ISO/IEC JTC 1/SC 21 N 11157

Date: 1997-10-31

ISO/IEC CD 13249-2:199x (E)

ISO/IEC JTC 1/SC 21/WG 3

Secretariat: Canada

Information Technology —Database Languages —

SQL Multimedia and Application Packages —

Part 2: Full-Text

Contents **Page**

Foreword

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

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 3.

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

International Standard ISO/IEC 13249 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information Technology*.

This document is based on the content of ISO/IEC Committee Draft Database Language (SQL).

This is the first edition of ISO/IEC SQL/MM – Part 2: Full-Text.

Introduction

The purpose of this International Standard is to define multimedia and application specific types and their associated routines using the user-defined features in ISO/IEC 9075.

SQL/MM is structured as a multi-part standard. At present it consists of the following parts:

Part 1:	Framework
Part 2:	Full-Text
Part 3:	Spatial
Part 4:	General Purpose Facilities
Part 5:	Still Image

Information Technology — Database Languages — SQL Multimedia and Application Packages — Part 2: Full-Text

1 Scope

This part of ISO SQL/MM:

a) introduces this part (Full-text) of this International Standard,

b) gives the references necessary for this part of this International Standard,

c) defines notations and conventions specific to this part of this International Standard,

d) defines concepts specific to this part of this International Standard,

e) defines the full-text data types and their associated routines.

2 Normative references

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

2.1 International standards

ISO/IEC 9075-1:199x, *Information Technology — Database Languages — SQL — Part 1: Framework (SQL/Framework).*

ISO/IEC 9075-2:199x, *Information Technology — Database Languages — SQL — Part 2: Foundation (SQL/Foundation).*

ISO/IEC 9075-4:1996, *Information Technology — Database Languages — SQL — Part 4: Persistent Stored Modules (SQL/PSM).*

ISO/IEC 9075-5:199x, *Information Technology — Database Languages — SQL — Part 5: Host Language Bindings (SQL/Bindings).*

ISO/IEC 13249-1:199x, *Information Technology — Database Languages — SQL Multimedia and Application*

2.2 Publicly available standards

ANSI/NISO Z39.19-1993, American National Standard for Information Systems/National Information Standards Organization, *Guidelines for the Construction, Format, and Management of Monolingual Thesauri.*

3 Definitions, notations, and conventions

3.1 Definitions

3.1.1 Definitions taken from ISO/IEC 9075

This International Standard makes use of the following terms defined in ISO/IEC 9075:

3.1.1.1
immediately contained

3.1.1.2
simply contained

3.1.1.3
SQL-invoked routine

3.1.2 Definitions taken from ANSI/NISO Z39.19

This International Standard makes use of the following terms defined in ANSI/NISO Z39.19:

3.1.2.1
thesaurus

3.1.3 Definitions provided in this part of ISO/IEC 13249

3.1.3.1
broader term
A superordinate term in a hierarchical relation (e.g. a broader term for "SQL" is "Database Language").

3.1.3.2
coordinate relation
A formal relation juxtaposing terms or classes of terms.

3.1.3.3
hierarchical relation
A formal relation between two terms or classes in which one term is subordinate to the other term.

3.1.3.4
narrower term
A subordinate term in a hierarchical relation (e.g a narrower term for "SQL" is "SQL/MM").

3.1.3.5
preferred term
A term chosen as a descriptor from a set of equivalent terms (e.g. a preferred term for "Structured Query Language" is "SQL").

3.1.3.6
related term
A term connected to another term by a coordinate relation (e.g. a related term for "SQL" is "DB2").

3.1.3.7
soundex term
A term having a different form though its pronunciation is similar to another term. (e.g. a soundex term for "there" is "their").

3.1.3.8
synonym term
A term having a different form but a similar meaning to another term (e.g. a synonym term for "SQL/MM" is "SQL Multimedia and Application Packages").

3.1.3.9
top term
The broadest term in a hierarchical relation. If it is defined that "Computer Language" is a broader term of "Database Language, then the top term of "SQL" is "Computer Language".

3.2 Notations

The notation used in ISO/IEC 13249-2 is defined in ISO/IEC 9075-1.

3.3 Conventions

3.3.1 Subclause structure

Subclauses for a type definition and its associated routines will be structured as follows:

x.3.5 FT_SampleDataType Type and Routines

x.3.5.1 FT_SampleDataType Type

x.3.5.2 FT_UserDefinedFunction1 Function

x.3.5.3 FT_UserDefinedFunction2 Function

x.3.5.4 FT_UserDefinedFunction3 Function

3.3.2 Structure within a subclause

1) **Purpose**: The Purpose section contains a brief description of the purpose of the type or routine.

2) **Definition**: This section contains the ISO/IEC 9075 syntax used to define the type or routine. In the case of routine specifications, the routine body should be defined using the facilities of ISO/IEC 9075-4 (SQL/PSM) where possible. This clause is in the Courier font. <key word>s, as defined in ISO/IEC 9075, are in uppercase. Parameter and variable identifiers are in lower case or mixed case. Data type, attribute and SQL-invoked routine identifiers are specified as defined in Subclause 3.3.3, "Data type, attribute and SQL-invoked routine identifiers."

3) **Definitional Rules**: This section contains an enumerated list of rules to be applied when defining the type or routine. If this section is empty, the section heading is not present.

4) **Description**: This section contains an enumerated list of rules describing the type or routine. For a type, the first item contains a statement indicating the attributes and routines that are part of the public specification. For a routine, the first item contains the definition of the routine's parameters. If this section is empty, the section heading is not present.

3.3.3 Data type, attribute and SQL-invoked routine identifiers

Data type identifiers, attribute identifiers and routine identifiers:

1) shall have a prefix. Full-Text uses "FT_."

2) shall not use the underbar character except in the prefix (i.e. only Alphanumeric characters [a-zA-Z0-9]),

3) shall capitalize each word used in the identifier. For example: FT_Primary,

4) shall be in Italic when used in the Definitional Rules and the Description sections.

3.3.4 Parameter identifiers

Parameter identifiers are in lowercase. Parameters are in Italic when used in the Definitional Rules and the Description sections. This will distinguish them from other identifiers used in the Definitional Rules and Description sections.

3.3.5 Meta-variables

Meta-variables used to define implementation-dependent or implementation-defined constants such as FT_MaxTextLength follow the conventions of Subclause 3.3.3, "Data type, attribute and SQL-invoked

3.3.6 Definitional variables

Definitional variables used in the Definitional Rules or Description sections are in uppercase Italics. This will distinguish them from other identifiers in the Definitional Rules or Description sections.

3.3.7 Exceptions

Except where otherwise specified, the phrase "an exception condition is raised:," followed by the name of a condition, is used in the Definitional Rules and Description sections to indicate that:

– The execution of a routine is unsuccessful.

– Application of Definitional Rules or Description items may be terminated.

The effect on any assignment target and SQL descriptor area of an SQL-statement that terminates with an exception condition, unless explicitly defined by ISO/IEC 9075, is implementation-dependent.

The phrase "a completion condition is raised:," followed by the name of a condition, is used in Definitional Rules and Description sections to indicate that application of Definitional Rules or Description items is not terminated and diagnostic information is to be made available; unless an exception condition is raised, the execution of the SQL-statement is successful.

4 Concepts

4.1 Schemas

ISO/IEC 9075 specifies that an object such as an SQL-invoked routine, a user-defined type, a domain, a table, a view, or a privilege is part of exactly one schema.

This International Standard does not include statements for creating schemas. An implementation-defined set of <schema definition> statements shall be effectively executed such that each <schema definition> statement that contains a <schema element> for a schema object defined in this International Standard shall contain exactly one <schema element> for each object defined by this International Standard. The number of such schemas and their names is implementation-defined.

It is assumed that the default character set of the SQL-schema in which an SQL-invoked routine specified in this International Standard is created includes the characters used in all character string literals contained in the body of that routine and a space character denoted by a blank space in such literals.

4.2 USAGE Privileges on User-defined Types

ISO/IEC 9075 specifies that users must have the USAGE privilege on a domain or a user-defined type before they can use it for defining other objects such as SQL-invoked routines, tables, view, domains or user-defined types.

This International Standard does not include the GRANT USAGE statements for the domains and user-defined types defined in this International Standard. For each object defined by this International Standard, a GRANT statement granting USAGE privilege to an implementation-defined set of grantees shall be effectively executed when these domains and user-defined types are created, except when explicitly noted by the Definitional Rules in this International Standard. It is implementation-defined whether the GRANT statement includes WITH GRANT OPTION.

4.3 UNDER Privileges on User-defined Types

ISO/IEC 9075 specifies users must have the UNDER privilege on a user-defined type *A* before they can use it for defining subtypes of *A*.

This International Standard does not include the GRANT UNDER statements for the user-defined types defined in this International Standard. For each object defined by this International Standard, a GRANT statement granting UNDER privilege to an implementation-defined set of grantees shall be effectively executed when these user-defined types are created, except when explicitly noted by the Definitional Rules in this International Standard. It is implementation-defined whether the GRANT statement includes WITH GRANT OPTION.

4.4 EXECUTE Privileges on Routines

ISO/IEC 9075 specifies that users must have the EXECUTE privilege on a routine before they can execute it.

This International Standard does not include the GRANT EXECUTE statements for the routines defined in this standard. For each routine defined by this International Standard, a GRANT statement granting EXECUTE privilege to an implementation-defined set of grantees shall be effectively executed when the routine is created, except where explicitly noted by the Definitional Rules in this International Standard. It is implementation-defined whether the GRANT statement includes WITH GRANT OPTION.

Editor's Note 2-045

The Concepts clause should contain a general overview on the purpose and functionality intended with the Full-Text part.

5 Full-text Data Types

The types in this family provide for the construction of text and search patterns for searching of text.

5.1 FullText Type and Routines

5.1.1 FullText Type

Purpose

The *FullText* type provides for the construction of text, for testing whether text contains specified patterns, and for turning text into character strings.

Definition

```
CREATE TYPE FullText
   (Contents CHARACTER VARYING(FT_MaxTextLength))

CREATE CAST (FullText AS CHARACTER VARYING(FT_MaxTextLength)
   WITH FullText_to_Character)
```

Definitional Rules

1) The attribute *Contents* is not part of the public interface. There are no GRANT statements granting EXECUTE privilege to the observer or mutator function for *Contents*.

2) *FT_MaxTextLength* is the implementation-defined maximum length for the character representation of an instance of *FullText*.

Description

1) The *FullText* type provides for public use:

   a) a function *FullText(CHARACTER VARYING)* to construct a *FullText* instance from a character string,
   b) a CAST to cast a *FullText* instance into a character string using the function *FullText(FullText),*
   c) a function *Contains(FullText, FT_Pattern),*
   d) a function *Contains(FullText, CHARACTER VARYING),*
   e) a function *StrctPattern_to_FT_Pattern(FullText_Token ARRAY).*

5.1.2 FullText_to_Character Function

Purpose

Return the character representation of a *FullText* instance.

Definition

```
CREATE FUNCTION FullText_to_Character
   (text FullText)
   RETURNS CHARACTER VARYING(FT_MaxTextLength)
   BEGIN
      RETURN text>>Contents;
   END
```

Definitional Rules

1) *FT_MaxTextLength* is the implementation-defined maximum length for the character representation of an instance of *FullText*.

Description

1) The function *FullText_to_Character(FullText)* takes the following input parameters:

   a) a *FullText* value *text*.

5.1.3 FullText Function

Purpose

Construct and initialize a *FullText* instance.

Definition

```
CREATE FUNCTION FullText
   (string CHARACTER VARYING(FT_MaxTextLength))
   RETURNS FullText
   BEGIN
      DECLARE temp FullText;
      SET temp = FullText();
      SET temp>>Contents = string;
      RETURN temp;
   END
```

Editor's Note 2-047

The *FullText* constructor function should be modified to forbid a NULL input value so as to prohibit a NULL *Contents* attribute.

Definitional Rules

1) *FT_MaxTextLength* is the implementation-defined maximum length for the character representation of an instance of *FullText*.

Description

1) The function *FullText (CHARACTER VARYING)* takes the following input parameters:

   a) a *CHARACTER VARYING* value *string*.

5.1.4 Contains Functions

Purpose

Search a *FullText* instance for an *FT_Pattern*.

Definition

```
CREATE FUNCTION Contains
   (text FullText,
    pattern FT_Pattern)
   RETURNS Boolean
   BEGIN
      --
      -- !! See Description
      --
   END

CREATE FUNCTION Contains
   (text FullText,
    pattern CHARACTER VARYING(FT_MaxPatternLength))
   RETURNS Boolean
   BEGIN
      RETURN Contains(text, CAST(pattern AS FT_Pattern));
   END
```

Definitional Rules

1) *FT_MaxPatternLength* is the implementation-dependent maximum length for the character representation of an instance of *FT_Pattern*.

Description

1) The function *Contains(FullText, FT_Pattern)* takes the following input parameters:

   a) a *FullText* value *text*,
   b) an *FT_Pattern* value *pattern*.

2) The function *Contains(FullText, CHARACTER VARYING)* takes the following input parameters:

   a) a *FullText* value *text*,
   b) a *CHARACTER VARYING* value *pattern*.

3) The result of the invocation *Contains(text, pattern)* of *Contains(FullText, FT_Pattern)* is determined as follows:

   Case:

   a) If the value of *pattern* does not have the format of a <search expression>, then an exception condition is raised: *SQL/MM Full-Text* - *invalid search expression*.
   b) Otherwise:
      Case:
      i) If *text*, *text>>Contents*, or *pattern* is the null value, unknown.
      ii) Otherwise, let *s_pattern* be the structured pattern of type *FT_Expr*, such that

SQL/MM Full-Text does not specify a function that could be used to return data which includes an indication of where a search predicate was satisfied. Many implementations support this by "match codes" which are special system-defined characters that are optionally placed in the return data to flag the portions which matched the search criteria.

This type of facility is related the Issue 2-009 "Match codes" as recorded in SQL/MM MAD-004. We believe that this type of facility is at least a Language Opportunity for "Later Progression".

Editor's Note 2-049

SQL/MM Full-Text does not specify a mechanism that could be used to return a "relevance" value to indicated "how well" a search predicate was satisfied. This could be accomplished by defining another function which takes the same parameters as the CONTAINS function but that returns a "relevance" value (instead of BOOLEAN) which could be tested in the WHERE clause. Many implementations define a RELEVANCE function which can be used in the <select list> of a SELECT statement so that the "relevance" value is accessible as part of the derived table.

If a "relevance" facility is added to SQL/MM Full-Text then we believe that the BNF productions for <word> and <phrase> (see Subclause 5.2, "FT_Pattern Distinct Type") should permit the specification of term weights which can be used in some "relevance" algorithms.

This type of facility is related the Issue 2-008 "Relevance ranking" as recorded in SQL/MM MAD-004. We believe that this type of facility is at least a Language Opportunity for "Later Progression".

5.1.5 StrctPattern_to_FT_Pattern Function

Purpose

Convert a sequence of *FullText_Token* values to an *FT_Pattern* value.

Definition

```
CREATE FUNCTION StrctPattern_to_FT_Pattern
   (tokarray FullText_Token ARRAY[FT_MaxArrayLength])
   RETURNS FT_Pattern
   BEGIN
      DECLARE result CHARACTER VARYING(FT_MaxTextLength);
      DECLARE i INTEGER;

      SET i = 1;
      IF tokarray IS NULL THEN
         RETURN CASE(NULL AS FT_Pattern);
      SET result = '(';
      WHILE i <= CARDINALITY(tokarray) DO
         SET result = result || '"'
            || CAST(InsertDQS(tokarray[i])
                  AS CHARARACTER VARYING(FT_MaxPatternLength))
            || '",';
         SET i = i + 1;
      END WHILE;
      SET result = TRIM(TRAILING ‘,’ FROM result) || ')';
      RETURN CAST(result AS FT_Pattern);
   END
```

Definitional Rules

1) *FT_MaxTextLength* is the implementation-defined maximum length for the character representation of an instance of *FullText*.

2) *FT_MaxPatternLength* is the implementation-dependent maximum length for the character representation of an instance of *FT_Pattern*.

3) *FT_MaxArrayLength* is the implementation-dependent maximum length for an array.

Description

1) The function *StrctPattern_to_FT_Pattern(FullText_Token ARRAY)* takes the following input parameters:

   a) an array *tokarray* whose elements are *FullText_Token* instances.

2) The function *StrctPattern_to_FT_Pattern(FullText_Token ARRAY)* returns an *FT_Pattern* instance of the form <token list>.

3) If the input argument *tokarray* is the null value or if any element of *tokarray* is the null value, the function *StrctPattern_to_FT_Pattern(FullText_Token ARRAY)* returns the null value.

5.1.6 Tokenize Function

Purpose

Convert a *FullText* value into a sequence of *FullText_Token* values.

Definition

```
CREATE FUNCTION Tokenize
   (text FullText)
   RETURNS FullText_Token ARRAY[FT_MaxArrayLength]
   BEGIN
      --
      -- !! See Description
      --
   END
```

Definitional Rules

1) *FT_MaxArrayLength* is the implementation-dependent maximum length for an array.

Description

1) The private function *Tokenize(FullText)* takes the following input parameters:

   a) a *FullText* value *text*.

2) *Tokenize*(*FullText)* returns an array representing a sequence of *FullText_Token* items. The result of *Tokenize* is the null value if *text* or *text>>Contents* is the null value.

3) Further details of the relationship between input and output are implementation-defined. Though not enforced by this standard, it is intended that *Tokenize*(*FullText)* reflects the language structure of the input text being processed.

5.1.7 TokenizePosition Function

Purpose

Convert a *FullText* value into a sequence of *FT_TokenPosition* values.

Definition

```
CREATE FUNCTION TokenizePosition
   (text FullText,
    unit FullText_Token)
   RETURNS FT_TokenPosition ARRAY[FT_MaxArrayLength]
   BEGIN
      --
      -- !! See Description
      --
   END
```

Definitional Rules

1) *FT_MaxArrayLength* is the implementation-dependent maximum length for an array.

Description

1) The private function *TokenizePosition(FullText, FullText_Token)* takes the following input parameters:

   a) a *FullText* value *text*.
   b) a *FullText_Token* value *unit* identifying a unit of text.

2) The unit information supported by an implementation is implementation-defined but shall include 'CHARACTERS', 'WORDS', 'SENTENCES' or 'PARAGRAPHS'.

3) The function *TokenizePosition(FullText, FullText_Token)* returns an array representing a set of *FT_TokenPosition* items with the attributes:

   a) A *FullText_Token* value *token* representing a word occurring in *text*.

   b) An INTEGER value *position* identifying the position of an occurrence of *token* in terms of the *unit* information specified (e.g. "third sentence").

   c) An INTEGER value *corrVal*. This value is intended to support the computation of the distance between two words as identify by two *FT_TokenPosition* items. *corrVal* is zero for the distance units 'WORDS', 'SENTENCES' and 'PARAGRAPHS'; its value is implementation-defined for any other distance unit, including the 'CHARACTERS' unit. In the latter case, possible values are zero, or values related to the length of the associated token.

   Let *t1* and *t2* be two *FT_TokenPosition* values. An implementation shall define the contents of the attribute *corrVal* in such a way that distance between *t1>>token* and *t2>>token* is given by:

   ```
   t2>>position - t1>position - t1>>corrVal
   ```

   provided *t1* precedes *t2* (i.e. *t2>>position >= t1>>position*).

4) The result of *TokenizePosition(FullText, FullText_Token)* shall be the null value if:

   a) *text* or *text>>Contents* is the null value.

   b) *unit* is the null value or a value not supported by the implementation.

4) Further details of the relationship between input and output are implementation-defined. Though not enforced by this standard, it is intended that *TokenizePosition(FullText, FullText_Token)* reflects the language structure of the input text being processed.

5.1.8 Segmentize Function

Purpose

Convert a *FullText* value into a sequence of *FullText* values.

Definition

```
CREATE FUNCTION Segmentize
   (text FullText,
    unit FullText_Token)
   RETURNS FullText ARRAY[FT_MaxArrayLength]
   BEGIN
      --
      -- !! See Description
      --
   END
```

Definitional Rules

1) *FT_MaxArrayLength* is the implementation-dependent maximum length for an array.

Description

1) The private function *Segmentize(FullText, FullText_Token)* takes the following input parameters:

   a) a *FullText* value *text*,
   b) a *FullText_Token* value *unit*.

2) The *unit* shall be either 'SENTENCE' or 'PARAGRAPH'.

3) *Segmentize(FullText, FullText_Token)* returns an array of *FullText* instances, which are either sentences or paragraphs of text. For every sentence (paragraph) of text there shall be exactly one element in the resulting array the content of which equals the content of this sentence (paragraph). The relative order of resulting array elements shall be the same as the order of the associated sentences (paragraphs).

5.2 FT_TokenPosition Type and Routines

5.2.1 FT_TokenPosition Type

Purpose

The *FT_TokenPosition* type provides facilities for the construction of data items intended to represent occurrences of words in some text.

