL
```

Syntax Rules

1) <return statement> shall be contained in an SQL routine body that is simply contained in the <routine body> of an <SQL-invoked function> *F*. Let *RDT* be the <returns data type> of the <returns clause> of *F*.

2) The <return value> <null specification> is equivalent to the <value expression>:

```
CAST (NULL AS RDT)
```

3) Let *VE* be the <value expression> of the <return value> immediately contained in <return statement>.

4) The declared type of *VE* shall be assignable to an item of the data type *RDT*, according to the Syntax Rules of Subclause 9.2, "Store assignment", with *RDT* and *VE* as *TARGET* and *VALUE*, respectively.

Access Rules

None.

General Rules

1) The value of *VE* is the *returned value* of the execution of the SQL routine body of *F*.

2) The execution of the SQL routine body of *F* is terminated immediately.

Conformance Rules

None.

16 Transaction management

16.1 <start transaction statement>

Function

Start an SQL-transaction and set its characteristics.

Format

```
<start transaction statement> ::=
    START TRANSACTION
    [ <transaction mode> [ { <comma> <transaction mode> }... ] ]

<transaction mode> ::=
    <isolation level>
  | <transaction access mode>
  | <diagnostics size>

<transaction access mode> ::=
    READ ONLY
  | READ WRITE

<isolation level> ::= ISOLATION LEVEL <level of isolation>

<level of isolation> ::=
    READ UNCOMMITTED
  | READ COMMITTED
  | REPEATABLE READ
  | SERIALIZABLE

<diagnostics size> ::= DIAGNOSTICS SIZE <number of conditions>

<number of conditions> ::= <simple value specification>
```

Syntax Rules

1) None of <isolation level>, <transaction access mode>, and <diagnostics size> shall be specified more than once in a single <start transaction statement>.

2) If <start transaction statement> contains at least one <transaction mode>, then:

   a) If an <isolation level> is not specified, then a <level of isolation> of SERIALIZABLE is implicit.

   b) If READ WRITE is specified, then the <level of isolation> shall not be READ UNCOMMITTED.

c) If a <transaction access mode> is not specified and a <level of isolation> of READ UNCOMMITTED is specified, then READ ONLY is implicit. Otherwise, READ WRITE is implicit.

d) If <number of conditions> is not specified, then an implementation-dependent value not less than 1 (one) is implicit for <number of conditions>.

3) The declared type of <number of conditions> shall be exact numeric with scale 0 (zero).

Access Rules

None.

General Rules

1) If a <start transaction statement> statement is executed when an SQL-transaction is currently active, then an exception condition is raised: *invalid transaction state — active SQL-transaction.*

2) Let *TXN* be the SQL-transaction that will be started after successful execution of <start transaction statement>.

3) If <start transaction statement> contains no <transaction mode>, then:

   a) The isolation level of *TXN* is set to the enduring transaction characteristic of isolation level.

   b) The transaction access mode level of *TXN* is set to the enduring transaction characteristic of access mode.

   c) The number of conditions of *TXN* is set to the enduring transaction characteristic of diagnostics size.

4) If <number of conditions> is specified and is less than 1 (one), then an exception condition is raised: *invalid condition number.*

5) If READ ONLY is specified, then the access mode of *TXN* is set to *read-only*. If READ WRITE is specified, then the access mode of *TXN* is set to *read-write.*

6) The isolation level of *TXN* is set to an implementation-defined isolation level that will not exhibit any of the phenomena that the explicit or implicit <level of isolation> would not exhibit, as specified in Table 8, "SQL-transaction isolation levels and the three phenomena".

7) If <number of conditions> is specified, then the condition area limit of *TXN* is set to <number of conditions>.

   NOTE 411 — The characteristics of a transaction begun by a <start transaction statement> are as specified in these General Rules regardless of the characteristics specified by any preceding <set transaction statement>. That is, even if one or more characteristics are omitted by the <start transaction statement>, the defaults specified in the Syntax Rules of this Subclause are effective and are not affected by any (preceding) <set transaction statement>.

8) *TXN* is started.

Conformance Rules

1) Without Feature T241, "START TRANSACTION statement", conforming SQL language shall not contain a <start transaction statement>.

**

2) Without Feature F111, “Isolation levels other than SERIALIZABLE”, conforming SQL language shall not contain an <isolation level> that contains a <level of isolation> other than SERIALIZABLE.

3) Without Feature F121, “Basic diagnostics management”, conforming SQL language shall not contain a <diagnostics size>.

16.2 <set transaction statement>

Function

Set the characteristics of the next SQL-transaction for the SQL-agent.

NOTE 412 — This statement has no effect on any SQL-transactions subsequent to the next SQL-transaction.

Format