Definition

```
CREATE TYPE FT_TokenPosition
   (token FullText_Token,
    position INTEGER,
    corrVal INTEGER)
```

Description

1) The *FT_TokenPosition* type provides for public use:

   a) an attribute *token*,
   b) an attribute *position*,
   c) an attribute *corrVal*.

2) The purpose of the *FT_TokenPosition* attributes is described in Subclause 5.1.7, "TokenizePosition Function" which is used to initialize these attributes.

5.3 FT_Pattern Type and Routines

5.3.1 FT_Pattern Type

Purpose

The *FT_Pattern* type provides for linear search patterns.

Definition

```
CREATE TYPE FT_Pattern
   AS CHARACTER VARYING(FT_MaxPatternLength)
```

Definitional Rules

1) *FT_MaxPatternLength* is the implementation-dependent maximum length for the character representation of an instance of *FT_Pattern*.

Description

1) The *FT_Pattern* type provides for public use a CHARACTER VARYING value.

2) Instances of type *FT_Pattern* are meant as input to the function *Contains(FullText, FT_Pattern)* of the data type *FullText*.

   NOTE: *FullText* type is described in Subclause 5.1.1, "FullText Type."

3) Instances of *FT_Pattern* must be producible from the following BNF for <search expression>.

```
<search expression> ::=
     <search term>
   | <search expression> <vertical bar> <search term>

<vertical bar> ::= |

<search term> ::=
     <search factor>
   | <search term> <ampersand> <search factor>

<ampersand> ::= &

<search factor> ::=
   [ NOT ] <search primary>

<search primary> ::=
     <text literal>
   | <text function invocation>
   | <context condition>
   | <left paren> <search expression> <right paren>

<text literal> ::=
     <word>
   | <phrase>

<word> ::=
   <double quote><word representation><double quote>
```

```
          [ <escape specification> ]

     <escape specification> ::=
        ESCAPE <double quote> <escape representation character>
           <double quote>

     <escape representation character> ::=
        !! See Description

     <phrase> ::=
        <double quote><phrase representation><double quote>
          [ <escape specification> ]

     <word representation> ::= <word representation part> ...

     <word representation part> ::=
          <word representation character>
        | <doublequote symbol>

     <word representation character> ::= !! See Description

     <doublequote symbol> ::=
        !! See Subclause 5.2, <token> and <separator>, in part 2
        !!    of ISO 9075

     <phrase representation> ::=
        <phrasepart representation> [<word separator>] <phrasepart
        representation>
        [ { [<word separator>] <phrasepart representation>} ... ]

     <phrasepart representation> ::=
          <word representation>
        | <optional word represtation>

     <optional word representation> ::=  %

     <word separator> ::= !! See Description

     <text function invocation> ::=
          <Proximity function invocation>
        | <Is_About function invocation>
        | <expansion function invocation>

     <expansion function invocation> ::=
          <Soundex_Exp function invocation>
        | <Broader_Term function invocation>
        | <Narrower_Term function invocation>
        | <Synonym function invocation>
        | <Preferred_Term function invocation>
        | <Related_Term function invocation>
        | <Top_Term function invocation>
```

Editor's Note 2-050

We believe that, in addition to soundex and thesaurus facilities, there should be provision for a linguistic processing facility that could be applied to a search term in the same manner as the aforementioned facilities. For example, an input verb could be conjugated through all possibilities of first, second and third person by singular and plural forms; nouns could be supplied in singular and plural forms, etc.

This type of facility is related the Issue 2-004 "Approximate searching" as recorded in SQL/MM MAD-004. We believe that this type of linguistic processing is at least a Language Opportunity for "Later Progression".

Editor's Note 2-051

We believe that it is useful to be able to specify that various search options and facilities should apply "globally" to an FT_Pattern instance. This would mean, for example, that instead of having to apply an <expansion function invocation> to every search term, one could indicate that the designated function should apply to every search term in the query, unless overridden by a specific <expansion function invocation>.

```
<Proximity function invocation> ::=
   PROXIMITY <left paren> <token list1>
      <comma> <distance>
      <comma> <unit>
      <comma> <order>
      <comma> <token list2>
      <right paren>

<left paren> ::= (

<right paren> ::= )

<comma> ::= ,

<token list1> ::=
     <token list>
   | <expansion function invocation>

<token list2> ::=
     <token list>
   | <expansion function invocation>

<token list> ::=
   <left paren> <word> [ { <comma> <word> }... ] <right paren>


<distance> ::= <unsigned integer>

<unsigned integer> ::=
   !! See Subclause 5.3, <literal>, in part 2 of ISO 9075

<unit> ::=
     CHARACTERS
   | WORDS
   | SENTENCES
   | PARAGRAPHS
   | !! See Description

<order> ::=
     ANY_ORDER
   | IN_ORDER
   | !! See Description

<Soundex_Exp function invocation> ::=
   SOUNDEX_EXP <left paren> <word> <right paren>
```

```
<Broader_Term function invocation> ::=
   BROADER_TERM <left paren> <thesaurus specification>
      <comma> <word>
      <comma> <thesaurus expansion count>
      <right paren>

<thesaurus specification> ::= !! See Description

<thesaurus expansion count> ::= <unsigned integer>

<Narrower_Term function invocation> ::=
   NARROWER_TERM <left paren> <thesaurus specification>
      <comma> <word>
      <comma> <thesaurus expansion count>
      <right paren>

<Synonym function invocation> ::=
   SYNONYM <left paren> <thesaurus specification>
      <comma> <word>
      <right paren>

<Preferred_Term function invocation> ::=
   PREFERRED_TERM <left paren> <thesaurus specification>
      <comma> <word>
      <right paren>

<Related_Term function invocation> ::=
   RELATED_TERM <left paren> <thesaurus specification>
      <comma> <word>
      <right paren>

<Top_Term function invocation> ::=
   TOP_TERM <left paren> <thesaurus specification>
      <comma> <word>
      <right paren>

<context condition> ::=
   <context argument> IN SAME <context unit> AS
   <context argument> [ { AND <context argument> } ... ]


<context unit> ::=
     SENTENCE
   | PARAGRAPH

<context argument> ::=
     <text literal>
   | <text literal list>
   | <expansion function invocation>

<text literal list> ::=
   <left paren> <text literal>
     [ { <comma> <text literal> } ... ] <right paren>

<Is_About function invocation> ::=
   IS_ABOUT <left paren> <phrase> <right paren>
```

a) A <word representation> is a non-empty sequence of <word representation part>s. A <word representation part> is either a <word representation character> or a <doublequote symbol>. The set of <word representation character>s does not contain <double quote>. Other than that, the set of <word representation character>s is implementation-defined; though not enforced by this standard, it is intended that the corresponding rules reflect the characteristics of the specific language from which the word has been taken. Wildcard characters '_'and '%'shall be among the admissible characters; however, a <word representation> shall contains at least one character that is not treated as a wildcard character.

If a <word representation> *WR* is immediately contained in a <word> or <phrase> which immediately contains an <escape specification> *ES*, then let *E* be the <escape representation character> immediately contained in *ES*. If *WR* contains either a '%'or an '_'that is preceded by *E*, those characters represent a '%'or an '_', and not a wildcard character. If an *E* is preceded by an *E*, the second *E* does not represent an <escape representation character>.

A <word representation> immediately contained in a <word> which is immediately contained in a <Soundex_Exp function invocation>, <Broader_Term function invocation>, <Narrower_Term function invocation>, <Synonym function invocation>, <Preferred_Term function invocation>, <Related_Term function invocation>, or <Top_Term function invocation> shall not contain a <word representation character> that is treated as a wildcard character. In addition the containing <word> shall not specify an <escape specification>.

b) A <phrase representation> is a sequence (two or more items) of <phrasepart representation>s. It is implementation-defined whether a specific <word separator> character is needed between two consecutive <phrasepart representation>s; though not enforced by this standard, it is intended that the corresponding rules reflect the characteristics of the specific language in which the phrase is being expressed. A <phrasepart representation shall contain at least one <word representation>.

Note: If a <phrasepart representation> *PPR* is simple contained in a <phase> which specifies an <escape specification> *ES*, then let *E* be the <escape character> immediately contained in *ES*. If *PPR* is *E*% then *PPR* does not represent an optional word.

4) Let *T* and *P* be a *FullText* instance and a *FT_Pattern* respectively. The value of Contains(*T*, *P*) is determined by the following:

a) If *P* is a <search expression> of the form *SE* <vertical bar> *ST*, then the result of

```
Contains(T, P)
```

is the result of

```
Contains(T, SE) OR Contains(T, ST)
```

b) If *P* is a <search term> of the form *ST* <ampersand> *SF*, then the result of

```
Contains(T, P)
```

is the result of

```
Contains(T, ST) AND Contains(T, SF)
```

c) If *P* is a <search factor> of the form NOT *SP*, then the result of

```
Contains(T, P)
```

is the result of

```
NOT Contains(T, SP)
```

d) If *P* is a <word> *W*, let *STL* be an *FT_TextLiteral* instance such that

```
W = StrctPattern_to_FT_Pattern(STL)
```

then the result of

```
Contains(T, W)
```

is the result of

```
Contains(T, STL)
```

(i.e. Contains returns *true* if *T* contains at least one token which matches *W*).

NOTE: A word pattern *W* may contain wildcard characters '_' (denoting a single character from the character set of <search expression>) or '%' (denoting a string of any length (zero or more) composed of characters from the character set of <search expression>).

****Editor's Note 2-038****
As outlined in SQL/MM LGW-023, different character sets can be supported by defining different "packages" of the Full-Text data types and routines for each character set to be supported. Each "package" would then be defined in a different schema to differentiate the character set to be supported. Explicit support for this design must be added to this International Standard.

****Editor's Note 2-039****
As outlined in SQL/MM LGW-023, different character sets can be supported by defining different "packages" of the Full-Text data types and routines for each character set to be supported. To access a specific *Contains* function the user must be able to set the SQL PATH (not currently supported in Core SQL3) or must be able to specify the schema explicitly. Explicit support for this design must be added to this International Standard.

****Editor's Note 2-043****
SQL/MM LGW-023 dealt with the problems of specifying the character set of a FullText or FT_Primary value. As demonstrated by SQL/MM LGW-024 the problem of specifying the language of an FT_Pattern value has not yet been tackled.

e) If *P* is a <phrase> *PHR*, let *SPP* be an *FT_Phrase* instance such that

```
PHR = StrctPattern_to_FT_Pattern(SPP)
```

then

```
Contains(T, PHR)
```

is the result of

```
Contains(T, SPP)
```

(i.e. *T* contains a contiguous sequence of tokens which match *PHR*).

NOTE: A token of *PHR* may be composed of wildcard characters only. If such a token consists of one or more '%'s, it denotes an optional word.

f) If *P* is a <Proximity function invocation> *PFI*, then let *TL1* be <token list1> and *TL2* be <token list2>.

Case:

i) If *TL1* is a <Broader_Term function invocation>, let *SBT* be an *FT_BroaderTerm* instance such that *TL1* is equal to *StrctPattern_to_FT_Pattern*(*SBT*). Replace *TL1* in *PFI* by the result of:

```
StrctPattern_to_FT_Pattern(GetBroaderTerms(SBT>>thesaurus,
   SBT>>startingTerm, SBT>>expansionCnt))
```

If *TL2* is a <Broader_Term function invocation>, let *SBT* be an *FT_BroaderTerm* instance such that *TL2* is equal to *StrctPattern_to_FT_Pattern*(*SBT*). Replace *TL2* in *PFI* by the result of:

```
StrctPattern_to_FT_Pattern(GetBroaderTerms(SBT>>thesaurus,
   SBT>>startingTerm, SBT>>expansionCnt))
```

ii) If *TL1* is a <Narrower_Term function invocation>, let *SBT* be an *FT_NarrowerTerm* instance such that *TL1* is equal to *StrctPattern_to_FT_Pattern(SBT*). Replace *TL1* in *PFI* by the result of:

```
StrctPattern_to_FT_Pattern(GetNarrowerTerms(SBT>>thesaurus,
   SBT>>startingTerm, SBT>>expansionCnt))
```

If *TL2* is a <Narrower_Term function invocation>, let *SBT* be an *FT_NarrowerTerm* instance such that *TL2* is equal to *StrctPattern_to_FT_Pattern(SBT*). Replace *TL2* in *PFI* by the result of:

```
StrctPattern_to_FT_Pattern(GetNarrowerTerms(SBT>>thesaurus,
   SBT>>startingTerm, SBT>>expansionCnt))
```

iii) If *TL1* is a <Synonym function invocation>, let *SST* be an *FT_Synonym* instance such that *TL1* is equal to *StrctPattern_to_FT_Pattern(SST*). Replace *TL1* in *PFI* by the result of:

```
StrctPattern_to_FT_Pattern(GetSynonymTerms(SST>>thesaurus,
   SST>>startingTerm))
```

If *TL2* is a <Synonym function invocation>, let *SST* be an *FT_Synonym* instance such that *TL2* is equal to *StrctPattern_to_FT_Pattern(SST*). Replace *TL2* in *PFI* by the result of:

```
StrctPattern_to_FT_Pattern(GetSynonymTerms(SST>>thesaurus,
   SST>>startingTerm))
```

iv) If *TL1* is a <Preferred_Term function invocation>, let *SPT* be an *FT_PreferredTerm* instance such that *TL1* is equal to *StrctPattern_to_FT_Pattern(SPT*). Replace *TL1* in *PFI* by the result of:

```
StrctPattern_to_FT_Pattern(GetPreferredTerms(SPT>>thesaurus,
   SPT>>startingTerm))
```

If *TL2* is a <Preferred_Term function invocation>, let *SPT* be an *FT_PreferredTerm* instance such that *TL2* is equal to *StrctPattern_to_FT_Pattern(SPT*). Replace *TL2* in *PFI* by the result of:

```
StrctPattern_to_FT_Pattern(GetPreferredTerms(SPT>>thesaurus,
   SPT>>startingTerm))
```

v) If *TL1* is a <Related_Term function invocation>, let *SRT* be an *FT_RelatedTerm* instance such that *TL1* is equal to *StrctPattern_to_FT_Pattern(SRT*). Replace *TL1* in *PFI* by the result of:

```
StrctPattern_to_FT_Pattern(GetRelatedTerms(SRT>>thesaurus,
   SRT>>startingTerm))
```

If *TL2* is a <Related_Term function invocation>, let *SRT* be an *FT_RelatedTerm* instance such that *TL2* is equal to *StrctPattern_to_FT_Pattern(SRT*). Replace *TL2* in *PFI* by the result of:

```
StrctPattern_to_FT_Pattern(GetRelatedTerms(SRT>>thesaurus,
   SRT>>startingTerm))
```

vi) If *TL1* is a <Top_Term function invocation>, let *STT* be an *FT_TopTerm* instance such that *TL1* is equal to *StrctPattern_to_FT_Pattern(STT*). Replace *TL1* in *PFI* by the result of:

```
StrctPattern_to_FT_Pattern(GetTopTerms(STT>>thesaurus,
   STT>>startingTerm))
```

If *TL2* is a <Top_Term function invocation>, let *STT* be an *FT_TopTerm* instance such that *TL2* is equal to *StrctPattern_to_FT_Pattern*(*STT*). Replace *TL2* in *PFI* by the result of:

```
StrctPattern_to_FT_Pattern(GetTopTerms(STT>>thesaurus,
   STT>>startingTerm))
```

vii) If *TL1* is a <Soundex function invocation>, let *SPHT* be an *FT_Soundex* instance such that *TL1* is equal to *StrctPattern_to_FT_Pattern*(*SPHT*). Replace *TL1* in *PFI* by the result of:

```
StrctPattern_to_FT_Pattern(GetSoundsSimilar(SPHT>>spoken))
```

If *TL2* is a <Soundex function invocation>, let *SPHT* be an *FT_Soundex* instance such that *TL2* is equal to *StrctPattern_to_FT_Pattern*(*SPHT*). Replace *TL2* in *PFI* by the result of:

```
   StrctPattern_to_FT_Pattern(GetSoundsSimilar(SPHT>>spoken))
```

Let *SPR* be an *FT_Proxi* instance such that

```
PFI = StrctPattern_to_FT_Pattern(SPR)
```

then the result of

```
Contains(T, PFI)
```

is the result of

```
Contains(T, SPR)
```

Editor's Note 2-052
We believe that the power of the <Proximity function invocation> could be significantly improved by permitting it to contain an arbitrary number of <token list>s instead of the current two lists. The semantics would be that at least one token from each of the <token list>s occur in the designated FullText instance, and that the distance between the first occurrence and the last not exceed the specified <distance> measured in the specified <unit>s. Note that <context condition> is structured to support an arbitrary number of <token list>s and changes to the <Proximity function invocation> could be patterned after <context condition>.

Editor's Note 2-053
The <Proximity function invocation> production uses the <token list1> and <token list2> productions which in turn permit <word>s but NOT <phrase>s to be used within the proximity search. Phrases must be valid within a proximity search.

Editor's Note 2-054
The <Proximity function invocation> production uses the <token list1> and <token list2> productions which in turn permit <word>s but NOT <expansion function invocation>s to be used with the proximity search. It would be useful to permit the use of one or more <expansion function invocation>s within a <token list>.

g) If *P* is a <context condition> *CCD*, let *n* be the number of <context argument>s immediately contained in *CCD*. For *i* ranging from 1 to *n*, let CA_i be these <context argument>s. Let *CCDC* be the canonical form of *CCD*, which is obtained by replacing every CA_i as follows:

Case:

i) If Ca_i is a <text literal>, replace Ca_i by:

```
(Cai)
```

ii) If Ca_i is a <Broader_Term function invocation>, let *SBT* be an *FT_BroaderTerm* instance such that Ca_i is equal to *StrctPattern_to_FT_Pattern*(*SBT*). Replace Ca_i by the result of:

```
StrctPattern_to_FT_Pattern(GetBroaderTerms(SBT>>thesaurus,
      SBT>>startingTerm, SBT>>expansionCnt))
```

iii) If Ca_i is a <Narrower_Term function invocation>, let *SBT* be an *FT_NarrowerTerm* instance such that Ca_i is equal to StrctPattern_to_FT_Pattern(*SBT*). Replace Ca_i by the result of:

```
StrctPattern_to_FT_Pattern(GetNarrowerTerms(SBT>>thesaurus,
      SBT>>startingTerm, SBT>>expansionCnt))
```

iv) If Ca_i is a <Synonym function invocation>, let *SBT* be an *FT_Synonym* instance such that Ca_i is equal to *StrctPattern_to_FT_Pattern*(*SBT*). Replace Ca_i by the result of:

```
StrctPattern_to_FT_Pattern(GetSynonymTerms(SBT>>thesaurus,
      SBT>>startingTerm))
```

v) If Ca_i is a <Preferred_Term function invocation>, let *SBT* be an *FT_PreferredTerm* instance such that Ca_i is equal to *StrctPattern_to_FT_Pattern*(*SBT*). Replace Ca_i by the result of:

```
StrctPattern_to_FT_Pattern(GetPreferredTerms(SBT>>thesaurus,
      SBT>>startingTerm))
```

vi) If Ca_i is a <Related_Term function invocation>, let *SBT* be an *FT_RelatedTerm* instance such that Ca_i is equal to *StrctPattern_to_FT_Pattern*(*SBT*). Replace Ca_i by the result of:

```
StrctPattern_to_FT_Pattern(GetRelatedTerms(SBT>>thesaurus,
      SBT>>startingTerm))
```

vii) If Ca_i is a <Top_Term function invocation>, let *SBT* be an *FT_TopTerm* instance such that Ca_i is equal to *StrctPattern_to_FT_Pattern*(*SBT*). Replace Ca_i by the result of:

```
StrctPattern_to_FT_Pattern(GetTopTerms(SBT>>thesaurus,
      SBT>>startingTerm))
```

viii) If Ca_i is a <Soundex function invocation>, let *SBT* be an *FT_Soundex* instance such that Ca_i is equal to *StrctPattern_to_FT_Pattern*(*SBT*). Replace Ca_i by the result of:

```
StrctPattern_to_FT_Pattern(GetSoundsSimilar(SBT>>spoken))
```

ix) Otherwise, Ca_i is left unchanged.

Let *SCR* be an *FT_Context* instance such that

```
CCDC = StrctPattern_to_FT_Pattern(SCR)
```

Then the result of

```
Contains(T, CCD)
```

is the result of

```
Contains(T, SCR)
```

h) If *P* is of the form <left paren> <search expression> <right paren>, let *SPSE* be an *FT_ParExpr* instance such that

```
P = StrctPattern_to_FT_Pattern(SPSE)
```

Then the result of

```
Contains(T, P)
```

is the result of

```
Contains(T, SPSE)
```

i) If *P* is a <Soundex_Exp function invocation> *SFI*, let *SSO* be an *FT_Soundex* instance such that

```
SFI = StrctPattern_to_FT_Pattern(SSO)
```

then the result of

```
Contains(T, SFI)
```

is the result of

```
Contains(T, SSO)
```

j) If *P* is a <Broader_Term function invocation> *BFI*, let *SBT* be an *FT_BroaderTerm* instance such that

```
BFI = StrctPattern_to_FT_Pattern(SBT)
```

then the result of

```
Contains(T, BFI)
```

is the result of

```
Contains(T, SBT)
```

k) If *P* is a <Narrower_Term function invocation> *NFI*, let *SNT* be an *FT_NarrowerTerm* instance such that

```
NFI = StrctPattern_to_FT_Pattern(SNT)
```

then the result of

```
Contains(T, NFI)
```

is the result of

```
Contains(T, SNT)
```

l) If *P* is a <Synonym function invocation> *SYFI*, let *SST* be an *FT_Synonym* instance such that

```
SYFI = StrctPattern_to_FT_Pattern(SST)
```

then the result of

```
Contains(T, SYFI)
```

is the result of

```
Contains(T, SST)
```

m) If *P* is a <Related_Term function invocation> *RTFI*, let *SRT* be an *FT_RelatedTerm* instance such that

```
RTFI = StrctPattern_to_FT_Pattern(SRT)
```

then the result of

```
Contains(T, RTFI)
```

is the result of

```
Contains(T, SRT)
```

n) If *P* is a <Preferred_Term function invocation> *PTFI*, let *SPT* be an *FT_PreferredTerm* instance such that

```
PTFI = StrctPattern_to_FT_Pattern(SPT)
```

then the result of

```
Contains(T, PTFI)
```

is the result of

```
Contains(T, SPT)
```

o) If *P* is a <Top_Term function invocation> *TTFI*, let *STT* be an *FT_TopTerm* instance such that

```
TTFI = StrctPattern_to_FT_Pattern(STT)
```

then the result of

```
Contains(T, TTFI)
```

is the result of

```
Contains(T, STT)
```

p) If *P* is a <Is_About function invocation> *IAFI*, let *SIA* be an *FT_IsAbout* instance such that

```
IAFI = StrctPattern_to_FT_Pattern(SIA)
```

then the result of

```
Contains(T, IAFI)
```

is the result of

```
Contains(T, SIA)
```

6 Structured Search Pattern Data Types

The types in this family provide for the construction of structured search patterns. The types form the following hierarchy:

- FT_Any
- FT_Primary
  - FT_TextLiteral
  - FT_Phrase
  - FT_Proxi
  - FT_Soundex
  - FT_BroaderTerm
  - FT_NarrowerTerm
  - FT_Synonym
  - FT_PreferredTerm
  - FT_RelatedTerm
  - FT_TopTerm
  - FT_IsAbout
  - FT_Context
  - FT_ParExpr
- FT_Term
- FT_Expr

6.1 FT_Any Type and Routines

6.1.1 FT_Any Type

Purpose

The *FT_Any* type provides facilities for the construction of a structured search pattern that represents a multiset the element type which is *FullText_Token*, and for testing whether at least one of members of such a multiset occurs in a given instance of type *FullText*.

Definition

```
CREATE TYPE FT_Any
   (Tokens FullText_Token ARRAY[FT_MaxArrayLength])
```

Definitional Rules

1) *FT_MaxArrayLength* is the implementation-dependent maximum length for an array.

Description

1) The *FT_Any* type provides for public use:

   a) an attribute *Tokens*,
   b) a function *FT_Any(FT_TextLiteral ARRAY)*,
   c) a function *Contains(FullText, FT_Any)*.

6.1.2 FT_Any Function

Purpose

Construct and initialize an *FT_Any* instance.

Definition

```
CREATE FUNCTION FT_Any
   (tokens FT_TextLiteral ARRAY[FT_MaxArrayLength])
   RETURNS FT_Any
   BEGIN
      DECLARE temp FT_Any;
      SET temp = FT_Any();
      SET temp>>Tokens = tokens;
      RETURN temp;
   END
```

Definitional Rules

1) *FT_MaxArrayLength* is the implementation-dependent maximum length for an array.

Description

1) The function *FT_Any(FT_TextLiteral ARRAY)* takes the following input parameters:

   a) an array *tokens* with elements of type *FT_TextLiteral* which represents a set of words or terms.

6.1.3 Contains Function

Purpose

Search a *FullText* instance for an *FT_Any*.

Definition

```
CREATE FUNCTION Contains
   (text FullText,
    any FT_Any)
   RETURNS BOOLEAN
   BEGIN
      DECLARE result BOOLEAN;
      DECLARE lent INTEGER;
      DECLARE lena INTEGER;
      DECLARE TokArray FullText_Token ARRAY[FT_MaxArrayLength];

      SET TokArray = Tokenize(text);

      IF TokArray IS NULL THEN
         SET lent = CAST(NULL AS INTEGER);
      ELSE
         SET lent = CARDINALITY(TokArray);
      END IF;
      IF any IS NULL THEN
         SET lena = CAST(NULL AS INTEGER);
      ELSEIF any>>Tokens IS NULL THEN
         SET lena = CAST(NULL AS INTEGER);
      ELSE SET lena = CARDINALITY(any>>Tokens);
      END IF;

      IF lent IS NULL AND lena IS NULL THEN
         RETURN UNKNOWN;
      ELSEIF lent = 0 OR lena = 0 THEN
         SET result = FALSE;
      ELSEIF lent <> 0 AND lena IS NULL OR
            lent IS NULL AND lena <> 0 THEN
         RETURN UNKNOWN;
      ELSE SET result =
         (WITH RECURSIVE Tab1(ind, token) AS
            (VALUES(1, TokArray[1])
               UNION
             SELECT ind + 1, TokArray[ind + 1]
             FROM   Tab1
             WHERE  ind < lent
            ),
             Tab2(ind, lit) AS
            (VALUES(1, any>>Tokens[1])
               UNION
             SELECT ind + 1, any>>Tokens[ind + 1]
             FROM   Tab2
             WHERE  ind < lena
            )
         VALUES
            (FOR SOME Tab1 tt
               (FOR SOME Tab2 ta
                  (matches(tt.token, ta.lit))
               )
```

```
                )
             );
          END IF;
          RETURN result;
       END
```

Definitional Rules

1) *FT_MaxArrayLength* is the implementation-dependent maximum length for an array.

Description

1) The function *Contains(FullText, FT_Any)* takes the following input parameters:

   a) a *FullText* item *text*,
   b) an *FT_Any* item *any*.

2) *Contains(FullText, FT_Any)* returns:

   Case:

   a) false, if either *Tokenize(text)* or *any>>Tokens* is empty, or if for every element *A* and *B* of *Tokenize(text)* and *any>>Tokens*, respectively,

      ```
      matches(A, B)
      ```

      is false.

   b) true, if there exist two elements *A* and *B* of *Tokenize(text)* and *any>>Tokens*, respectively, such that

      ```
      matches(A, B)
      ```

      is true;

   c) Otherwise, unknown.

3) In particular, this result is obtained if:

   a) Any of *text* or *Tokenize(text)* is the null value, and *any* or *any>>Tokens* is the null value.

   b) *text* or *Tokenize(text)* is the null value, but *any>>Tokens* is an non-empty array.

   c) *any* or *any>>Tokens* is the null value, but *Tokenize(text)* is an non-empty array.

6.2 FT_Primary Type and Routines

6.2.1 FT_Primary Type

Purpose

The *FT_Primary* type is the root type of a number elementary search pattern types. It provides a facility for negating any search pattern the type of which is a subtype of *FT_Primary*.

Definition

```
CREATE TYPE FT_Primary NOT INSTANTIABLE
   (NOT_tag BOOLEAN)
```

Description

1) The *FT_Primary* type provides for public use:

   a) an attribute *NOT_tag*,
   b) a function *NOTT(FT_Primar*y).

2) Instances of *FT_Primary* cannot be created. Only instances of subtypes of *FT_Primary* can be created..

6.2.2 NOTT Function

Purpose

Mark a structured search pattern as a negated search pattern.

Definition

```
CREATE FUNCTION NOTT
   (prim FT_Primary)
   RETURNS FT_Primary
   BEGIN
      SET prim>>NOT_tag = NOT prim>>NOT_tag;
      RETURN prim;
   END
```

Description

1) The function *NOTT(FT_Primary)* takes the following input parameters:

   a) a *FT_Primary* value *prim*.

Editor's Note 2-055
Is the *NOTT* function provided by the *FT_Primary* type needed since it does not appear to be used in the Full-Text specification?

6.2.3 StrctPattern_to_FT_Pattern Function

Purpose

Convert an *FT_Primary* value to an *FT_Pattern* value.

Definition

```
CREATE FUNCTION StrctPattern_to_FT_Pattern
   (prim FT_Primary)
   RETURNS FT_Pattern
   BEGIN
      DECLARE result CHARACTER VARYING(FT_MaxPatternLength);
      SET result = '" "'; -- dummy result
      RETURN CAST(result AS FT_Pattern);
   END
```

Definitional Rules

1) *FT_MaxPatternLength* is the implementation-dependent maximum length for the character representation of an instance of *FT_Pattern*.

Description

1) The function *StrctPattern_to_FT_Pattern(FT_Primary)* takes the following input parameters:

   a) a *FT_Primary* value *prim*.

2) The function *StrctPattern_to_FT_Pattern(FT_Primary)* is a dummy function that will never be called since there are no *FT_Primary* instances which are not instances of a subtype of *FT_Primary*.

6.3 FT_TextLiteral Type and Routines

6.3.1 FT_TextLiteral Type

Purpose

The *FT_TextLiteral* type provides facilities for the construction of literal search patterns, and for searching of occurrences of literals in text.

Definition

```
CREATE TYPE FT_TextLiteral
   UNDER FT_Primary
   (LitPart FullText_Token,
    EscapeSpec CHARACTER(1))
```

Description

1) The *FT_TextLiteral* type provides for public use:

   a) an attribute *LitPart*,
   b) an attribute *EscapeSpec*,
   c) a function *FT_TextLiteral(FullText_Token)* and a function *FT_TextLiteral(FullText_Token, CHARACTER)*,
   d) a function *Contains(FullText, FT_TextLiteral)*,
   e) a function *StrctPattern_to_FT_Pattern(FT_TextLiteral)*.

6.3.2 FT_TextLiteral Functions

Purpose

Construct and initialize an *FT_TextLiteral* instance.

Definition

```
CREATE FUNCTION FT_TextLiteral
   (w FullText_Token)
   RETURNS FT_TextLiteral
   BEGIN
      DECLARE temp FT_TextLiteral;
      SET temp = FT_TextLiteral();
      SET temp>>LitPart = EliminateDQS(w);
      SET temp>>NotTag = TRUE;
      RETURN temp;
   END

CREATE FUNCTION FT_TextLiteral
   (w FullText_Token,
    EscapeChar CHARACTER(1))
   RETURNS FT_TextLiteral
   BEGIN
      DECLARE temp FT_TextLiteral;

      SET temp = FT_TextLiteral(w);
      SET temp>>EscapeSpec = EscapeChar;
      RETURN temp;
   END
```

Description

1) The function *FT_TextLiteral(FullText_Token)* takes the following input parameters:

   a) a *FullText_Token* value *w*.

2) The function *FT_TextLiteral(FullText_Token, CHARACTER)* takes the following input parameters:

   a) a *FullText_Token* value *w*,
   b) a *CHARACTER* value *EscapeChar*.

3) In the process of constructing an *FT_TextLiteral*, the appearance of <doublequote symbol>s in the token *w* is taken care of by the function *EliminateDQS(FullText_Token)*. *EliminateDQS(FullText_Token*) replaces each <doublequote symbol> in a token by a <double quote>.

6.3.3 Contains Function

Purpose

Search a *FullText* instance for an *FT_TextLiteral*.

Definition

```
CREATE FUNCTION Contains
   (text FullText,
    lit FT_TextLiteral)
   RETURNS BOOLEAN
   BEGIN
      DECLARE result BOOLEAN;

      IF Tokenize(text) IS NULL THEN
         RETURN UNKNOWN;
      END IF;
      IF CARDINALITY(Tokenize(text)) = 0 THEN
         SET result = FALSE;
      ELSE
         SET result = (WITH RECURSIVE tempTab(pos, token) AS
            (VALUES(1, Tokenize(text)[1])
                  UNION
             SELECT tt.pos + 1, Tokenize(text)[tt.pos + 1]
             FROM   tempTab tt
             WHERE  tt.pos < CARDINALITY(Tokenize(text))
            )
             VALUES(FOR SOME tempTab tt
               (matches(tt.token, lit))
         );
      END IF;
      RETURN (lit>>NOT_tag = result);
   END
```

Description

1) The function *Contains(FullText, FT_TextLiteral)* takes the following input parameters:

   a) a *FullText* item *text*,
   b) an *FT_TextLiteral* item *lit*.

2) Let *TL* be the result of the invocation of *Tokenize(text)*, and *T* be *lit>>LitPart*, with leading and trailing blanks removed; if *lit>>EscapeSpec* is the null value, let *TT* be *T*; otherwise, let *TT* be *T* ESCAPE *lit>>EscapeSpec*.

   a) Case:

      i) If *TL* is empty, then let *R* be false.

      ii) If

```
TLE NOT LIKE TT
```

is true for every element *TLE* of *TL*, with leading and trailing blanks removed from *TLE*, then let *R* be false.

iii) If *TL* contains at least one element *TLE*, with leading and trailing blanks removed, such that

```
TLE LIKE TT
```

is true, then let *R* be true.

iv) Otherwise, let *R* be unknown.

b) The function *Contains(FullText, FT_TextLiteral)* returns:

i) unknown, if *lit>>NOT_tag* is the null value,

ii) *R*, if *lit>>NOT_tag* is true,

iii) Otherwise, NOT *R*.

6.3.4 StrctPattern_to_FT_Pattern Function

Purpose

Convert an *FT_TextLiteral* value to an *FT_Pattern* value.

Definition

```
CREATE FUNCTION StrctPattern_to_FT_Pattern
   (lit FT_TextLiteral)
   RETURNS FT_Pattern
   BEGIN
      DECLARE result CHARACTER VARYING(FT_MaxPatternLength);

      SET result = '"' || TRIM(BOTH ' ' FROM InsertDQS(lit>>LitPart))
         || CASE WHEN lit>>EscapeSpec IS NULL THEN
                '"'
            ELSE
                '" ESCAPE "' || lit>>EscapeSpec || '"'
            END;
      IF lit>>NOT_TAG IS UNKNOWN THEN
         SET result = NULL;
      ELSEIF NOT lit>>NOT_tag THEN
         SET result = 'NOT ' || result;
      END IF;
      RETURN CAST(result AS FT_Pattern);
   END
```

Definitional Rules

1) *FT_MaxPatternLength* is the implementation-dependent maximum length for the character representation of an instance of *FT_Pattern*.

Description

1) The function *StrctPattern_to_FT_Pattern(FT_TextLiteral)* takes the following input parameters:

   a) a *FT_TextLiteral* value *text*.

2) *StrctPattern_to_FT_Pattern(FT_TextLiteral)* converts an instance of type *FT_TextLiteral* into an *FT_Pattern* of the form <word> or of the form NOT <word>.

3) In the course of constructing the *FT_Pattern*, <double quote>s appearing in the *LitPart* attribute of the *FT_TextLiteral text* are taken care of by the function *InsertDQS(FullText_Token)*. *InsertDQS(FullText_Token)* replaces each <double quote> in a token by a <doublequote symbol>.

4) If the input argument *lit* is the null value, or if *lit>>NOT_tag* is unknown, then the result is the null value.

6.3.5 EliminateDQS Function

Purpose

Eliminate a double quote symbol from a *FullText_Token*.

Definition

```
CREATE FUNCTION EliminateDQS
   (w FullText_Token)
   RETURNS FullText_Token
   BEGIN
      --
      -- !! See Description
      --
   END
```

Description

1) The function *EliminateDQS(FullText_Token)* takes the following input parameters:

   a) a *FullText_Token* value *w*.

2) *EliminateDQS(FullText_Token)* replaces each <doublequote symbol> in *w* by a <double quote>.

6.3.6 InsertDQS Function

Purpose

Insert a double quote symbol in a *FullText_Token*.

Definition

```
CREATE FUNCTION InsertDQS
   (w FullText_Token)
   RETURNS CHARACTER VARYING(FT_MaxPatternLength)
   BEGIN
      --
      -- !! See Description
      --
   END
```

Definitional Rules

1) *FT_MaxPatternLength* is the implementation-dependent maximum length for the character representation of an instance of *FT_Pattern*.

Description

1) The function *InsertDQS(FullText_Token)* takes the following input parameters:

   a) a *FullText_Token* value *w*.

2) *InsertDQS(FullText_Token)* replaces each <double quote> in a token by a <doublequote symbol>.

6.3.7 matches Function

Purpose

Compare a *FullText_Token* value with an *FT_TextLiteral* value.

Definition

```
CREATE FUNCTION matches
   (tok FullText_Token,
    tlit FT_TextLiteral)
   RETURNS BOOLEAN
   BEGIN
      RETURN (CASE WHEN tlit>>EscapeSpec IS NULL THEN
         TRIM(BOTH ' ' FROM tok) LIKE
            TRIM(BOTH ' ' FROM tlit>>LitPart)
      ELSE
         TRIM(BOTH ' ' FROM tok) LIKE
            TRIM(BOTH ' ' FROM tlit>>LitPart) ESCAPE tlit>>EscapeSpec
      END
      );
   END
```

Description

1) The private function *matches(FullText_Token, FT_TextLiteral)* takes the following input parameters:

   a) a *FullText_Token* item *tok*,
   b) an *FT_TextLiteral* item *tlit*.

2) *matches(FullText_Token, FT_TextLiteral)* compares *tok* and *tlit* using the LIKE operator to return a BOOLEAN value.

6.4 FT_Phrase Type and Routines

6.4.1 FT_Phrase Type

Purpose

The *FT_Phrase* type provides for the construction of phrase search patterns, and for searching of occurrences of the phrases in text.

Definition

```
CREATE TYPE FT_Phrase
   UNDER FT_Primary
   (PhrasePart FullText_Token ARRAY[FT_MaxArrayLength]
    EscapeSpec CHARACTER(1))
```

Definitional Rules

1) *FT_MaxArrayLength* is the implementation-dependent maximum length for an array.

Description

1) The *FT_Phrase* type provides for public use:

   a) a function *FT_Phrase(FullText_Token ARRAY)* and a function *FT_Phrase(FullText_Token ARRAY, CHARACTER)*,
   b) a function *Contains(FullText, FT_Phrase)*,
   c) a function *StrctPattern_to_FT_Pattern(FT_Phrase)*.

2) An *FT_Phrase* instance denotes an array of *FullText_Token* tokens which in turn represents a sequence of words. The array may be either empty or the null value.

   Tokens may contain wild card characters '%' and '_'. The '%' wildcard denotes an arbitrary number (zero or more) of characters which are admissible within a token. An '_' wildcard denotes one arbitrary character out of the set of characters which are admissible within a token.

   A token may be the null value.

   Note: *FT_Phrase* instances are intentionally more general than <phrase>s which contain at least two <word representation>s, none of which may be a NULL string.

3) If a token exclusively consists of '%' wildcards, it denotes an optional word.

6.4.2 FT_Phrase Functions

Purpose

Construct and initialize an *FT_Phrase* instance.

Definition

```
CREATE FUNCTION FT_Phrase
   (wl FullText_Token ARRAY[FT_MaxArrayLength])
   RETURNS FT_Phrase
   BEGIN
      DECLARE temp FT_Phrase;
      DECLARE i INTEGER;

      SET temp = FT_Phrase();
      IF wl IS NULL THEN
         RETURN temp;
      END IF;
      SET temp>>NOT_tag = TRUE;
      SET temp>>PhrasePart =
         CAST(EMPTY AS FullText_Token ARRAY[FT_MaxArrayLength]);
      -- This function expects a list of FullText tokens
      -- where <doublequote symbol>s have not been
      -- eliminated yet.  Therefore, tokens in wl may contain
      -- <doublequote symbol>s that have to be turned into
      -- <double quote>s
      SET i = 0;
      L1: WHILE (i < CARDINALITY(wl)) DO
         SET temp>>PhrasePart = CONCATENATE(temp>>PhrasePart,
            ARRAY[EliminateDQS(wl[i + 1])]);
         SET i = i + 1;
      END WHILE L1;
      RETURN temp;
   END

CREATE FUNCTION FT_Phrase
   (w FullText_Token,
    EscapeChar CHARACTER(1))
   RETURNS FT_Phrase
   BEGIN
      DECLARE temp FT_Phrase;

      SET temp = FT_Phrase(w);
      SET temp>>EscapeSpec = EscapeChar;
      RETURN temp;
   END
```

Definitional Rules

1) *FT_MaxArrayLength* is the implementation-dependent maximum length for an array.

Description

1) The function *FT_Phrase(FullText_Token ARRAY)* takes the following input parameters:

   a) an array *wl* of *FullText_Token*s, representing a sequence of words.

2) The function *FT_Phrase(FullText_Token ARRAY, CHARACTER)* takes the following input parameters:

   a) an array *wl* of *FullText_Tokens*, representing a sequence of words,
   b) a *CHARACTER(1)* value *EscapeChar*.