```
<set transaction statement> ::=
    SET [ LOCAL ] <transaction characteristics>

<transaction characteristics> ::=
    TRANSACTION <transaction mode> [ { <comma> <transaction mode> }... ]
```

Syntax Rules

1) None of <isolation level>, <transaction access mode>, and <diagnostics size> shall be specified more than once in a single <transaction characteristics>.

2) If LOCAL is specified, then <number of conditions> shall not be specified.

Access Rules

None.

General Rules

1) Case:

   a) If a <set transaction statement> that does not specify LOCAL is executed, then

      Case:

      i) If an SQL-transaction is currently active, then an exception condition is raised: *invalid transaction state — active SQL-transaction.*

      ii) If an SQL-transaction is not currently active, then if there are any holdable cursors remaining open from the previous SQL-transaction and the isolation level of the previous SQL-transaction is not the same as the isolation level determined by the <level of isolation>, then an exception condition is raised: *invalid transaction state — held cursor requires same isolation level.*

   b) If a <set transaction statement> that specifies LOCAL is executed, then:

      i) If the SQL-implementation does not support SQL-transactions that affect more than one SQL-server, then an exception condition is raised: *feature not supported — multiple server transactions.*

      ii) If there is no SQL-transaction that is currently active, then an exception condition is raised: *invalid transaction state — no active SQL-transaction for branch transaction.*

iii) If there is an active SQL-transaction and there has been a transaction-initiating SQL-statement executed at the current SQL-connection in the context of the active SQL-transaction, then an exception condition is raised: *invalid transaction state — branch transaction already active.*

iv) If the transaction access mode of the SQL-transaction is read-only and <transaction access mode> specifies READ WRITE, then an exception condition is raised: *invalid transaction state — inappropriate access mode for branch transaction.*

v) If the isolation level of the SQL-transaction is SERIALIZABLE and <level of isolation> specifies anything except SERIALIZABLE, then an exception condition is raised: *invalid transaction state — inappropriate isolation level for branch transaction.*

vi) If the isolation level of the SQL-transaction is REPEATABLE READ and <level of isolation> specifies anything except REPEATABLE READ or SERIALIZABLE, then an exception condition is raised: *invalid transaction state — inappropriate isolation level for branch transaction.*

vii) If the isolation level of the SQL-transaction is READ COMMITTED and <level of isolation> specifies READ UNCOMMITTED, then an exception condition is raised: *invalid transaction state — inappropriate isolation level for branch transaction.*

NOTE 413 — If the isolation level of the SQL-transaction is READ UNCOMMITTED, then any <level of isolation> is permissible.

2) If <number of conditions> is specified and is less than 1 (one), then an exception condition is raised: *invalid condition number.*

3) Case:

a) If LOCAL is not specified, then let *TXN* be the next SQL-transaction for the SQL-agent.

b) Otherwise, let *TXN* be the branch of the active SQL-transaction at the current SQL-connection.

4) If READ ONLY is specified, then the access mode of *TXN* is set to *read-only*. If READ WRITE is specified, then the access mode of *TXN* is set to *read-write.*

5) The isolation level of *TXN* is set to an implementation-defined isolation level that will not exhibit any of the phenomena that the explicit or implicit <level of isolation> would not exhibit, as specified in Table 8, "SQL-transaction isolation levels and the three phenomena".

6) If <number of conditions> is specified, then the condition area limit of *TXN* is set to <number of conditions>.

7) If <number of conditions> is not specified, then the condition area limit of *TXN* is set to an implementation-dependent value not less than 1 (one).

Conformance Rules

1) Without Feature T251, "SET TRANSACTION statement: LOCAL option", conforming SQL language shall not contain a <set transaction statement> that immediately contains LOCAL.

16.3 <set constraints mode statement>

Function

If an SQL-transaction is currently active, then set the constraint mode for that SQL-transaction in the current SQL-session. If no SQL-transaction is currently active, then set the constraint mode for the next SQL-transaction in the current SQL-session for the SQL-agent.

NOTE 414 — This statement has no effect on any SQL-transactions subsequent to this SQL-transaction.

Format

```
<set constraints mode statement> ::=
    SET CONSTRAINTS <constraint name list> { DEFERRED | IMMEDIATE }