6.4.3 Contains Function

Purpose

Search a *FullText* instance for an *FT_Phrase.*

Definition

```
CREATE FUNCTION Contains
   (text FullText,
    phr FT_Phrase)
   RETURNS BOOLEAN
   BEGIN
      DECLARE tokarray FullText_Token ARRAY[FT_MaxArrayLength];
      DECLARE result BOOLEAN;
      DECLARE lent INTEGER;
      DECLARE lenp INTEGER;
      DECLARE nmsk INTEGER;
      DECLARE i    INTEGER;

      SET tokarray = Tokenize(text);
      IF tokarray IS NULL THEN
         RETURN UNKNOWN;
      END IF;
      SET lent = CARDINALITY(tokarray);
      IF (phr IS NULL OR phr>>PhrasePart IS NULL) AND
            lent <> 0 THEN
         RETURN UNKNOWN;
      END IF;
      SET lenp = CARDINALITY(phr>>PhrasePart);
      SET nmsk = 0;
      SET i    = 1;
      --------------------------------------------
      - find tokens representing an optional word
      --------------------------------------------
      L1: WHILE (i <= lenp) DO
         IF phr>>PhrasePart[i] SIMILAR '$%+' ESCAPE '$' THEN
            SET nmsk = nmsk + 1;
         END IF;
         SET i = i + 1;
      END WHILE L1;
      IF lent = 0 OR lent < lenp - nmsk THEN
         SET result = FALSE;
      ELSEIF lenp = 0 OR lenp - nmsk = 0 THEN
         SET result = TRUE;
      ELSE
         SET result = WITH RECURSIVE temptoks(pos, token) AS
            (VALUES (1, tokarray[1])
               UNION
             SELECT tt.pos + 1, tokarray[tt.pos + 1]
             FROM   temptoks tt
             WHERE  tt.pos < lent
            )
            VALUES (FOR SOME temptoks tt
               (tt.pos >= 1 AND tt.pos <= lent + 1 - (lenp - nmsk)
                  AND
               matches(tokarray, tt.pos, lent, phr>>PhrasePart, 1,
                     lenp, phr>>EscapeSpec)
               )
            );
      END IF;
      RETURN (lit>>NOT_tag = result);
   END
```

Definitional Rules

1) *FT_MaxArrayLength* is the implementation-dependent maximum length for an array.

Description

1) The function *Contains(FullText, FT_Phrase)* takes the following input parameters:

   a) a *FullText* argument *text*,
   b) an *FT_Phrase* value *phr*.

2) Case:

   a) If *text* denotes an empty array of tokens, or if the number of tokens is less than the number of tokens in *phr>>PhrasePart,* not counting the tokens denoting optional words, then let *R* be false.

   b) If either *phr*, *phr>>PhrasePart* or *text* is the null value, then let *R* be unknown.

   c) If the number of tokens in *phr>>PhrasePart* is zero, not counting the tokens denoting optional words, then let *R* be true.

   d) Otherwise:

      i) Let *n* be the number of tokens in *phr>>PhrasePart*. Let *now* be the number of optional words in *phr>>PhrasePart*. Let *STS* be a set of *m* sequences of tokens, where *m* is 2 to the power of *n*, such that:

         A) *phr>>PhrasePart* is an element of *STS*.

         B) Every other element of *STS* (if *m* > 1) is obtained from *phr>>PhrasePart* by removing one of the possible combinations of optional words.

         C) No two elements of *STS* are equal.

      ii) Let *S1* be a sequence of *L* tokens contained in text, and *S2* an element of *STS* of the same length *L*. For *j* ranging from 1 to *L*, let $S1_j$ and $S2_j$ be elements of *S1* and *S2*, respectively. If *phr>>EscapeSpec* is the null value, let *TT* be $S2_j$; otherwise, let *TT* be $S2_j$ ESCAPE *lit>>EscapeSpec*.

      iii) Case:

         A) If there exists some *S1* and some *S2* such that

            $S1_j$ LIKE TT

            is true for every *j*, then let *R* be true.

B) If for every possible pair (*S1*, *S2*)

`S1`$_j$ `LIKE TT`

is false for at least one *j*, then let *R* be false.

3) The function *Contains(FullText, FT_Phrase)* returns:

Case:

A) unknown, if *NOT_tag* is the null value.

B) NOT *R*, if *NOT_tag* is false.

C) Otherwise, *R*.

6.4.4 StrctPattern_to_FT_Pattern Function

Purpose

Convert an *FT_Phrase* value to an *FT_Pattern* value.

Definition

```
CREATE FUNCTION StrctPattern_to_FT_Pattern
   (phrP FT_Phrase)
   RETURNS FT_Pattern
   BEGIN
      DECLARE result CHARACTER VARYING(FT_MaxPatternLength);
      DECLARE len INTEGER;
      DECLARE i INTEGER;

      IF phrP IS NULL THEN
         RETURN CAST(NULL AS FT_Pattern);
      ELSEIF phrP>>PhrasePart IS NULL THEN
         RETURN CAST(NULL AS FT_Pattern);
      END IF;

      SET i = 1;
      SET len = CARDINALITY(phrP>>PhrasePart);
      SET result = ‘"‘;
      WHILE (i <= len) DO
         SET result = result
            || InsertDQS(phrP>>PhrasePart[i])
            || ‘ ‘;
         SET i = i + 1;
      END WHILE;

      SET RESULT = TRIM(TRAILING ‘ ‘ FROM result)
         || CASE WHEN phrP>>EscapeSpec IS NULL THEN
               ‘"‘
            ELSE
               ‘" ESCAPE "‘ || phrP>>EscapeSpec || ‘"‘;

      IF NOT phrP>>NOT_TAG IS NULL THEN
         SET result = NULL;
      ELSIF NOT phrP>>NOT_tag THEN
         SET result = 'NOT ' || result;
      END IF;
      RETURN CAST(result AS FT_Pattern);
   END
```

Definitional Rules

1) *FT_MaxPatternLength* is the implementation-dependent maximum length for the character representation of an instance of *FT_Pattern*.

Description

1) The function *StrctPattern_to_FT_Pattern(FT_Phrase)* takes the following input parameters:

   a) an *FT_Phrase* value *phrP*.

2) *StrctPattern_to_FT_Pattern(FT_Phrase)* returns an *FT_Pattern* of the form <phrase> or the form NOT <phrase>.

3) If the input argument *phrP* or *phrP>>PhrasePart* is the null value, or if *phrP>>NOT_tag* is <u>unknown</u>, then the result is the null value.

6.4.5 matches Function

Purpose

Compare two *FullText_Token* array values.

Definition

```
   CREATE FUNCTION matches
      (tokarray FullText_Token ARRAY[FT_MaxArrayLength],
       post     INTEGER,
       lent     INTEGER,
       phrP     FullText_Token ARRAY[FT_MaxArrayLength],
       posp     INTEGER,
       lenp     INTEGER
       EscapeChar CHARACTER(1))
      RETURNS BOOLEAN
      BEGIN
         RETURN
         CASE
            -- pattern exhausted, match found
            WHEN (posp > lenp) THEN
               TRUE
            -- text to be tested exhausted, no match found
            WHEN (post > lent) THEN
               FALSE
            ELSE -- test successful so far; continue
               CASE
                  WHEN phrP[posp] NOT SIMILAR '$%+' ESCAPE '$' THEN
                     matches(tokarray[post],
                        FT_TextLiteral(phrP[posp]), EscapeChar)
                     AND
                     matches(tokarray, post+1, lent, phrP, posp+1,
                        lenp, EscapeChar)
                  ELSE matches(tokarray, post, lent, phrP, posp+1,
                        lenp, EscapeChar)
                     OR
                     matches(tokarray, post+1, lent, phrP, posp+1,
                        lenp, EscapeChar)
               END
            END
      END
```

Definitional Rules

1) *FT_MaxArrayLength* is the implementation-dependent maximum length for an array.

Description

1) The private function *matches(FullText_Token ARRAY, INTEGER, INTEGER, FullText_Token ARRAY, INTEGER, INTEGER, CHARACTER)* takes the following input parameters:

   a) an array *tokarray* of *FullText_Token*s, representing a sequence of words.
   b) an INTEGER value *post*,
   c) an INTEGER value *lent*,
   d) an array *PhrP* of *FullText_Token*s, representing a sequence of phrases.
   e) an INTEGER value *post*,
   f) an INTEGER value *lent*,
   g) a CHARACTER value *EscapeChar*.

6.5 FT_Proxi Type and Routines

6.5.1 FT_Proxi Type

Purpose

FT_Proxi instances represent proximity search patterns.

Definition

```
CREATE TYPE FT_Proxi
   UNDER FT_Primary
   (TL1 FullText_Token ARRAY[FT_MaxArrayLength],
    TL2 FullText_Token ARRAY[FT_MaxArrayLength],
    dv  INTEGER,        -- distance value
    du  FullText_Token, -- distance unit
    oi  FullText_Token) -- order indicator
```

Definitional Rules

1) *FT_MaxArrayLength* is the implementation-dependent maximum length for an array.

Description

1) The *FT_Proxi* type provides for public use

   a) a function *FT_Proxi(FullText_Token ARRAY, FullText_Token ARRAY, INTEGER, FullText_Token, FullText_Token),*
   b) a function *Contains(FullText, FT_Proxi),*
   c) a function *StrctPattern_to_FT_Pattern(FT_Proxi).*

6.5.2 FT_Proxi Function

Purpose

Construct and initialize an *FT_Proxi* instance.

Definition

```
CREATE FUNCTION FT_Proxi
   (TokList1 FullText_Token ARRAY[FT_MaxArrayLength],
    TokList2 FullText_Token ARRAY[FT_MaxArrayLength],
    DistanceValue INTEGER,
    DistanceUnit FullText_Token,
    OrderIndicator FullText_Token)
   RETURNS FT_Proxi
   BEGIN
      DECLARE temp FT_Proxi;
      SET temp = FT_Proxi();
      SET temp>>NotTag = TRUE;
      SET temp>>TL1 = TokList1;
      SET temp>>TL2 = TokList2;
      SET temp>>dv = DistanceValue;
      SET temp>>du = DistanceUnit;
      SET temp>>oi = OrderIndicator;
      RETURN temp;
   END
```

Definitional Rules

1) *FT_MaxArrayLength* is the implementation-dependent maximum length for an array.

Description

1) The function *FT_Proxi(FullText_Token ARRAY, FullText_Token ARRAY, INTEGER, FullText_Token, FullText_Token)* takes the following input parameters:

   a) an array *TokList1* of *FullText_Token* elements, which represents a set of words,
   b) an array *TokList2* of *FullText_Token* elements, which represents a set of words,
   c) an INTEGER value *DistanceValue*,
   d) a *FullText_Token* value *DistanceUnit*,
   e) a *FullText_Token* value *OrderIndicator*.

2) All arguments may be null values. *TokList1* and *TokList2* may be empty.

6.5.3 Contains Function

Purpose

Search a *FullText* instance for an *FT_Proxi.*

Definition

```
CREATE FUNCTION Contains
   (text FullText,
    prox FT_Proxi)
   RETURNS BOOLEAN
   BEGIN
      DECLARE result BOOLEAN;
      DECLARE TokText FT_TokenPosition ARRAY[FT_MaxArrayLength];
      DECLARE lent   INTEGER;
      DECLARE lentl1 INTEGER;
      DECLARE lentl2 INTEGER;

      IF prox>>du <> 'CHARACTERS' OR
         prox>>du <> 'WORDS' OR
         prox>>du <> 'SENTENCES' OR
         prox>>du <> 'PARAGRAPHS' THEN
         RETURN -- !! See Description ;
      END IF;

      IF prox>>oi <> 'ANY_ORDER' OR
         prox>>oi <> 'IN_ORDER' THEN
         RETURN -- !! See Description ;
      END IF;

      IF prox>>dv < 0 THEN
         RETURN -- !! See Description ;
      END IF;

      SET TokText = TokenizePosition(text, prox>>du);
      IF TokText IS NULL THEN
         SET lent = CAST(NULL AS INTEGER)
      ELSE
         SET lent = CARDINALITY(TokText);
      END IF;

      IF prox IS NULL OR prox>>TL1 IS NULL THEN
         SET lentl1 = CAST(NULL AS INTEGER)
      ELSE
         SET lentl1 = CARDINALITY(prox>>TL1);
      END IF;

      IF prox IS NULL OR prox>>TL2 IS NULL THEN
         SET lentl2 = CAST(NULL AS INTEGER)
      ELSE
         SET lentl2 = CARDINALITY(prox>>TL2);
      END IF;

      IF lent = 0 OR lentl1 = 0 OR lentl2 = 0 THEN
         SET result = FALSE;
      ELSEIF lent IS NULL OR lentl1 IS NULL OR lentl2 IS NULL THEN
```

```
            RETURN UNKNOWN;
         ELSE
            SET result =
            CASE
               WHEN (lent = 0 OR lentl1 = 0 OR lentl2 = 0) THEN FALSE
            ELSE
               WITH RECURSIVE
                  ttTab(ind, tp) AS
                     (VALUES(1, TokText[1])
                        UNION
                      SELECT ind + 1, TokText[ind + 1]
                      FROM   ttTab
                      WHERE  ind < lent
                     ),
                  tl1Tab(ind, tok) AS
                     (VALUES(1, prox>>TL1[1])
                        UNION
                      SELECT ind + 1, prox>>TL1[ind + 1]
                      FROM   tl1Tab
                      WHERE  ind < lentl1
                     ),
                  tl2Tab(ind, tok) AS
                     (VALUES(1, prox>>TL2[1])
                        UNION
                      SELECT ind + 1, prox>>TL2[ind + 1]
                      FROM   tl2Tab
                      WHERE  ind < lentl2
                     )
               VALUES(FOR SOME ttTab  tt1,tl1Tab l1,ttTab  tt2,tl2Tab l2
                  (matches(tt1.tp>>token, l1.tok) AND
                   matches(tt2.tp>>token, l2.tok) AND
                   tt2.tp>>position BETWEEN
                     tt1.tp>>position - (prox>>dv + tt2.tp->corrVal) *
                        (CASE prox>>oi
                        WHEN 'IN_ORDER' THEN
                           0
                        ELSE
                           1
                        END)
                     AND tt1.tp>>position + prox>>dv + tt1.tp>>corrVal)
                  )
               )
            END;
            END IF;
         RETURN (prox>>NOT_tag = result);
      END
```

Definitional Rules

1) *FT_MaxArrayLength* is the implementation-dependent maximum length for an array.

Description

1) The function *Contains(FullText, FT_Proxi)* takes the following input parameters:

   a) a *FullText* value *text*,
   b) an *FT_Proxi* value *prox*.

2) Case:

   a) The result of the function *Contains(FullText, FT_Proxi)* is implementation-defined if any of the following holds:

      i) The distance value *prox>>dv* is less than zero.

      ii) The distance unit *prox>>du* has a value other than than 'CHARACTERS', 'WORDS', 'SENTENCES', or 'PARAGRAPHS'.

      iii) The order indication *prox>>oi* has a value other than 'ANY_ORDER' or 'IN_ORDER'.

   b) Otherwise,

      Case:

      I) If *prox>>TL1*, *prox>>TL2* or the result of *TokenizePosition(text, prox>>du)* is empty, then let *R* be false.

      ii) If *prox*, *prox>>TL1*, *prox>>TL2* or the result of *TokenizePosition(text, prox>>du)* is the null value, then let *R* be unknown.

      iii) Otherwise, let *TPS1* be the result of *TokenizePosition(text, prox>>du)*; let *TPS2* be the set of all pairs (*tp1*, *tp2*) such that *tp1* and *tp2* are elements of *TPS1*, and

         Case:

         A) The order indication *prox>>oi* has the value IN_ORDER'and the difference

            ```
            tp2>>pos - tp1>>pos
            ```

            is not negative and not greater than the distance value prox>>dv.

         B) The order indication *prox>>oi* has the value 'ANY_ORDER'and the absolute value of the difference

            ```
            tp2>>pos - tp1>>pos
            ```

            is not greater than the distance value *prox>>dv*.

         Let *WPS* be the set of all pairs (*w1*, *w2*) such that every *w1* and every *w2* is an element of *prox>TL1* and *prox>>TL2*, respectively.

         Case:

         A) If there is at least one pair (*tp1*, *tp2*) and one pair (*w1*, *w2*) such that both

            ```
            matches(tp1>>token, w1)
            ```

            and

            ```
            matches(tp2>>token, w2)
            ```

            are true then let *R* be true.

B) If for all pairs (*tp1*, *tp2*) and (*w1*, *w2*) both

```
matches(tp1>>token, w1)
```

and

```
matches(tp2>>token, w2)
```

are <u>false</u> then let *R* be <u>false</u>.

C) Otherwise, let *R* be <u>unknown</u>.

Note: The function *matches* is described in Subclause 6.3.7, "matches Function."

3) The function *Contains(FullText, FT_Proxi)* returns:

A) <u>unknown</u>, if *prox>>NOT_tag* is the null value.

B) NOT *R*, if *prox>>NOT_tag* is <u>false</u>.

C) Otherwise, *R*.

6.5.4 StrctPattern_to_FT_Pattern Function

Purpose

Convert an *FT_Proxi* value to an *FT_Pattern* value.

Definition

```
CREATE FUNCTION StrctPattern_to_FT_Pattern
   (prx FT_Proxi)
   RETURNS FT_Pattern
   BEGIN
      DECLARE result CHARACTER VARYING(FT_MaxPatternLength);

      IF prx is NULL THEN
         RETURN CAST(NULL TO FT_Pattern);
      END IF;

      SET result = 'PROXIMITY('
         || CAST(StrctPattern_to_FT_Pattern(prx>>TL1)
               AS CHARACTER VARYING(FT_MaxPatternLength)) || ','
         || TRIM(BOTH ' ' FROM CAST(prx>>dv
               AS CHARACTER VARYING(FT_MaxPatternLength)) || ','
         || '"' || TRIM(BOTH ' ' FROM prx>>du) || '"' || ','
         || '"' || TRIM(BOTH ' ' FROM prx>>oi) || '"' || ','
         || CAST(StrctPattern_to_FT_Pattern(prx>>TL2)
               AS CHARACTER VARYING(FT_MaxPatternLength)) ||')';

      IF NOT prx>>NOT_tag IS UNKNOWN THEN
         SET result = NULL;
      ELSEIF NOT prx>>NOT_tag THEN
         result = 'NOT ' || result;
      END IF;
      RETURN CAST(result AS FT_Pattern);
   END
```

Definitional Rules

1) *FT_MaxPatternLength* is the implementation-dependent maximum length for the character representation of an instance of *FT_Pattern*.

Description

1) The function *StrctPattern_to_FT_Pattern(FT_Proxi)* takes the following input parameters:

   a) an *FT_Proxi* value *prx*.

2) *StrctPattern_to_FT_Pattern(FT_Proxi)* returns an *FT_Pattern* of the form <Proximity function invocation> or the form NOT <Proximity function invocation>.

3) If the input argument *prx* or any of the attributes *prx>>TL1*, *prx>>du*, *prx>>dv*, *prx>>oi* are the null value, or if *prx>>NOT_tag* is unknown, then the result is the null value.

6.6 FT_Soundex Type and Routines

6.6.1 FT_Soundex Type

Purpose

FT_Soundex instances represent a search token to be matched in text due to phonetic critieria.

Definition

```
CREATE TYPE FT_Soundex
   UNDER FT_Primary
   (spoken FT_TextLiteral)
```

Description

1) The *FT_Soundex* type provides for public use:

   a) a function *FT_Soundex(FT_TextLiteral)*,
   b) a function *Contains(FullText, FT_Soundex)*,
   c) a function *StrctPattern_to_FT_Pattern(FT_Soundex)*.

6.6.2 FT_Soundex Function

Purpose

Construct and initialize an *FT_Soundex* instance.

Definition

```
CREATE FUNCTION FT_Soundex
   (snd FT_TextLiteral)
   RETURNS FT_Soundex
   BEGIN
      DECLARE temp FT_Soundex;
      SET temp = FT_Soundex();
      SET temp>>spoken = snd;
      SET temp>>NotTag = TRUE;
   END
```

Description

1) The function *FT_Soundex(FT_TextLiteral)* takes the following input parameters:

   a) an *FT_TextLiteral* value *snd*.

2) Though not enforced by this standard, *snd* is intended to represent a sound pattern which is potentially equivalent to a number of tokens. The equivalence is language dependent and implementation-defined.

6.6.3 Contains Function

Purpose

Search a *FullText* instance for an *FT_Soundex*.

Definition

```
CREATE FUNCTION Contains
   (text FullText,
    snd FT_Soundex)
   RETURNS BOOLEAN
   BEGIN
      DECLARE result BOOLEAN;

      SET result = Contains(text,
            FT_Any(GetSoundsSimilar(snd>>spoken)));

      RETURN (snd>>NOT_tag = result);
   END
```

Description

1) The function *Contains(FullText, FT_Soundex)* takes the following input parameters:

   a) a *FullText* value *text*,
   b) an *FT_Soundex* value *snd*.

2) Let *R* be the result of

```
   Contains(text, FT_Any(GetSoundsSimilar(snd>>spoken)))
```

   Case:

   a) If *snd>>NOT_tag* is unknown, then the function *Contains(FullText, FT_Soundex)* returns unknown.

   b) If *snd>>NOT_tag* is false, then the function *Contains(FullText, FT_Soundex)* returns NOT *R*.

   c) Otherwise, the function *Contains(FullText, FT_Soundex)* returns *R*.

6.6.4 StrctPattern_to_FT_Pattern Function

Purpose

Convert an *FT_Soundex* value to an *FT_Pattern* value.

Definition

```
CREATE FUNCTION StrctPattern_to_FT_Pattern
   (snd FT_Soundex)
   RETURNS FT_Pattern
   BEGIN
      DECLARE result CHARACTER VARYING(FT_MaxPatternLength);


      IF snd IS NULL THEN
         RETURN CAST(NULL as FT_Pattern);
      ELSEIF snd>>NOT_tag IS UNKNOWN THEN
         RETURN CAST(NULL AS FT_Pattern);
      END IF;

      SET result = 'SOUNDEX_EXP('
            || CAST(StrctPattern_to_FT_Pattern(snd>>spoken)
               AS CHARACTER VARYING(FT_MaxPatternLength)
            || '")';

      IF NOT snd>>NOT_TAG THEN
         SET result = 'NOT ' || result;
      END IF;
      RETURN CAST(result AS FT_Pattern);
   END
```

Definitional Rules

1) *FT_MaxPatternLength* is the implementation-dependent maximum length for the character representation of an instance of *FT_Pattern*.

Description

1) The function *StrctPattern_to_FT_Pattern(FT_Soundex)* takes the following input parameters:

   a) an *FT_Soundex* value *snd*.

2) *StrctPattern_to_FT_Pattern(FT_Soundex)* returns an *FT_Pattern* of the form <Soundex_Exp function invocation> or the form NOT <Soundex_Exp function invocation>.

3) If the input argument *snd*, *snd>>spoken*, or *snd>>spoken>>LitPart* is the null value, or if the attribute *snd>>NOT_tag* is unknown, then the result is the null value.

6.6.5 GetSoundsSimilar Function

Purpose

Return x.

Definition

```
CREATE FUNCTION GetSoundsSimilar
   (spoken FT_TextLiteral)
   RETURNS FullText_Token ARRAY[FT_MaxArrayLength]
   BEGIN
      --
      -- !! See Description
      --
   END
```

Definitional Rules

1) *FT_MaxArrayLength* is the implementation-dependent maximum length for an array.

Description

1) The private function *GetSoundsSimilar(FT_TextLiteral)* takes the following input parameters:

   a) a *FT_TextLiteral* value *spoken*.

2) *GetSoundsSimilar(FT_TextLiteral)* permits the generation of an array of *FT_TextLiteral* items (representing a set of words) each of which has a different form though it has similar pronunciation as the input word. The input argument *spoken* is included in the generated array of tokens. As for the generation of this token array, while it depends on language, it is at least possible to realize it by a development rule with the dscription of a syllable level.

3) If the input parameter *spoken* or *spoken>>LitPart* is the null value, then the result of *GetSoundsSimilar(FT_TextLiteral)* is the null value. Further details of *GetSoundsSimilar(FT_TextLiteral)* are implementation-defined.

6.7 FT_BroaderTerm Type and Routines

6.7.1 FT_BroaderTerm Type

Purpose

FT_BroaderTerm instances represent one or more thesaurus hierarchies and a search token; the latter is to be matched in text with corresponding broader terms as indicated by the named thesaurus hierarchies.

Definition

```
CREATE TYPE FT_BroaderTerm
   UNDER FT_Primary
   (thesaurus     CHARACTER VARYING(FT_ThesNameLength),
    startingTerm  FT_TextLiteral,
    expansionCnt  INTEGER)
```

Definitional Rules

1) *FT_ThesNameLength* is the implementation-dependent maximum length for the character representation of a thesaurus name.

Description

1) The *FT_BroaderTerm* type provides for public use:

   a) a function *FT_BroaderTerm(CHARACTER VARYING, FT_TextLiteral, INTEGER)*,
   b) a function *Contains(FullText, FT_BroaderTerm)*,
   c) a function *StrctPattern_to_FT_Pattern*(*FT_BroaderTerm)*.

2) For the purpose of this type, a thesaurus is effectively a table with two columns, *NarrowerTerm* and *BroaderTerm*, respectively. For a given row, the values contained in the two columns represent terms, the second one being a broader term of the first one.

3) The number of available thesauri and their names are implementation-defined.

6.7.2 FT_BroaderTerm Function

Purpose

Constructs and initialize an *FT_BroaderTerm* instance.

Definition

```
CREATE FUNCTION FT_BroaderTerm
   (thes_name CHARACTER VARYING(FT_ThesNameLength),
    strt FT_TextLiteral,
    thes_exp_count INTEGER)
   RETURNS FT_BroaderTerm
   BEGIN
      DECLARE temp FT_BroaderTerm;
      SET temp = FT_BroaderTerm();
      SET temp>>thesaurus = thes_name;
      SET temp>>startingTerm = strt;
      SET temp>>expansionCnt = thes_exp_count;
      SET temp>>NotTag = TRUE;
      RETURN temp;
   END
```

Definitional Rules

1) *FT_ThesNameLength* is the implementation-dependent maximum length for the character representation of a thesaurus name.

Description

1) The function *FT_BroaderTerm(CHARACTER VARYING, FT_TextLiteral, INTEGER)* takes the following input parameters:

   a) a CHARACTER VARYING value *thes_name*,
   b) an *FT_TextLiteral* value *strt*,
   c) an INTEGER value *thes_exp_count*.

6.7.3 Contains Function

Purpose

Search a *FullText* instance for an *FT_BroaderTerm.*

Definition

```
CREATE FUNCTION Contains
   (text FullText,
    brdt FT_BroaderTerm)
   RETURNS BOOLEAN
   BEGIN
      DECLARE BrdArray FullText_Token ARRAY[FT_MaxArrayLength];
      DECLARE result BOOLEAN;

      SET BrdArray = GetBroaderTerms(brdt>>thesaurus ,
            brdt>>startingTerm,
            brdt>>expansionCnt);
      SET result = Contains(text, FT_Any(BrdArray));

      RETURN (brdt>>NOT_tag = result);
   END
```

Definitional Rules

1) *FT_MaxArrayLength* is the implementation-dependent maximum length for an array.

Description

1) The function *Contains(FullText, FT_BroaderTerm)* takes the following input parameters:

   a) a *FullText* value *text*,
   b) an *FT_BroaderTerm* value *brdt*.

2) Let *R* be the result of

```
Contains(text, FT_Any(GetBroaderTerms(brdt>>thesaurus,
   brdt>>startingTerm, brdt>>expansionCnt)))
```

   Case:

   a) If *brdt>>NOT_tag* is unknown, then *Contains(FullText, FT_BroaderTerm)* returns unknown.

   b) If *brdt>>NOT_tag* is false, then *Contains(FullText, FT_BroaderTerm)* returns NOT *R*.

   c) Otherwise, *Contains(FullText, FT_BroaderTerm)* returns *R*.

6.7.4 StrctPattern_to_FT_Pattern Function

Purpose

Convert an *FT_BroaderTerm* value to an *FT_Pattern* value.

Definition

```
CREATE FUNCTION StrctPattern_to_FT_Pattern
   (brdt FT_BroaderTerm)
   RETURNS FT_Pattern
   BEGIN
      DECLARE result CHARACTER VARYING(FT_MaxPatternLength);

      IF brdt IS NULL THEN
         RETURN CAST(NULL AS FT_Pattern);
      ELSEIF brdt>>NOT_tag IS UNKNOWN THEN
         RETURN CAST(NULL AS FT_Pattern);
      END IF;

      SET result = 'BROADER_TERM('
            || brdt>>thesaurus
            || ','
            || CAST(StrctPattern_to_FT_Pattern(brdt>>startingTerm)
                  AS CHARACTER VARYING(FT_MaxPatternLength))
            || ','
            || CASE WHEN brdt>>expansionCnt IS NULL THEN
                  ‘ALL’
               ELSE
                  TRIM(BOTH' ' FROM CAST(brdt>>expansionCnt
                     AS CHARACTER VARYING(FT_MaxPatternLength))
               END
            || ')';

      IF NOT brdt>>NOT_TAG THEN
         SET result = 'NOT ' || result;
      END IF;
      RETURN CAST(result AS FT_Pattern);
   END
```

Definitional Rules

1) *FT_MaxPatternLength* is the implementation-dependent maximum length for the character representation of an instance of *FT_Pattern*.

Description

1) The function *StrctPattern_to_FT_Pattern(FT_BroaderTerm)* takes the following input parameters:

   a) an *FT_BroaderTerm* value *brdt*.

2) *StrctPattern_to_FT_Pattern(FT_BroaderTerm)* returns an *FT_Pattern* of the form <Broader_Term function invocation> or NOT <Broader_Term function invocation>.

3) If the input argument *brdt* is the null value, or if *brdt>>NOT_tag* is unknown, then the result is the null value.

6.7.5 GetBroaderTerms Function

Purpose

Get broader terms from a thesaurus.

Definition

```
CREATE FUNCTION GetBroaderTerms
   (thes_name CHARACTER VARYING(FT_ThesNameLength),
    startingTerm FT_TextLiteral,
    thes_exp_count INTEGER)
   RETURNS FullText_Token ARRAY[FT_MaxArrayLength]
   BEGIN
      DECLARE ret FT_TextLiteral ARRAY[FT_MaxArrayLength];
      DECLARE strt FT_TextLiteral;
      DECLARE strt_termid INTEGER;
      DECLARE local_exp_countINTEGER;

      SET thes_name = TRIM(BOTH ' ' FROM thes_name);
      SET strt = TRIM(BOTH ' ' FROM startingTerm>>LitPart);

      SET local_exp_count =
         CASE
            WHEN thes_exp_count IS NOT NULL THEN
               thes_exp_count
            ELSE
               1
         END;

      SET strt_termid =
         (SELECT TERMID
          FROM TERM_DICTIONARY
          WHERE TRIM(BOTH ' ' FROM EXPR) = strt
                AND TRIM(BOTH ' ' FROM THNAME_DIC) = thes_name
         );

      SET ret=CAST(EMPTY AS FT_TextLiteral ARRAY[FT_MaxArrayLength]);

   L1: FOR elem AS
         WITH RECURSIVE done_so_far (TERMID,NARROWER_TERMID,LEVEL) AS
            (SELECT TERMID, NARROWER_TERMID, 0
             FROM TERM_HIERARCHY
             WHERE NARROWER_TERMID = strt_termid
               AND TRIM(BOTH ' ' FROM THNAME_HRR) = thes_name
               AND local_exp_count >= 0
                  UNION
             SELECT more.TERMID, more.NARROWER_TERMID,
                     CASE
                       WHEN thes_exp_count IS NOT NULL THEN
                          B.LEVEL + 1
                       ELSE
                          0
                     END AS LEVEL
             FROM done_so_far B, TERM_HIERARCHY more
             WHERE B.TERMID = more.NARROWER_TERMID
                AND TRIM(BOTH ' ' FROM more.THNAME_HRR) = thes_name
                AND B.LEVEL < local_exp_count
            )
         SELECT ARRAY[FT_TextLiteral(
               TRIM(BOTH ' ' FROM TD.EXPR)] AS EXRParr1
         FROM TERM_DICTIONARY TD, done_so_far f
         WHERE TD.TERMID = f.TERMID
               AND TRIM(BOTH ' ' FROM TD.THNAME_DIC) = thes_name

            DO  -- for every row of the above query result,
                -- append the value of column EXPRarr1 to the array

               SET ret = CONCATENATE(ret, EXPRarr1);
      END FOR L1;
   END
```

Definitional Rules

1) *FT_ThesNameLength* is the implementation-dependent maximum length for the character representation of a thesaurus name.

2) *FT_MaxArrayLength* is the implementation-dependent maximum length for an array.

Description

1) The private function *GetBroaderTerms(CHARACTER VARYING, FT_TextLiteral, INTEGER)* takes the following input parameters:

   a) a CHARACTER VARYING value *thes_name*, denoting a thesaurus *TH*,
   b) an *FT_TextLiteral* value *startingTerm*,
   c) an INTEGER value *thes_exp_count*.

2) *GetBroaderTerms(CHARACTER VARYING, FT_TextLiteral, INTEGER)* returns an array of *FT_TextLiteral* elements which each represent a broader term.

Editor's Note 2-044

The function GetBroaderTerms does NOT take into consideration the possibility that a value returned from the TERM_DICTIONARY might contain a delimiter which has to be handled by using the InsertDQS function.

Note that this problem also exists in the following subclauses:
a) 5.3.7 "FT_NarrowerTerm Abstract Data Type", function GetNarrowerTerms
b) 5.3.8 "FT_Synonym Abstract Data Type", function GetSynonymTerms
c) 5.3.9 "FT_PreferredTerm Abstract Data Type", function GetPreferredTerms
d) 5.3.10 "FT_RelatedTerm Abstract Data Type", function GetRelatedTerms
e) 5.3.11 "FT_TopTerm Abstract Data Type", function GetTopTerms

3) The result of *GetBroaderTerms(CHARACTER VARYING, FT_TextLiteral, INTEGER)* is empty if one of the following holds:

   a) The term *strt* is not contained in column *NarrowerTerm* of *TH*,
   b) either *strt* or *thes_name* is the null value,
   c) the expansion count *thes_exp_count* is smaller than zero.

4) If the expansion count *thes_exp_count* is zero, *GetBroaderTerms(CHARACTER VARYING, FT_TextLiteral, INTEGER)* returns all terms in column *BroaderTerm* of those rows of *TH* the values of which in column *NarrowerTerm* are equivalent to *strt*. If the expansion count *thes_exp_count* is *n* > 0, the resulting array represents the set:

   MS_1 UNION MS_2

   where MS_1 is the multiset represented by the result of

   ```
   GetBroaderTerms(thes_name, strt, thes_exp_count - 1)
   ```

   and MS_2 is given by

   $MS_{2,1}$ UNION ... $MS_{2,i}$ ... UNION $MS_{2,m}$,

   where *m* is the number of elements in MS_1, *i* ranges from 1 to *m*, E_i is some element of MS_1, and $MS_{2,i}$ is represented by

   GetBroaderTerms(thes_name, E_i, 0)

5) If the expansion count *thes_exp_count* is NULL, expansion is carried on until no new broader terms can be found.

6) The term *strt* is **not** included in the result.

6.8 FT_NarrowerTerm Type and Routines

6.8.1 FT_NarrowerTerm Type

Purpose

FT_NarrowerTerm instances represent one or more thesaurus hierarchies and a search token; the latter is to be matched in text with corresponding narrower terms as indicated by the named thesaurus hierarchies.

Definition

```
CREATE TYPE FT_NarrowerTerm
   UNDER FT_Primary
   (thesaurus    CHARACTER VARYING(FT_ThesNameLength),
    startingTerm FT_TextLiteral,
    expansionCnt INTEGER)
```

Definitional Rules

1) *FT_ThesNameLength* is the implementation-dependent maximum length for the character representation of a thesaurus name.

Description

1) The *FT_NarrowerTerm* type provides for public use:

   a) a function *FT_NarrowerTerm(CHARACTER VARYING, FT_TextLiteral, INTEGER)*,
   b) a function *Contains(FullText, FT_NarrowerTerm),*
   c) a function *StrctPattern_to_FT_Pattern(FT_NarrowerTerm).*

2) For the purpose of this data type, a thesaurus is effectively a table with two columns, *NarrowerTerm* and *BroaderTerm*. For a given row, the values contained in the two columns represent terms, the first being a narrower term of the second one.

3) The number of available thesauri and their names are implementation-defined.

6.8.2 FT_NarrowerTerm Function

Purpose

Construct and initialize an *FT_NarrowerTerm* instance.

Definition

```
CREATE FUNCTION FT_NarrowerTerm
   (thes_name CHARACTER VARYING(FT_ThesNameLength),
    strt FT_TextLiteral,
    thes_exp_count INTEGER)
   RETURNS FT_NarrowerTerm
   BEGIN
      DECLARE temp FT_NarrowerTerm;
      SET temp = FT_NarrowerTerm();
      SET temp>>thesaurus = thes_name;
      SET temp>>startingTerm = strt;
      SET temp>>expansionCnt = thes_exp_count;
      SET temp>>NotTag = TRUE;
      RETURN temp;
   END
```

Definitional Rules

1) *FT_ThesNameLength* is the implementation-dependent maximum length for the character representation of a thesaurus name.

Description

1) The function *FT_NarrowerTerm(CHARACTER VARYING, FT_TextLiteral, INTEGER))* takes the following input parameters:

   a) a CHARACTER VARYING value *thes_name*,
   b) an *FT_TextLiteral* value *strt*,
   c) an INTEGER value *thes_exp_count*.

6.8.3 Contains Function

Purpose

Search a *FullText* instance for an *FT_NarrowerTerm.*

Definition

```
CREATE FUNCTION Contains
   (text FullText,
    nrwt FT_NarrowerTerm)
   RETURNS BOOLEAN
   BEGIN
      DECLARE NrwArray FullText_Token ARRAY[FT_MaxArrayLength];
      DECLARE result BOOLEAN;

      SET NrwArray = GetNarrowerTerms(nrwt>>thesaurus ,
            nrwt>>startingTerm, nrwt>>expansionCnt);
      SET result = Contains(text, FT_Any(NrwArray));

      RETURN (nrwt>>NOT_tag = result);
   END
```

Definitional Rules

1) *FT_MaxArrayLength* is the implementation-dependent maximum length for an array.

Description

1) The function *Contains(FullText, FT_NarrowerTerm)* takes the following input parameters:

   a) a *FullText* value *text*,
   b) an *FT_NarrowerTerm* value *nrwt*.

2) Let *R* be the result of

```
Contains(text, FT_Any(GetNarrowerTerms(nrwt>>thesaurus,
   nrwt>>startingTerm, nrwt>>expansionCnt)))
```

   Case:

   a) If *nrwt>>NOT_tag* is <u>unknown</u>, then *Contains(FullText, FT_NarrowerTerm)* returns <u>unknown</u>.

   b) If *nrwt>>NOT_tag* is <u>false</u>, then *Contains(FullText, FT_NarrowerTerm)* returns NOT *R*.

   c) Otherwise, *Contains(FullText, FT_NarrowerTerm)* returns *R*.

6.8.4 StrctPattern_to_FT_Pattern Function

Purpose

Convert an *FT_NarrowerTerm* value to an *FT_Pattern* value.

Definition

```
CREATE FUNCTION StrctPattern_to_FT_Pattern
   (nrwt FT_NarrowerTerm)
   RETURNS FT_Pattern
   BEGIN
      DECLARE result CHARACTER VARYING(FT_MaxPatternLength);

      IF nrwt IS NULL THEN
         RETURN CAST(NULL AS FT_Pattern)
      ELSEIF nrwt>>NOT_tag IS UNKNOWN THEN
         RETURN CAST(NULL AS FT_Pattern)
      END IF;

      SET result = 'NARROWER_TERM('
            || nrwt>>thesaurus
            || ','
            || CAST(StrctPattern_to_FT_Pattern(nrwt>>startingTerm)
                  AS CHARACTER VARYING(FT_MaxPatternLength))
            || ','
            || CASE WHEN nrwt>>expansionCnt IS NULL THEN
                  ‘ALL’
               ELSE
                  TRIM(BOTH' ' FROM CAST(nrwt>>expansionCnt
                     AS CHARACTER VARYING(FT_MaxPatternLength))
               END
            || ')';

      IF NOT nrwt>>NOT_TAG THEN
         SET result = 'NOT ' || result;
      END IF;
      RETURN CAST(result AS FT_Pattern);
   END
```

Definitional Rules

1) *FT_MaxPatternLength* is the implementation-dependent maximum length for the character representation of an instance of *FT_Pattern*.

Description

1) The function *StrctPattern_to_FT_Pattern(FT_NarrowerTerm)* takes the following input parameters:

   a) an *FT_NarrowerTerm* value *nrwt*.

2) *StrctPattern_to_FT_Pattern(FT_NarrowerTerm)* returns an *FT_Pattern* of the form <Narrower_Term function invocation> or NOT <Narrower_Term function invocation>.

3) If the input argument *nrwt* is the null value, or if *nrwt>>NOT_tag* is unknown, then the result is the null value.

6.8.5 GetNarrowerTerms Function

Purpose

Get narrower terms from a thesaurus.