<constraint name list> ::=
    ALL
  | <constraint name> [ { <comma> <constraint name> }... ]
```

Syntax Rules

1) If a <constraint name> is specified, then it shall identify a constraint.

2) The constraint identified by <constraint name> shall be DEFERRABLE.

Access Rules

None.

General Rules

1) If an SQL-transaction is currently active, then let *TXN* be the currently active SQL-transaction. Otherwise, let *TXN* be the next SQL-transaction for the SQL-agent.

2) If IMMEDIATE is specified, then

   Case:

   a) If ALL is specified, then the constraint mode in *TXN* of all constraints that are DEFERRABLE is set to *immediate*.

   b) Otherwise, the constraint mode in *TXN* for the constraints identified by the <constraint name>s in the <constraint name list> is set to *immediate*.

3) If DEFERRED is specified, then

   Case:

   a) If ALL is specified, then the constraint mode in *TXN* of all constraints that are DEFERRABLE is set to *deferred*.

b) Otherwise, the constraint mode in *TXN* for the constraints identified by the <constraint name>s in the <constraint name list> is set to *deferred.*

Conformance Rules

1) Without Feature F721, “Deferrable constraints”, conforming SQL language shall not contain a <set constraints mode statement>.

16.4 <savepoint statement>

Function

Establish a savepoint.

Format

```
<savepoint statement> ::= SAVEPOINT <savepoint specifier>

<savepoint specifier> ::= <savepoint name>
```

Syntax Rules

None.

Access Rules

None.

General Rules

1) Let *S* be the <savepoint name>.
2) If *S* identifies an existing savepoint established within the current savepoint level, then that savepoint is destroyed.
3) If the number of savepoints that now exist within the current SQL-transaction is equal to the implementation-defined maximum number of savepoints per SQL-transaction, then an exception condition is raised: *savepoint exception — too many.*
4) A savepoint is established in the current savepoint level and at the current point in the current SQL-transaction. *S* is assigned as the identifier of that savepoint.

Conformance Rules

1) Without Feature T271, "Savepoints", conforming SQL language shall not contain a <savepoint statement>.

16.5 <release savepoint statement>

Function

Destroy a savepoint.

Format

```
<release savepoint statement> ::= RELEASE SAVEPOINT <savepoint specifier>
```

Syntax Rules

None.

Access Rules

None.

General Rules

1) Let *S* be the <savepoint name>.
2) If *S* does not identify a savepoint established in the current savepoint level, then an exception condition is raised: *savepoint exception — invalid specification.*
3) The savepoint identified by *S* and all savepoints established in the current savepoint level subsequent to the establishment of *S* are destroyed.

Conformance Rules

1) Without Feature T271, "Savepoints", conforming SQL language shall not contain a <release savepoint statement>.

16.6 <commit statement>

Function

Terminate the current SQL-transaction with commit.

Format

```
<commit statement> ::= COMMIT [ WORK ] [ AND [ NO ] CHAIN ]
```

Syntax Rules

1) If neither AND CHAIN nor AND NO CHAIN is specified, then AND NO CHAIN is implicit.

Access Rules

None.

General Rules

1) If the current SQL-transaction is part of an encompassing transaction that is controlled by an agent other than the SQL-agent, then an exception condition is raised: *invalid transaction termination.*

2) If an atomic execution context is active, then an exception condition is raised: *invalid transaction termination.*

3) For every open cursor that is not a holdable cursor *CR* in any SQL-client module associated with the current SQL-transaction, the following statement is implicitly executed:

```
CLOSE CR
```

4) For every temporary table in any SQL-client module associated with the current SQL-transaction that specifies the ON COMMIT DELETE option and that was updated by the current SQL-transaction, the execution of the <commit statement> is effectively preceded by the execution of a <delete statement: searched> that specifies DELETE FROM *T*, where *T* is the <table name> of that temporary table.

5) The effects specified in the General Rules of Subclause 16.3, "<set constraints mode statement>" occur as if the statement SET CONSTRAINTS ALL IMMEDIATE were executed for each active SQL-connection.

6) Case:

   a) If any constraint is not satisfied, then any changes to SQL-data or schemas that were made by the current SQL-transaction are canceled and an exception condition is raised: *transaction rollback — integrity constraint violation.*

   b) If the execution of any <triggered SQL statement> is unsuccessful, then any changes to SQL-data or schemas that were made by the current SQL-transaction are canceled and an exception condition is raised: *transaction rollback — triggered action exception.*

c) If any other error preventing commitment of the SQL-transaction has occurred, then any changes to SQL-data or schemas that were made by the current SQL-transaction are canceled and an exception condition is raised: *transaction rollback* with an implementation-defined subclass value.

d) Otherwise, any changes to SQL-data or schemas that were made by the current SQL-transaction are eligible to be perceived by all concurrent and subsequent SQL-transactions.

7) All savepoint levels are destroyed and a new savepoint level is established.

NOTE 415 — Destroying a savepoint level destroys all existing savepoints that are established at that level.

8) Every valid non-holdable locator value is marked invalid.

9) The current SQL-transaction is terminated. If AND CHAIN was specified, then a new SQL-transaction is initiated with the same access mode, isolation level, and diagnostics area limit as the SQL-transaction just terminated. Any branch transactions of the SQL-transaction are initiated with the same access mode, isolation level, and diagnostics area limit as the corresponding branch of the SQL-transaction just terminated.

10) The <statement name> or <extended statement name> of every held cursor remains valid.

Conformance Rules

1) Without Feature T261, “Chained transactions”, conforming SQL language shall not contain a <commit statement> that immediately contains CHAIN.

16.7 <rollback statement>

Function

Terminate the current SQL-transaction with rollback, or rollback all actions affecting SQL-data and/or schemas since the establishment of a savepoint.

Format

```
<rollback statement> ::= ROLLBACK [ WORK ] [ AND [ NO ] CHAIN ] [ <savepoint clause> ]

<savepoint clause> ::= TO SAVEPOINT <savepoint specifier>
```

Syntax Rules

1) If AND CHAIN is specified, then <savepoint clause> shall not be specified.

2) If neither AND CHAIN nor AND NO CHAIN is specified, then AND NO CHAIN is implicit.

Access Rules

None.

General Rules

1) If the current SQL-transaction is part of an encompassing transaction that is controlled by an agent other than the SQL-agent and the <rollback statement> is not being implicitly executed, then an exception condition is raised: *invalid transaction termination.*

2) If a <savepoint clause> is not specified, then:

   a) If an atomic execution context is active, then an exception condition is raised: *invalid transaction termination.*

   b) All changes to SQL-data or schemas that were made by the current SQL-transaction are canceled.

   c) All savepoint levels are destroyed and a new savepoint level is established.

      NOTE 416 — Destroying a savepoint level destroys all existing savepoints that are established at that level.

   d) Every valid locator is marked invalid.

   e) All open cursors in any SQL-client module associated with the current SQL-transaction are closed.

   f) The current SQL-transaction is terminated. If AND CHAIN was specified, then a new SQL-transaction is initiated with the same access mode, isolation level, and diagnostics area limit as the SQL-transaction just terminated. Any branch transactions of the SQL-transaction are initiated with the same access mode, isolation level, and diagnostics area limit as the corresponding branch of the SQL-transaction just terminated.

3) If a <savepoint clause> is specified, then:

   a) Let *S* be the <savepoint name>.

   b) If *S* does not specify a savepoint established within the current savepoint level, then an exception condition is raised: *savepoint exception — invalid specification.*

   c) If an atomic execution context is active, and *S* specifies a savepoint established before the beginning of the most recent atomic execution context, then an exception condition is raised: *savepoint exception — invalid specification.*

   d) All changes to SQL-data or schemas that were made by the current SQL-transaction subsequent to the establishment of *S* are canceled.

   e) All savepoints established by the current SQL-transaction subsequent to the establishment of *S* are destroyed.

      NOTE 417 — Destroying a savepoint level destroys all existing savepoints that are established at that level.

   f) Every valid locator that was generated in the current SQL-transaction subsequent to the establishment of *S* is marked invalid.

   g) For every open cursor *CR* in any SQL-client module associated with the current SQL-transaction that was opened subsequent to the establishment of *S*, the following statement is implicitly executed:

      ```
      CLOSE CR
      ```

   h) The status of any open cursors in any SQL-client module associated with the current SQL-transaction that were opened by the current SQL-transaction before the establishment of *S* is implementation-defined.

      NOTE 418 — The current SQL-transaction is not terminated, and there is no other effect on the SQL-data or schemas.

Conformance Rules

1) Without Feature T271, "Savepoints", conforming SQL language shall not contain a <savepoint clause>.

2) Without Feature T261, "Chained transactions", conforming SQL language shall not contain a <rollback statement> that immediately contains CHAIN.

This page intentionally left blank.