Definition

```
CREATE FUNCTION GetNarrowerTerms
   (thes_name CHARACTER VARYING(FT_ThesNameLength),
    startingTerm FT_TextLiteral,
    thes_exp_count INTEGER)
   RETURNS FullText_Token ARRAY[FT_MaxArrayLength]
   BEGIN
      DECLARE ret FT_TextLiteral ARRAY[FT_MaxArrayLength];
      DECLARE strt FT_TextLiteral;
      DECLARE strt_termid INTEGER;
      DECLARE local_exp_count INTEGER;

      SET thes_name = TRIM(BOTH ' ' FROM thes_name);
      SET strt = TRIM(BOTH ' ' FROM startingTerm>>LitPart);

      SET local_exp_count =
         CASE
            WHEN thes_exp_count IS NOT NULL THEN
                thes_exp_count
            ELSE
                1
         END;

      SET strt_termid =
         (SELECT TERMID
          FROM TERM_DICTIONARY
          WHERE TRIM(BOTH ' ' FROM EXPR) = strt
                AND TRIM(BOTH ' ' FROM THNAME_DIC) = thes_name
         );

      SET ret=CAST(EMPTY AS FT_TextLiteral ARRAY[FT_MaxArrayLength]);

   L1: FOR elem AS
         WITH RECURSIVE done_so_far (TERMID,NARROWER_TERMID,LEVEL) AS
            (SELECT TERMID, NARROWER_TERMID, 0
             FROM TERM_HIERARCHY
             WHERE TERMID = strt_termid
               AND TRIM(BOTH ' ' FROM THNAME_HRR) = thes_name
               AND local_exp_count >= 0
                  UNION
             SELECT more.TERMID, more.NARROWER_TERMID,
                      CASE
                        WHEN thes_exp_count IS NOT NULL THEN
                           B.LEVEL + 1
                        ELSE
                           0
                      END AS LEVEL
             FROM done_so_far N, TERM_HIERARCHY more
             WHERE more.TERMID = N.NARROWER_TERMID
                AND TRIM(BOTH ' ' FROM more.THNAME_HRR) = thes_name
                AND N.LEVEL < local_exp_count
            )
         SELECT ARRAY[FT_TextLiteral(
               TRIM(BOTH ' ' FROM TD.EXPR)] AS EXRParr1
         FROM TERM_DICTIONARY TD, done_so_far f
         WHERE TD.TERMID = f.NARROWER.TERMID
               AND TRIM(BOTH ' ' FROM TD.THNAME_DIC) = thes_name

            DO  -- for every row of the above query result,
                -- append the value of column EXPRarr1 to the array

               SET ret = CONCATENATE(ret, EXPRarr1);
      END FOR L1;
   END
```

Definitional Rules

1) *FT_ThesNameLength* is the implementation-dependent maximum length for the character representation of a thesaurus name.

2) *FT_MaxArrayLength* is the implementation-dependent maximum length for an array.

Description

1) The private function *GetNarrowerTerms(CHARACTER VARYING, FT_TextLiteral, INTEGER)* takes the following input parameters:

   a) a CHARACTER VARYING value *thes_name*, denoting a thesaurus *TH*,
   b) an *FT_TextLiteral* value *strt*,
   c) an INTEGER value *thes_exp_count*.

2) *GetNarrowerTerms(CHARACTER VARYING, FT_TextLiteral, INTEGER)* returns an array of *FT_TextLiteral* elements which each represent a narrower term.

3) The result of *GetNarrowerTerms(CHARACTER VARYING, FT_TextLiteral, INTEGER)* is empty if one of the following holds:

   a) The term *strt* is not contained in column *BroaderTerm* of *TH*,
   b) either *strt* or *thes_name* is the null value,
   c) the expansion count *thes_exp_count* is smaller than zero.

4) If the expansion count *thes_exp_count* is zero, *GetNarrowerTerms(CHARACTER VARYING, FT_TextLiteral, INTEGER)* returns all terms in column *NarrowerTerm* of those rows of *TH* the values of which in column *BoraderTerm* are equivalent to *strt*. If the expansion count *thes_exp_count* is *n* > 0, the resulting array represents the set:

   MS_1 UNION MS_2

   where MS_1 is the multiset represented by the result of

   ```
   GetNarrowerTerms(thes_name, strt, thes_exp_count - 1)
   ```

   and MS_2 is given by

   $MS_{2,1}$ UNION ... $MS_{2,i}$ ... UNION $MS_{2,m}$

where *m* is the number of elements in MS_1, *i* ranges from 1 to *m*, E_i is some element of MS_1, and $MS_{2,i}$ is represented by

```
GetNarrowerTerms(thes_name, Ei, 0)
```

5) If the expansion count *thes_exp_count* is the null value, expansion is carried on until no new narrower terms can be found.

6) The term *strt* is **not** included in the result.

6.9 FT_Synonym Type and Routines

6.9.1 FT_Synonym Type

Purpose

FT_Synonym instances provide for the construction of synonym search patterns, and for searching of occurrences of synonyms in text.

Definition

```
CREATE TYPE FT_Synonym
   UNDER FT_Primary
   (startingTerm FT_TextLiteral,
    thesaurus CHARACTER VARYING(FT_ThesNameLength))
```

Definitional Rules

1) *FT_ThesNameLength* is the implementation-dependent maximum length for the character representation of a thesaurus name.

Description

1) The *FT_Synonym* type provides for public use:

   b) a function *FT_Synonym(CHARACTER VARYING, FT_TextLiteral),*
   c) a function *Contains(FullText, FT_Synonym),*.
   a) a function *StrctPattern_to_FT_Pattern(FT_*Synonym).

2) For the purpose of this data type, a thesaurus is effectively a table with one column, say *Ring*, the values of which represent sets of terms. In the context of such a thesaurus, two terms *T1* and *T2* are considered to be synonyms of each other, if the thesaurus contains at least one *Ring* value which contains both *T1* and *T2*.

3) The number of available thesauri and their names are implementation-defined.

6.9.2 FT_Synonym Function

Purpose

Construct and initialize an *FT_Synonym* instance.

Definition

```
CREATE FUNCTION FT_Synonym
   (thes_name CHARACTER VARYING(FT_ThesNameLength),
    strt FT_TextLiteral)
   RETURNS FT_Synonym
   BEGIN
      DECLARE temp FT_Synonym;
      SET temp = FT_Synonym();
      SET temp>>thesaurus = thes_name;
      SET temp>>startingTerm = strt;
      SET temp>>NotTag = TRUE;
      RETURN temp;
   END
```

Definitional Rules

1) *FT_ThesNameLength* is the implementation-dependent maximum length for the character representation of a thesaurus name.

Description

1) The function *FT_Synonym(CHARACTER VARYING*, *FT_TextLiteral)* takes the following input parameters:

   a) a CHARACTER VARYING value *thes_name*,
   b) an *FT_TextLiteral* value *strt*.

6.9.3 Contains Function

Purpose

Search a *FullText* instance for an *FT_Synonym.*

Definition

```
CREATE FUNCTION Contains
   (text FullText,
    synt FT_Synonym)
   RETURNS BOOLEAN
   BEGIN
      DECLARE SynArray FullText_Token ARRAY[FT_MaxArrayLength];
      DECLARE result BOOLEAN;

      SET SynArray = GetSynonymTerms(synt>>thesaurus,
            synt>>startingTerm);
      SET result = Contains(text, FT_Any(SynArray));

      RETURN (synt>>NOT_tag = result);
   END
```

Definitional Rules

1) *FT_MaxArrayLength* is the implementation-dependent maximum length for an array.

Description

1) The function *Contains(FullText, FT_Synonym)* takes the following input parameters:

   a) a *FullText* value *text*,
   b) an *FT_Synonym* value *synt*.

2) Let *R* be the result of

```
Contains(text, FT_Any(GetSynonymTerms(synt>>thesaurus,
   synt>>startingTerm)))
```

3) Case:

   a) If *synt>>NOT_tag* is unknown, then *Contains(FullText, FT_Synonym)* returns unknown.

   b) If *synt>>NOT_tag* is false, then *Contains(FullText, FT_Synonym)* returns NOT *R*.

   c) Otherwise, *Contains(FullText, FT_Synonym)* returns *R*.

6.9.4 StrctPattern_to_FT_Pattern Function

Purpose

Convert an *FT_Synonym* value to an *FT_Pattern* value.

Definition

```
CREATE FUNCTION StrctPattern_to_FT_Pattern
   (synt FT_Synonym)
   RETURNS FT_Pattern
   BEGIN
      DECLARE result CHARACTER VARYING(FT_MaxPatternLength);

      IF synt IS NULL THEN
         RETURN CAST(NULL AS FT_Pattern)
      ELSEIF synt>>NOT_tag IS UNKNOWN THEN
         RETURN CAST(NULL AS FT_Pattern)
      END IF;

      SET result = 'SYNONYM('
            || synt>>thesaurus
            || ','
            || CAST(StrctPattern_to_FT_Pattern(synt>>startingTerm)
                  AS CHARACTER VARYING(FT_MaxPatternLength))
            || ')';

      IF NOT synt>>NOT_TAG THEN
         SET result = 'NOT ' || result;
      END IF;
      RETURN CAST(result AS FT_Pattern);
   END
```

Definitional Rules

1) *FT_MaxPatternLength* is the implementation-dependent maximum length for the character representation of an instance of *FT_Pattern*.

Description

1) The function *StrctPattern_to_FT_Pattern(FT_Synonym)* takes the following input parameters:

   a) an *FT_Synonym* value *synt*.

2) *StrctPattern_to_FT_Pattern(FT_Synonym)* returns an *FT_Pattern* of the form <Synonym_Term function invocation> or NOT <Synonym_Term function invocation>.

3) If the input argument *synt* is the null value, or if *synt>>NOT_tag* is unknown, then the result is the null value.

6.9.5 GetSynonymTerms Function

Purpose

Get synonym terms from a thesaurus.

Definition

```
CREATE FUNCTION GetSynonymTerms
   (thes_name CHARACTER VARYING(FT_ThesNameLength),
    startingTerm FT_TextLiteral)
   RETURNS FullText_Token ARRAY[FT_MaxArrayLength]
   BEGIN
      DECLARE ret FT_TextLiteral ARRAY[FT_MaxArrayLength];
      DECLARE strt FT_TextLiteral;
      DECLARE strt_termid INTEGER;

      SET thes_name = TRIM(BOTH ' ' FROM thes_name);
      SET strt = TRIM(BOTH ' ' FROM startingTerm>>LitPart);
      SET strt_termid =
         (SELECT TERMID
          FROM TERM_DICTIONARY
          WHERE TRIM(BOTH ' ' FROM EXPR) = strt
                AND TRIM(BOTH ' ' FROM THNAME_DIC) = thes_name
         );

      SET ret=CAST(EMPTY AS FT_TextLiteral ARRAY[FT_MaxArrayLength]);

   L1: FOR elem AS
         WITH RECURSIVE done_so_far (TERMID,SYNONYM_TERMID) AS
            (SELECT TERMID, SYNONYM_TERMID
             FROM TERM_SYNONYM
             WHERE TERMID = strt_termid
               AND TRIM(BOTH ' ' FROM THNAME_HRR) = thes_name
                  UNION
             SELECT more.TERMID, more.SYNONYM_TERMID
             FROM done_so_far S, TERM_SYNONYM more
             WHERE more.TERMID = S.SYNONYM_TERMID
                AND TRIM(BOTH ' ' FROM more.THNAME_HRR) = thes_name
            )
         SELECT ARRAY[FT_TextLiteral(
               TRIM(BOTH ' ' FROM TD.EXPR)] AS EXRParr1
         FROM TERM_DICTIONARY TD, done_so_far f
         WHERE TD.TERMID = f.SYNONYM.TERMID
               AND TRIM(BOTH ' ' FROM TD.THNAME_DIC) = thes_name

            DO  -- for every row of the above query result,
                -- append the value of column EXPRarr1 to the array

               SET ret = CONCATENATE(ret, EXPRarr1);
      END FOR L1;
   END
```

Definitional Rules

1) *FT_ThesNameLength* is the implementation-dependent maximum length for the character representation of a thesaurus name.

2) *FT_MaxArrayLength* is the implementation-dependent maximum length for an array.

Description

1) The private function *GetSynonymTerms(CHARACTER VARYING, FT_TextLiteral)* takes the following input parameters:

   a) a CHARACTER VARYING value *thes_name*, denoting a thesaurus *TH*,
   b) an *FT_TextLiteral* value *startingTerm*.

2) *GetSynonymTerms(CHARACTER VARYING, FT_TextLiteral)* returns an array of *FT_TextLiteral* elements, which stands for a multiset of synonym terms.

3) The result of an invocation of *GetSynonymTerms(CHARACTER VARYING, FT_TextLiteral)* is empty if one of the following holds:

   a) The term *strt* is not contained in *Ring* value of *TH*,
   b) either *strt* or *thes_name* is the null value.

4) Let *n* be the number of Ring values containing *strt*, and let R_i denote such a value. The result of invoking *GetSynonymTerms(CHARACTER VARYING, FT_TextLiteral)* represents the following set:

   R_1 UNION ... R_i ... UNION R_m

5) The term *strt is* included in the result.

6.10 FT_PreferredTerm Type and Routines

6.10.1 FT_PreferredTerm Type

Purpose

FT_Preferred Term instances provide for the construction of preferred term search patterns, and for searching of occurrences of the associated preferred terms in text.

Definition

```
CREATE TYPE FT_PreferredTerm
   UNDER FT_Primary
   (thesaurus    CHARACTER VARYING(FT_ThesNameLength),
    startingTerm FT_TextLiteral)
```

Definitional Rules

1) *FT_ThesNameLength* is the implementation-dependent maximum length for the character representation of a thesaurus name.

Description

1) The *FT_PreferredTerm* type provides for public use:

   a) a function *FT_PreferredTerm(CHARACTER VARYING, FT_TextLiteral),*
   b) a function *Contains(FullText, FT_PreferredTerm),*
   c) a function *StrctPattern_to_FT_Pattern(FT_PreferredTerm).*

2) For the purpose of this data type, a thesaurus is effectively a table with three columns, say *PreferredTerm*, *TermId*, and *SynonymTerm*, the values of which represent terms. For a given row, two values *TermId* and *SynonymTerm* represent terms which are synonyms of each other, and *PreferredTerm* represents a preferred term associated with either of the former terms.

3) The number of available thesauri and their names are implementation-defined.

6.10.2 FT_PreferredTerm Function

Purpose

Construct and initialize an *FT_PreferredTerm* instance.

Definition

```
CREATE FUNCTION FT_PreferredTerm
   (thes_name CHARACTER VARYING(FT_ThesNameLength),
    strt FT_TextLiteral)
   RETURNS FT_PreferredTerm
   BEGIN
      DECLARE temp FT_PreferredTerm;
      SET temp = FT_PreferredTerm();
      SET temp>>thesaurus = thes_name;
      SET temp>>startingTerm = strt;
      SET temp>>NotTag = TRUE;
      RETURN temp;
   END
```

Definitional Rules

1) *FT_ThesNameLength* is the implementation-dependent maximum length for the character representation of a thesaurus name.

Description

1) The function *FT_PreferredTerm(CHARACTER VARYING, FT_TextLiteral)* takes the following input parameters:

   a) an CHARACTER VARYING value *thes_name*,
   b) an *FT_TextLiteral* value *strt*.

6.10.3 Contains Function

Purpose

Search a *FullText* instance for an *FT_PreferredTerm.*

Definition

```
CREATE FUNCTION Contains
   (text FullText,
    pfdt FT_PreferredTerm)
   RETURNS BOOLEAN
   BEGIN
      DECLARE PfdArray FullText_Token ARRAY[FT_MaxArrayLength];
      DECLARE result BOOLEAN;

      SET PfdArray = GetPreferredTerms(pfdt>>thesaurus,
            pfdt>>startingTerm);
      SET result = Contains(text, FT_Any(PrdArray));

      RETURN (pfdt>>NOT_tag = result);
   END
```

Definitional Rules

1) *FT_MaxArrayLength* is the implementation-dependent maximum length for an array.

Description

1) The function *Contains(FullText, FT_PreferredTerm)* takes the following input parameters:

   a) a *FullText* value *text*,
   b) an *FT_PreferredTerm* value *strt*.

2) Let *R* be the result of

```
Contains(text, FT_Any(GetPreferredTerms(pfdt>>thesaurus,
   pfdt>>startingTerm)))
```

3) Case:

   a) If *pfdt>>NOT_tag* is unknown, then *Contains(FullText, FT_PreferredTerm)* returns unknown.

   b) If *pfdt>>NOT_tag* is false, then *Contains(FullText, FT_PreferredTerm)* returns NOT *R*.

   c) Otherwise, *Contains(FullText, FT_PreferredTerm)* returns *R*.

6.10.4 StrctPattern_to_FT_Pattern Function

Purpose

Convert an *FT_PreferredTerm* value to an *FT_Pattern* value.

Definition

```
CREATE FUNCTION StrctPattern_to_FT_Pattern
   (pfdt FT_PreferredTerm)
   RETURNS FT_Pattern
   BEGIN
      DECLARE result CHARACTER VARYING(FT_MaxPatternLength);

      IF pfdt IS NULL THEN
         RETURN CAST(NULL AS FT_Pattern)
      ELSEIF pfdt>>NOT_tag IS UNKNOWN THEN
         RETURN CAST(NULL AS FT_Pattern)
      END IF;

      SET result = 'PREFERRED_TERM('
            || pfdt>>thesaurus
            || ','
            || CAST(StrctPattern_to_FT_Pattern(pfdt>>startingTerm)
                  AS CHARACTER VARYING(FT_MaxPatternLength))
            || ')';

      IF NOT pfdt>>NOT_TAG THEN
         SET result = 'NOT ' || result;
      END IF;
      RETURN CAST(result AS FT_Pattern);
   END
```

Definitional Rules

1) *FT_MaxPatternLength* is the implementation-dependent maximum length for the character representation of an instance of *FT_Pattern*.

Description

1) The function *StrctPattern_to_FT_Pattern(FT_PreferredTerm)* takes the following input parameters:

   a) an *FT_PreferredTerm* value *pfdt*.

2) *StrctPattern_to_FT_Pattern(PreferredTerm)* returns into an *FT_Pattern* of the form <Preferred_Term function invocation> or NOT <Preferred_Term function invocation>.

3) If the input argument *pfdt* is the null value, or if *pfdt>>NOT_tag* is unknown, then the result is the null value.

6.10.5 GetPreferredTerms Function

Purpose

Get preferred terms from a thesaurus.

Definition

```
CREATE FUNCTION GetPreferredTerms
   (thes_name CHARACTER VARYING(FT_ThesNameLength),
    startingTerm FT_TextLiteral)
   RETURNS FullText_Token ARRAY[FT_MaxArrayLength]
   BEGIN
      DECLARE ret FT_TextLiteral ARRAY[FT_MaxArrayLength];
      DECLARE strt FT_TextLiteral;
      DECLARE strt_termid INTEGER;

      SET thes_name = TRIM(BOTH ' ' FROM thes_name);
      SET strt = TRIM(BOTH ' ' FROM startingTerm>>LitPart);
      SET strt_termid =
         (SELECT TERMID
          FROM TERM_DICTIONARY
          WHERE TRIM(BOTH ' ' FROM EXPR) = strt
                AND TRIM(BOTH ' ' FROM THNAME_DIC) = thes_name
         );

      SET ret=CAST(EMPTY AS FT_TextLiteral ARRAY[FT_MaxArrayLength]);

   L1: FOR elem AS
         WITH temp_preferred (TERMID) AS
            (SELECT PREFERRED_TERMID
             FROM TERM_SYNONYM
             WHERE TERMID = strt_termid
               AND TRIM(BOTH ' ' FROM THNAME_HRR) = thes_name
            )
         SELECT ARRAY[FT_TextLiteral(
               TRIM(BOTH ' ' FROM TD.EXPR)] AS EXRParr1
         FROM TERM_DICTIONARY TD, temp_preferred
         WHERE TD.TERMID = temp_preferred.TERMID
               AND TRIM(BOTH ' ' FROM TD.THNAME_DIC) = thes_name

            DO  -- for every row of the above query result,
                -- append the value of column EXPRarr1 to the array

               SET ret = CONCATENATE(ret, EXPRarr1);
      END FOR L1;
   END
```

Definitional Rules

1) *FT_ThesNameLength* is the implementation-dependent maximum length for the character representation of a thesaurus name.

2) *FT_MaxArrayLength* is the implementation-dependent maximum length for an array.

Description

1) The private function *GetPreferredTerms(CHARACTER VARYING, FT_TextLiteral)* takes the following input parameters:

    a) a CHARACTER VARYING value *thes_name*, denoting a thesaurus *TH*,
    b) an *FT_TextLiteral* value *strt*.

2) *GetPreferredTerms(CHARACTER VARYING, FT_TextLiteral)* returns an array of *FT_TextLiteral* elements which stands for a set of preferred terms.

3) The result of an invocation of *GetPreferredTerms* is empty if one of the following holds:

    a) In table TERM_SYNONYM there is no pair (TERMID, THNAME_SYN), such that the TERMID value represents *strt*, and the THNAME_SYN value is equivalent to *thes_name*,
    b) either *strt* or *thes_name* is the null value.

4) Otherwise, for every row of TERM_SYNONYM with a pair (TERMID, THNAME_SYN)) such that the TERMID value represents *strt* and the THNAME_SYN value is equivalent to *thes_name*, the term represented by the PREFERRED_TERMID value is included in the result.

5) The term *strt* **is** included in the result.

6.11 FT_RelatedTerm Type and Routines

6.11.1 FT_RelatedTerm Type

Purpose

FT_RelatedTerm instances provide for the construction of related term search patterns, and for searching of occurrences of the associated related terms in text.

Definition

```
CREATE TYPE FT_RelatedTerm
   UNDER FT_Primary
   (thesaurus CHARACTER VARYING(FT_ThesNameLength),
    startingTerm FT_TextLiteral)
```

Definitional Rules

1) *FT_ThesNameLength* is the implementation-dependent maximum length for the character representation of a thesaurus name.

Description

1) The *FT_RelatedTerm* type provides for public use:

   a) a function *FT_RelatedTerm(CHARACTER VARYING, FT_TextLiteral)*,
   b) a function *Contains(FullText, FT_RelatedTerm),*
   c) a function *StrctPattern_to_FT_Pattern(FT_RelatedTerm)*.

2) For the purpose of this data type, a thesaurus is effectively a table, say *TH*, with two columns *Term* and *Related_Term*. For a given row, the two values *Term* and *Related_Term* represent terms such that the second is related to the first one.

3) The number of available thesauri and their names are implementation-defined.

6.11.2 FT_RelatedTerm Function

Purpose

Construct and initialize an *FT_RelatedTerm* instance.

Definition

```
CREATE FUNCTION FT_RelatedTerm
   (thes_name CHARACTER VARYING(FT_ThesNameLength),
    strt FT_TextLiteral)
   RETURNS FT_RelatedTerm
   BEGIN
      DECLARE temp FT_RelatedTerm;
      SET temp = FT_RelatedTerm();
      SET temp>>thesaurus = thes_name;
      SET temp>>startingTerm = strt;
      SET temp>>NotTag = TRUE;
      RETURN temp;
   END
```

Definitional Rules

1) *FT_ThesNameLength* is the implementation-dependent maximum length for the character representation of a thesaurus name.

Description

1) The function *FT_RelatedTerm(CHARACTER VARYING, FT_TextLiteral)* takes the following input parameters:

   a) a CHARACTER VARYING value *thes_name*,
   b) an *FT_TextLiteral* value *strt*.

6.11.3 Contains Function

Purpose

Search a *FullText* instance for an *FT_RelatedTerm.*

Definition

```
CREATE FUNCTION Contains
   (text FullText,
    rltdt FT_RelatedTerm)
   RETURNS BOOLEAN
   BEGIN
      DECLARE RltdArray FullText_Token ARRAY[FT_MaxArrayLength];
      DECLARE result BOOLEAN;

      SET RltdArray = GetRelatedTerms(rltdt>>thesaurus,
            rltdt>>startingTerm);
      SET result = Contains(text, FT_Any(RltdArray));

      RETURN (rltdt>>NOT_tag = result);
   END
```

Definitional Rules

1) *FT_MaxArrayLength* is the implementation-dependent maximum length for an array.

Description

1) The function *Contains(FullText, FT_RelatedTerm)* takes the following input parameters:

   a) a *FullText* value *text*,
   b) an *FT_RelatedTerm* value *rltdt*.

2) Let *R* be the result of

```
Contains(text, FT_Any(GetRelatedTerms(rltdt>>thesaurus,
   rltdt>>startingTerm)))
```

3) Case:

   a) If *rltdt>>NOT_tag* is unknown, then *Contains(FullText, FT_RelatedTerm)* returns unknown.

   b) If *rltdt>>NOT_tag* is false, then *Contains(FullText, FT_RelatedTerm)* returns NOT *R*.

   c) Otherwise, *Contains(FullText, FT_RelatedTerm)* returns *R*.

6.11.4 StrctPattern_to_FT_Pattern Function

Purpose

Convert an *FT_RelatedTerm* value to an *FT_Pattern* value.

Definition

```
CREATE FUNCTION StrctPattern_to_FT_Pattern
   (rltdt FT_RelatedTerm)
   RETURNS FT_Pattern
   BEGIN
      DECLARE result CHARACTER VARYING(FT_MaxPatternLength);

      IF rltdt IS NULL THEN
         RETURN CAST(NULL AS FT_Pattern)
      ELSEIF rltdt>>NOT_tag IS UNKNOWN THEN
         RETURN CAST(NULL AS FT_Pattern)
      END IF;

      SET result = 'RELATED_TERM('
            || rltdt>>thesaurus
            || ','
            || CAST(StrctPattern_to_FT_Pattern(rltdt>>startingTerm)
                  AS CHARACTER VARYING(FT_MaxPatternLength))
            || ')';

      IF NOT rltdt>>NOT_TAG THEN
         SET result = 'NOT ' || result;
      END IF;
      RETURN CAST(result AS FT_Pattern);
   END
```

Definitional Rules

1) *FT_MaxPatternLength* is the implementation-dependent maximum length for the character representation of an instance of *FT_Pattern*.

Description

1) The function *StrctPattern_to_FT_Pattern(FT_RelatedTerm)* takes the following input parameters:

   a) an *FT_RelatedTerm* value *rltdt*.

2) *StrctPattern_to_FT_Pattern(FT_RelatedTerm)* returns an *FT_Pattern* of the form <Related_Term function invocation> or NOT <Related_Term function invocation>.

3) If the input argument *rltdt* is the null value, or if *rltdt>>NOT_tag* is unknown, then the result is the null value.

6.11.5 GetRelatedTerms Function

Purpose

Get related terms from a thesaurus.

Definition

```
CREATE FUNCTION GetRelatedTerms
   (thes_name CHARACTER VARYING(FT_ThesNameLength),
    startingTerm FT_TextLiteral)
   RETURNS FullText_Token ARRAY[FT_MaxArrayLength]
   BEGIN
      DECLARE ret FT_TextLiteral ARRAY[FT_MaxArrayLength];
      DECLARE strt FT_TextLiteral;
      DECLARE strt_termid INTEGER;

      SET thes_name = TRIM(BOTH ' ' FROM thes_name);
      SET strt = TRIM(BOTH ' ' FROM startingTerm>>LitPart);
      SET strt_termid =
         (SELECT TERMID
          FROM TERM_DICTIONARY
          WHERE TRIM(BOTH ' ' FROM EXPR) = strt
                AND TRIM(BOTH ' ' FROM THNAME_DIC) = thes_name
         );

      SET ret=CAST(EMPTY AS FT_TextLiteral ARRAY[FT_MaxArrayLength]);

   L1: FOR elem AS
         WITH temp_related (TERMID) AS
            (SELECT RELATED_TERMID
             FROM TERM_RELATED
             WHERE TERMID = strt_termid
               AND TRIM(BOTH ' ' FROM THNAME_HRR) = thes_name
            )
         SELECT ARRAY[FT_TextLiteral(
               TRIM(BOTH ' ' FROM TD.EXPR)] AS EXRParr1
         FROM TERM_DICTIONARY TD, temp_related
         WHERE (TD.TERMID = temp_related.TERMID
               OR TD.TERMID = strt_termid)
               AND TRIM(BOTH ' ' FROM TD.THNAME_DIC) = thes_name

            DO  -- for every row of the above query result,
                -- append the value of column EXPRarr1 to the array

               SET ret = CONCATENATE(ret, EXPRarr1);
      END FOR L1;
   END
```

Definitional Rules

1) *FT_ThesNameLength* is the implementation-dependent maximum length for the character representation of a thesaurus name.

2) *FT_MaxArrayLength* is the implementation-dependent maximum length for an array.

Description

1) The private function *GetRelatedTerms(CHARACTER VARYING, FT_TextLiteral)* takes the following input parameters:

    a) a CHARACTER VARYING value *thes_name*, denoting a thesaurus *TH*,
    b) an *FT_TextLiteral* value *startingTerm*.

2) *GetRelatedTerms(CHARACTER VARYING, FT_TextLiteral)* returns an array of *FT_TextLiteral* elements which stands for a set of related terms.

3) The result of an invocation of *GetRelatedTerms(CHARACTER VARYING, FT_TextLiteral)* is empty if one of the following holds:

    a) In thesaurus table *TH* denoted by *thes_name* there is no row such that the *Term* value represents *strt*.
    b) Either *strt* or *thes_name* is the null value.

4) Otherwise, for every row of *TH* with a pair (*Term*, *Related_Term*) such that the *Term* value represents *strt*, the term represented by the *Related_Term* value is included in the result.

6.12 FT_TopTerm Type and Routines

6.12.1 FT_TopTerm Type

Purpose

FT_TopTerm instances provide for the construction of top term search patterns, and for searching of occurrences of the associated top terms in text.

Definition

```
CREATE TYPE FT_TopTerm
   UNDER FT_Primary
   (thesaurus CHARACTER VARYING(FT_ThesNameLength),
    startingTerm FT_TextLiteral)
```

Definitional Rules

1) *FT_ThesNameLength* is the implementation-dependent maximum length for the character representation of a thesaurus name.

Description

1) The *FT_TopTerm* type provides for public use:

   a) a function *FT_TopTerm(CHARACTER VARYING, FT_TextLiteral),*
   b) a function *Contains(FullText, FT_TopTerm),*
   c) a function *StrctPattern_to_FT_Pattern(FT_TopTerm).*

2) For the purpose of this data type, a thesaurus is effectively a table with two columns, *NarrowerTerm* and *BroaderTerm*. For a given row, the values contained in the two columns represent terms, the first being a narrower term of the second one.

3) The number of available thesauri and their names are implementation-defined.

6.12.2 FT_TopTerm Function

Purpose

Construct and initialize an *FT_TopTerm* instance.

Definition

```
CREATE FUNCTION FT_TopTerm
   (thes_name CHARACTER VARYING(FT_ThesNameLength),
    strt FT_TextLiteral)
   RETURNS FT_TopTerm
   BEGIN
      DECLARE temp FT_TopTerm;
      SET temp = FT_TopTerm();
      SET temp>>thesaurus = thes_name;
      SET temp>>startingTerm = strt;
      SET temp>>NotTag = TRUE;
      RETURN temp;
   END
```

Definitional Rules

1) *FT_ThesNameLength* is the implementation-dependent maximum length for the character representation of a thesaurus name.

Description

1) The function *FT_TopTerm(CHARACTER VARYING, FT_TextLiteral)* takes the following input parameters:

   a) a CHARACTER VARYING value *thes_name*,
   b) an *FT_TextLiteral* value *strt*.

6.12.3 Contains Function

Purpose

Search a *FullText* instance for an *FT_TopTerm.*

Definition

```
CREATE FUNCTION Contains
   (text FullText,
    topt FT_TopTerm)
   RETURNS BOOLEAN
   BEGIN
      DECLARE TopdArray FullText_Token ARRAY[FT_MaxArrayLength];
      DECLARE result BOOLEAN;

      SET TopArray = GetTopTerms(topt>>thesaurus,
            topt>>startingTerm);
      SET result = Contains(text, FT_Any(TopArray));

      RETURN (topt>>NOT_tag = result);
   END
```

Definitional Rules

1) *FT_MaxArrayLength* is the implementation-dependent maximum length for an array.

Description

1) The function *Contains(FullText, FT_TopTerm)* takes the following input parameters:

   a) a *FullText* value *text*,
   b) an *FT_TopTerm* value *topt*.

2) Let *R* be the result of

```
Contains(text, FT_Any(GetTopTerms(topt>>thesaurus,
   topt>>startingTerm)))
```

3) Case:

   a) If *topt>>NOT_tag* is unknown, then *Contains(FullText, FT_TopTerm)* returns unknown.

   b) If *topt>>NOT_tag* is false, then *Contains(FullText, FT_TopTerm)* returns NOT *R*.

   c) Otherwise, *Contains(FullText, FT_TopTerm)* returns *R*.

6.12.4 StrctPattern_to_FT_Pattern Function

Purpose

Convert an *FT_TopTerm* value to an *FT_Pattern* value.

Definition

```
CREATE FUNCTION StrctPattern_to_FT_Pattern
   (topt FT_TopTerm)
   RETURNS FT_Pattern
   BEGIN
      DECLARE result CHARACTER VARYING(FT_MaxPatternLength);

      IF topt IS NULL THEN
         RETURN CAST(NULL AS FT_Pattern)
      ELSEIF topt>>NOT_tag IS UNKNOWN THEN
         RETURN CAST(NULL AS FT_Pattern)
      END IF;

      SET result = 'TOP_TERM('
            || topt>>thesaurus
            || ','
            || CAST(StrctPattern_to_FT_Pattern(topt>>startingTerm)
                  AS CHARACTER VARYING(FT_MaxPatternLength))
            || ')';

      IF NOT topt>>NOT_TAG THEN
         SET result = 'NOT ' || result;
      END IF;
      RETURN CAST(result AS FT_Pattern);
   END
```

Definitional Rules

1) *FT_MaxPatternLength* is the implementation-dependent maximum length for the character representation of an instance of *FT_Pattern*.

Description

1) The function *StrctPattern_to_FT_Pattern(FT_TopTerm)* takes the following input parameters:

   a) an *FT_TopTerm* value *topt*.

2) *StrctPattern_to_FT_Pattern(FT_TopTerm)* returns an *FT_Pattern* of the form <Top_Term function invocation> or NOT <Top_Term function invocation>.

3) If the input argument *topt* is the null value, or if *topt>>NOT_tag* is unknown, then the result is the null value.

6.12.5 GetTopTerms Function

Purpose

Get top terms from a thesaurus.

Definition

```
CREATE FUNCTION GetTopTerms
   (thes_name CHARACTER VARYING(FT_ThesNameLength),
    startingTerm FT_TextLiteral)
   RETURNS FullText_Token ARRAY[FT_MaxArrayLength]
   BEGIN
      DECLARE ret FT_TextLiteral ARRAY[FT_MaxArrayLength];
      DECLARE strt FT_TextLiteral;
      DECLARE strt_termid INTEGER;

      SET thes_name = TRIM(BOTH ' ' FROM thes_name);
      SET strt = TRIM(BOTH ' ' FROM startingTerm>>LitPart);
      SET strt_termid =
         (SELECT TERMID
          FROM TERM_DICTIONARY
          WHERE TRIM(BOTH ' ' FROM EXPR) = strt
                AND TRIM(BOTH ' ' FROM THNAME_DIC) = thes_name
         );

      SET ret=CAST(EMPTY AS FT_TextLiteral ARRAY[FT_MaxArrayLength]);

   L1: FOR elem AS
         WITH RECURSIVE done_so_far (TERMID, NARROWER_TERMID) AS
            (SELECT TERMID, NARROWER_TERMID
             FROM TERM_HIERARCHY
             WHERE NARROWER_TERMID = strt_termid
               AND TRIM(BOTH ' ' FROM THNAME_HRR) = thes_name
               AND local_exp_count >= 0
                  UNION
             SELECT more.TERMID, more.NARROWER_TERMID
             FROM done_so_far B, TERM_HIERARCHY more
             WHERE more.NARROWER_TERMID = B.TERMID
                AND TRIM(BOTH ' ' FROM more.THNAME_HRR) = thes_name
            )
         SELECT ARRAY[FT_TextLiteral(
               TRIM(BOTH ' ' FROM TD.EXPR)] AS EXRParr1
         FROM TERM_DICTIONARY TD, done_so_far f
         WHERE TD.TERMID = f.TERMID
               AND TRIM(BOTH ' ' FROM TD.THNAME_DIC) = thes_name
               AND NOT EXISTS
               (SELECT *
                FROM done_so_far d
                WHERE d.NARROWER_TERMID = f.TERMID
               )

            DO  -- for every row of the above query result,
                -- append the value of column EXPRarr1 to the array

               SET ret = CONCATENATE(ret, EXPRarr1);
      END FOR L1;
    END
```

Definitional Rules

1) *FT_ThesNameLength* is the implementation-dependent maximum length for the character representation of a thesaurus name.

2) *FT_MaxArrayLength* is the implementation-dependent maximum length for an array.

Description

1) The function *GetTopTerms(CHARACTER VARYING, FT_TextLiteral)* takes the following input parameters:

   a) a CHARACTER VARYING value *thes_name*,
   b) an *FT_TextLiteral* value *startingTerm*.

2) *GetTopTerms(CHARACTER VARYING, FT_TextLiteral)* returns an array of *FT_TextLiteral* elements, which stands for a set of narrower terms.

3) The result of an invocation of *GetTopTerms(CHARACTER VARYING, FT_TextLiteral)* is equivalent to the result of invoking *GetBroaderTerms(CHARACTER VARYING, FT_TextLiteral)*, using *thes_name*, *strt*, and NULL as input arguments, and subsequently removing all terms for which there exists a broader term according to the thesaurus denoted by *thes_name*.

4) The term *strt is* **not** included in the result.

6.13 FT_IsAbout Type and Routines

6.13.1 FT_IsAbout Type

Purpose

FT_IsAbout instances provide for the construction of search patterns stating a topic in form of a *FullText* value, and and for testing whether a text is pertinent to this value.

Definition

```
CREATE TYPE FT_IsAbout UNDER FT_Primary
   (phr FullText)
```

Description

1) The *FT_IsAbout* type provides for public use:

   a) a function *FT_IsAbout(FullText),*
   b) a function *Contains(FullText, FT_IsAbout),*
   c) a function *StrctPattern_to_FT_Pattern(FT_IsAbout).*

6.13.2 FT_IsAbout Function

Purpose

Construct and initialize an *FT_IsAbout* instance.

Definition

```
CREATE FUNCTION FT_IsAbout
   (phr FullText)
   RETURNS FT_IsAbout
   BEGIN
      DECLARE temp FT_IsAbout;
      SET temp = FT_IsAbout();
      SET temp>>phr = phr;
      SET temp>>NotTag = TRUE;
      RETURN temp;
   END
```

Description

1) The function *FT_IsAbout(FullText)* takes the following input parameters:

   a) a *FullText* value *phr*.

6.13.3 Contains Function

Purpose

Search a *FullText* instance for an *FT_IsAbout.*

Definition

```
CREATE FUNCTION Contains
   (text FullText,
    phr FT_IsAbout)
   RETURNS BOOLEAN
   BEGIN
      DECLARE result BOOLEAN;
      --
      -- !! See description
      --
      RETURN result;
   END
```

Description

1) The function *Contains(FullText, FT_IsAbout)* takes the following input parameters:

   a) a *FullText* value *text*,
   b) an *FT_IsAbout* value *phr*.

2) *Contains(FullText, FT_IsAbout)* tests whether a given *FullTex*t item is pertinent to the *FullText* item of a given *FT_IsAbout* instance. The result is subject to implementor-defined criteria of pertinence.

Editor's Note 2-046

The definition of FT_IsAbout is entirely implementation-defined. Is there really any value in standardizing something with no conformance criteria, and hence no portability expectations?

6.13.4 StrctPattern_to_FT_Pattern Function

Purpose

Convert an *FT_IsAbout* value to an *FT_Pattern* value.

Definition

```
CREATE FUNCTION StrctPattern_to_FT_Pattern
   (phr FT_IsAbout)
   RETURNS FT_Pattern
   BEGIN
      --
      -- !! See description
      --
   END
```

Description

1) The function *StrctPattern_to_FT_Pattern(FT_IsAbout)* takes the following input parameters:

   a) an *FT_IsAbout* value *phr*.

****Editor's Note 2-014****

StrctPattern_to_FT_Pattern Function code or description for FT_IsAbout type to be supplied.

6.14 FT_Context Type and Routines

6.14.1 FT_Context Type

Purpose

FT_Context instances represent context search patterns.

Definition

```
CREATE TYPE FT_Context
   UNDER FT_Primary
   (ArgArray FT_PhraseList ARRAY[FT_MaxArrayLength],
    du FullText_Token)
```

Definitional Rules

1) *FT_MaxArrayLength* is the implementation-dependent maximum length for an array.

Description

1) The *FT_Context* type provides for public use:

   a) a function *FT_Context(FT_PhraseList, FT_PhraseList, FT_PhraseList ARRAY, FullText_Token),*
   b) a function *Contains(FullText, FT_Context),*
   c) a function *StrctPattern_to_FT_Pattern(FT_Context).*

6.14.2 FT_Context Function

Purpose

Constructs and initialize an *FT_Context* instance.

Definition

```
CREATE FUNCTION FT_Context
   (Arg1 FT_PhraseList,
    Arg2 FT_PhraseList,
    Arg3 FT_PhraseList ARRAY[FT_MaxArrayLength],
    DistanceUnit FullText_Token)
   RETURNS FT_Context
   BEGIN
      DECLARE temp FT_Context;
      SET temp = FT_Context();
      SET temp>>NotTag = TRUE;
      SET temp>>ArgArray = CONCATENATE(ARRAY[Arg1, Arg2], Arg3);
      SET temp>>du  = DistanceUnit;
      RETURN temp;
   END
```

Definitional Rules

1) *FT_MaxArrayLength* is the implementation-dependent maximum length for an array.

Description

1) The function *FT_Context(FT_PhraseList, FT_PhraseList, FT_PhraseList ARRAY, FullText_Token)* takes the following input parameters:

   a) an *FT_PhraseList* value *Arg1*,
   b) an *FT_PhraseList* value *Arg2*,
   c) a (possibly empty) array *Arg3* the elements of which are *FT_PhraseList* instances,
   d) a *FullText_Token* value *DistanceUnit*.

2) All arguments may be the null value.

6.14.3 Contains Function

Purpose

Search a *FullText* instance for an *FT_Context.*

Definition

```
CREATE FUNCTION Contains
   (text FullText,
    context FT_Context)
   RETURNS FT_Boolean
   BEGIN
      DECLARE result BOOLEAN;
      DECLARE ftl FullText ARRAY[FT_MaxArrayLength];
      DECLARE segno INTEGER;
      DECLARE argno INTEGER;

      IF context>>du <> 'SENTENCES' OR
           context>>du <> 'PARAGRAPHS' THEN
         RETURN
         --
         -- !! See Description
         --
      END IF;

      IF context IS NULL THEN
         SET argno = CAST(NULL AS INTEGER);
      ELSEIF context>>ArgArray IS NULL THEN
         SET argno = CAST(NULL AS INTEGER)
      ELSE
         SET argno = CARDINALITY(context>>ArgArray);
      END IF;

      SET ftl = Segmentize(text, context>>du);

      IF ftl IS NULL THEN
         SET segno = CAST(NULL AS INTEGER);
      ELSE
         SET segno = CARDINALITY(ftl);
      END IF;

      IF setno IS NULL THEN
         RETURN UNKNOWN;
      ELSEIF segno = 0 THEN
         SET RESULT = FALSE;
      ELSEIF (segno <> 0 AND argno = 0) THEN
         SET RESULT = TRUE;
      ELSEIF (segno <>0 AND argno IS NULL) THEN
         SET RESULT = UNKNOWN;
      ELSE
         SET RESULT =

         (WITH RECURSIVE SegTab(ind, seg) AS
            (VALUES(1, ftl[1])
               UNION
             SELECT ind + 1, ftl[ind + 1]
```

```
                FROM   SegTab
                WHERE  ind < segno
               ),
               ContextTab(ind, ca) AS
               (VALUES(1, context>>ArgArray[1])
                  UNION
                SELECT ind + 1, context>>ArgArray[ind + 1]
                FROM   ContextTab
                WHERE  ind < argno
               )
            VALUES
               (FOR SOME SegTab st(ind, seg)
                  (FOR ALL ContextTab ct(ind, ca)
                     (Contains(seg, ca))
                  )
               )
            );
         END IF;
         RETURN (context>>NOT_tag = result);
      END
```

Definitional Rules

1) *FT_MaxArrayLength* is the implementation-dependent maximum length for an array.

Description

1) The function *Contains(FullText, FT_Context)* takes the following input parameters:

   a) a *FullText* value *text*,
   b) an *FT_Context* value *context*.

2) Case:

   a) If either *text* or *context* or *context>>ArgArray* is the null value, then the result of *Contains(text, context)* is unknown.

   b) Otherwise, let *n* be the number of elements of *context>>ArgArray*, and for *i* ranging from 1 to *n*, let CA_i be the elements of *context>>ArgArray*. Depending on the distance unit *context>>du* specified, let *m* be the number of sentences (paragraphs) of text, and for *j* ranging from 1 to *m*, let SEG_j be the *FullText* instances representing these sentences (paragraphs).

   Case:

   i) If there exists some SEG_j, such that the result of

      Contains(SEG_j, CA_i)

      is true, for every Ca_i , then let *R* be true.

   i) If for every SEG_j, such that the result of

      Contains(SEG_j, CA_i)

      is false, for at least one Ca_i , then let *R* be false.

iii) Otherwise, let *R* be unknown.

3) *Contains(FullText, FT_Context)* returns:

Case:

a) unknown, if *context>>NOT_tag* is unknown.

b) NOT *R*, if *context>>NOT_tag* is false.

c) Otherwise, *R*.

6.14.4 StrctPattern_to_FT_Pattern Function

Purpose

Convert an *FT_Context* value to an *FT_Pattern* value.

Definition

```
CREATE FUNCTION StrctPattern_to_FT_Pattern
   (context FT_Context)
   RETURNS FT_Pattern
   BEGIN
      DECLARE result CHARACTER VARYING(FT_MaxPatternLength);
      DECLARE i INTEGER;
      DECLARE n INTEGER;

      IF context IS NULL THEN
         RETURN NULL;
      ELSEIF context>>ArgArray IS NULL THEN
         RETURN NULL;
      ELSEIF context>>NOT_tag IS UNKNOWN THEN
         RETURN NULL;
      END IF;

      SET n = CARDINALITY(context>>ArgArray);
      SET result =
         CAST(StrctPattern_to_FT_Pattern(context>>ArgArray[1])
           AS CHARACTER VARYING(FT_MaxPatternLength)
         || 'IN SAME '
         || CASE
          WHEN UPPER(TRIM(BOTH ' ' FROM context>>du)) = 'SENTENCES'
             THEN 'SENTENCE'
          WHEN UPPER(TRIM(BOTH ' ' FROM context>>du)) = 'PARAGRAPHS'
            THEN 'PARAGRAPH'
          ELSE context>>du
            END
         ||' AS ' ||
         CAST(StrctPattern_to_FT_Pattern(context>>ArgArray[2])
           AS CHARACTER VARYING(FT_MaxPatternLength));

      SET i = 3;

      L1: WHILE (n >= i) DO
            SET result = result || ' AND '||
             CAST(StrctPattern_to_FT_Pattern(context>>ArgArray[i])
               AS CHARACTER VARYING(max_pattern_length));
         SET i = i + 1;
      END WHILE L1;

      IF NOT context>>NOT_tag THEN
         SET result = 'NOT ' || result;
      END IF;
      RETURN CAST(result AS FT_Pattern);
   END
```

Definitional Rules

1) *FT_MaxPatternLength* is the implementation-dependent maximum length for the character representation of an instance of *FT_Pattern*.

Description

1) The function *StrctPattern_to_FT_Pattern(FT_Context)* takes the following input parameters:

   a) an *FT_Context* value *context*.

2) *StrctPattern_to_FT_Pattern(FT_Context)* returns an *FT_Pattern* of the form <context condition> or NOT <context condition>.

3) If the input argument *context* or *context>>ArgArray* is the null value, or if *context>>NOT_tag* is unknown, then the result is the null value.

6.15 FT_ParExpr Type and Routines

6.15.1 FT_ParExpr Type

Purpose

FT_ParExpr provides for the construction of *FT_Term* patterns as *FT_Primary* instances of the type *FT_ParExpr*, for searching occurrences of *FT_ParExpr* patterns in *FullText* items, and for turning *FT_ParExpr* instances into equivalent *FT_Pattern* instances.

Definition

```
CREATE TYPE FT_ParExpr
   UNDER FT_Primary
   (Body FT_Expr)
```

Description

1) The *FT_ParExpr* type provides for public use:

   a) a function *FT_ParExpr(FT_Expr)*,
   b) a function *Contains(FullText, FT_ParExpr)*,
   c) a function *StrctPattern_to_FT_Pattern(FT_ParExpr)*.

****Editor's Note 2-042****
FT_ParExpr type states that its functions are provided "for public use." This is not really correct since they are defined for internal use only and would not be exposed to users of the FullText type.

6.15.2 FT_ParExpr Function

Purpose

Construct and initialize an *FT_ParExpr* instance.

Definition

```
CREATE FUNCTION FT_ParExpr
   (expr FT_Expr)
   RETURNS FT_ParExpr
   BEGIN
      DECLARE temp FT_ParExpr;
      SET temp = FT_ParExpr();
      SET temp>>Body = expr;
      SET temp>>NOT_tag = TRUE;
      RETURN temp;
   END
```

Description

1) The function *FT_ParExpr(FT_Expr)* takes the following input parameters:

   a) an *FT_Expr* value *expr*.

6.15.3 Contains Function

Purpose

Search a *FullText* instance for an *FT_ParExpr*.

Definition

```
CREATE FUNCTION Contains
   (text FullText,
    part FT_ParExpr)
   RETURNS BOOLEAN
   BEGIN
      RETURN Contains(text, part>>Body)
   END
```

Description

1) The function *Contains(FullText, FT_ParExpr)* takes the following input parameters:

   a) a *FullText* value *text*,
   b) an *FT_ParExpr* value *part*.

2) The result of *Contains(text, part)* is the result of *Contains(text, part>>Body)*.

6.15.4 StrctPattern_to_FT_Pattern Function

Purpose

Convert an *FT_ParExpr* value to an *FT_Pattern* value.

Definition

```
CREATE FUNCTION StrctPattern_to_FT_Pattern
   (part FT_ParExpr)
   RETURNS(FT_Pattern)
   BEGIN
      DECLARE result CHARACTER VARYING(FT_MaxPatternLength);
      IF part IS NULL THEN
         RETURN CAST(NULL AS FT_Pattern);
      ELSEIF part>>NOT_tag IS NULL THEN
         RETURN CAST(NULL AS FT_Pattern);
      END IF;

      SET result = '('|| CAST(StrctPattern_to_FT_Pattern(part>>Body)
         AS CHARACTER VARYING(FT_MaxPatternLength)) || ')';

      IF NOT part>>NOT_Tag THEN
         SET result = 'NOT ' || result;
      END IF;
      RETURN CAST(result AS FT_Pattern);
   END
```

Definitional Rules

1) *FT_MaxPatternLength* is the implementation-dependent maximum length for the character representation of an instance of *FT_Pattern*.

Description

1) The function *StrctPattern_to_FT_Pattern(FT_ParExpr)* takes the following input parameters:

   a) an *FT_ParExpr* value *part*.

2) *StrctPattern_to_FT_Pattern(FT_ParExpr)* returns an *FT_Pattern* of the form <left paren> <search expression> <right paren> except for the following cases:

   a) If *part* is the null value, or if *part>>NOT_tag* is the null value, then the result is the null value.

   b) If *StrctPattern_to_FT_Pattern(part>>Body)* is the null value, then the result is the null value.

6.16 FT_Term Type and Routines

6.16.1 FT_Term Type

Purpose

FT_Term instances represent search patterns consisting of a sequence of *FT_Primary* search patterns; all items in the list are intended to be matched.

Definition

```
CREATE TYPE FT_Term
   (ConjunctsArray FT_Primary ARRAY[FT_MaxArrayLength])
```

Definitional Rules

1) *FT_MaxArrayLength* is the implementation-dependent maximum length for an array.

Description

1) The *FT_Term* type provides for public use:

   a) a function *FT_Term(FT_Primary ARRAY),*
   b) a function *Contains(FullText, FT_Term),*
   c) a function *StrctPattern_to_FT_Pattern(FT_Term).*

6.16.2 FT_Term Function

Purpose

Construct and initialize an *FT_Term* instance.

Definition

```
CREATE FUNCTION FT_Term
   (pArray FT_Primary ARRAY[FT_MaxArrayLength])
   RETURNS FT_Term
   BEGIN
      DECLARE temp FT_Term;
      DECLARE i INTEGER;

      SET temp = FT_Term();
      IF pArray IS NULL THEN
         RETURN temp;
      END IF;
      SET i = 1;
      SET temp>>ConjunctsArray =
            CAST(EMPTY AS FT_Primary ARRAY[FT_MaxArrayLength]);

   L1: WHILE (i <= CARDINALITY(pArray)) DO
         SET temp>>ConjunctsArray =
               CONCATENATE(temp>>ConjunctsArray, ARRAY[pArray[i]]);
         SET i = i + 1;
      END WHILE L1;

      RETURN temp;
   END
```

Definitional Rules

1) *FT_MaxArrayLength* is the implementation-dependent maximum length for an array.

Description

1) The function *FT_Term(FT_Primary ARRAY)* takes the following input parameters:

   a) an array *pArray* with elements of type *FT_Primary*.

2) *pArray* may be empty or the null value.

   Note: The definition of *FT_Term* instances is intentionally more general than the definition of the corresponding <search term>s.

6.16.3 Contains Function

Purpose

Search a *FullText* instance for an *FT_Term.*

Definition

```
CREATE FUNCTION Contains
   (text FullText,
    term FT_Term)
   RETURNS BOOLEAN
   BEGIN
      DECLARE i INTEGER ;
      DECLARE result BOOLEAN;

      IF term IS NULL THEN
         RETURN UNKNOWN;
      ELSEIF term>>ConjunctsArray IS NULL THEN
         RETURN UNKNOWN;
      END IF;

      SET i = 1 ;
      SET result = TRUE;
   L1: WHILE (i <= CARDINALITY(term>>ConjunctsArray))
            AND (result IS TRUE OR result IS UNKNOWN) DO
         SET result = result
               AND Contains(text, term>>ConjunctsArray[i]));

         SET i = i + 1;
      END WHILE L1;

      RETURN result;
   END
```

Description

1) The function *Contains(FullText, FT_Term)* takes the following input parameters:

   a) a *FullText* value *text*,
   b) an *FT_Term* value *term*.

2) The result of *Contains(FullText, FT_Term)* is:

   Case:

   a) unknown, if *term* or *term>>ConjunctsArray* is the null value.

   b) true, if for all *FT_Primary* elements *P* of *term>>ConjunctsArray*

   ```
   Contains(text, P)
   ```

   returns true.

   c) false, if at least one *FT_Primary* element *P* of *term>>ConjunctsArray* is such that

```
Contains(text, P)
```

returns <u>false</u>.

d) Otherwise, <u>unknown</u>.

6.16.4 StrctPattern_to_FT_Pattern Function

Purpose

Convert an *FT_Term* value to an *FT_Pattern* value.

Definition

```
CREATE FUNCTION StrctPattern_to_FT_Pattern
   (term FT_Term)
   RETURNS FT_Pattern
   BEGIN
      DECLARE i INTEGER;
      DECLARE result CHARACTER VARYING(FT_MaxPatternLength);

      IF term IS NULL THEN
         RETURN CAST(NULL AS FT_Pattern);
      END IF;

      SET i = 1;
      SET result = '';

   L1: WHILE(i <= CARDINALITY(term>>ConjunctsArray)) DO
         SET result = result
            || CAST(
               StrctPattern_to_FT_Pattern(term>>ConjunctsArray[i])
                  AS CHARACTER VARYING(FT_MaxPatternLength))
            || '&';
         SET i = i + 1;
      END WHILE L1;

      SET result = TRIM(TRAILING '&' FROM result);
      RETURN CAST(result AS FT_Pattern);
   END
```

Definitional Rules

1) *FT_MaxPatternLength* is the implementation-dependent maximum length for the character representation of an instance of *FT_Pattern*.

Description

1) The function *StrctPattern_to_FT_Pattern(FT_Term)* takes the following input parameters:

   a) an *FT_Term* value *term*.

2) *StrctPattern_to_FT_Pattern(FT_Term)* returns an *FT_Pattern* of the form <search term>, except for the following cases:

   a) If *term>>ConjunctsArray* is empty, the result is represented by an empty string.

   b) If *term* or *term>>ConjunctsArray* is the null value, then the result is the null value.

   c) If any element of *term>>ConjunctsArray* is the null value, then the result is the null value.

6.17 FT_Expr Type and Routines

6.17.1 FT_Expr Type

Purpose

FT_Expr instances represent search patterns consisting of a sequence of *FT_Term* search patterns; at least one item in such a list is intended to be matched.

Definition

```
CREATE TYPE FT_Expr
   (DisjunctsArray FT_Term ARRAY[FT_MaxArrayLength])
```

Definitional Rules

1) *FT_MaxArrayLength* is the implementation-dependent maximum length for an array.

Description

1) The *FT_Expr* type provides for public use:

   a) a function *FT_Expr(FT_Term ARRAY),*
   b) a function *Contains(FullText, FT_Expr),*
   c) a function *StrctPattern_to_FT_Pattern(FT_Expr).*

6.17.2 FT_Expr Function

Purpose

Construct and initialize an *FT_Expr* instance.

Definition

```
CREATE FUNCTION FT_Expr
   (tArray FT_Term ARRAY[FT_MaxArrayLength])
   RETURNS FT_Expr
   BEGIN
      DECLARE temp FT_Expr;
      DECLARE i INTEGER;

      SET temp = FT_Expr();
      IF tArray IS NULL THEN
         RETURN temp;
      END IF;
      SET i = 1;
      SET temp>>DisjunctsArray =
            CAST(EMPTY AS FT_Term ARRAY[FT_MaxArrayLength]);

   L1: WHILE (i <= CARDINALITY(tArray)) DO
         SET temp>>DisjunctsArray =
               CONCATENATE(temp>>DisjunctsArray, ARRAY[tArray[i]]);
         SET i = i + 1;
      END WHILE L1;

      RETURN temp;
   END
```

Definitional Rules

1) *FT_MaxArrayLength* is the implementation-dependent maximum length for an array.

Description

1) The function *FT_Expr(FT_Term ARRAY)* takes the following input parameters:

   a) an array *tArray* with elements of type *FT_Term*.

2) *tArray* may be empty or the null value.

   Note: The definition of *FT_Expr* instances is intentionally more general than the definition of the corresponding <search expression>s.

6.17.3 Contains Function

Purpose

Search a *FullText* instance for an *FT_Expr*.

Definition

```
CREATE FUNCTION Contains
   (text FullText,
    expr FT_Expr)
   RETURNS BOOLEAN
   BEGIN
      DECLARE i INTEGER ;
      DECLARE result BOOLEAN;

      IF expr IS NULL THEN
         RETURN UNKNOWN;
      ELSEIF expr>>DisjunctsArray IS NULL THEN
         RETURN UNKNOWN;
      END IF;

      SET i = 1 ;
      SET result = FALSE;
   L1: WHILE (i <= CARDINALITY(expr>>DisjunctsArray))
            AND (result IS FALSE OR result IS UNKNOWN) DO
         SET result = result
               OR Contains(text, expr>>DisjunctsArray[i]));

         SET i = i + 1;
      END WHILE L1;

      RETURN result;
   END
```

Description

1) The function *Contains(FullText, FT_Expr)* takes the following input parameters:

   a) a *FullText* value *text*,
   b) an *FT_Expr* value *expr*.

2) The result of *Contains(FullText, FT_Expr)* is:

   Case:

   a) unknown, if *expr* or *expr>>DisjunctsArray* is the null value.

   b) true, if at least one *FT_Term* element *T* of *expr>>DisjunctsArray*

      ```
      Contains(text, T)
      ```

      returns true.

   c) false, if for all *FT_Term* elements *T* of *expr>>DisjunctsArray* is such that

```
Contains(text, T)
```

returns <u>false</u>.

d) Otherwise, <u>unknown</u>.

6.17.4 StrctPattern_to_FT_Pattern Function

Purpose

Convert an *FT_Expr* value to an *FT_Pattern* value.

Definition

```
CREATE FUNCTION StrctPattern_to_FT_Pattern
   (expr FT_Expr)
   RETURNS FT_Pattern
   BEGIN
      DECLARE i INTEGER;
      DECLARE result CHARACTER VARYING(FT_MaxPatternLength);

      IF term IS NULL OR expr>>DisjunctsArray IS NULL THEN
         RETURN CAST(NULL AS FT_Pattern);
      END IF;

      SET i      = 1;
      SET result = '';

      L1: WHILE(i <= CARDINALLITY(expr>>DisjunctsArray)) DO
         SET result = result
            || CAST(
               StrctPattern_to_FT_Pattern(expr>>DisjunctsArray[i])
                  AS CHARACTER VARYING(FT_MaxPatternLength))
            || '|';
         SET i = i + 1;
      END WHILE L1;

      SET result = TRIM(TRAILING '|' FROM result);
      RETURN CAST(result AS FT_Pattern);
   END
```

Definitional Rules

1) *FT_MaxPatternLength* is the implementation-dependent maximum length for the character representation of an instance of *FT_Pattern*.

Description

1) The function *StrctPattern_to_FT_Pattern(Ft_Expr)* takes the following input parameters:

   a) an *FT_Expr* value *expr*.

2) *StrctPattern_to_FT_Pattern(Ft_Expr)* returns an *FT_Pattern* of the form <search expression> except for the following cases:

a) If *expr>>DisjunctsList* is empty, the result is represented by an empty string.
b) If *expr* or *expr>>DisjunctsList* is the null value, then the result is the null value.
c) If any element of *expr>>DisjunctsList* is the null value, then the result is the null value.

6.18 FT_PhraseList Type and Routines

6.18.1 FT_PhraseList Type

Purpose

FT_PhraseList type provides facilities for the construction of a structured search pattern that represents a multiset of element type of which is *FT_Phrase*, and for testing whether at least one of the members of such a multiset occurs in a given instance of type *FullText*.

Definition

```
CREATE TYPE FT_PhraseList
   (Phrases FT_Phrase ARRAY[FT_MaxArrayLength])
```

Definitional Rules

1) *FT_MaxArrayLength* is the implementation-dependent maximum length for an array.

Description

1) The *FT_PhraseList* type provides for public use:

   a) a function *FT_PhraseList(FT_Phrase ARRAY)*,
   b) a function *Contains(FullText, FT_PhraseList)*,
   c) a function *StrctPattern_to_FT_Pattern(FT_PhraseList)*.

6.18.2 FT_PhraseList Function

Purpose

Construct and initialize an *FT_PhraseList* instance.

Definition

```
CREATE FUNCTION FT_PhraseList
   (phra FT_Phrase ARRAY[FT_MaxArrayLength])
   RETURNS FT_PhraseList
   BEGIN
      DECLARE temp FT_PhraseList;

      SET temp = FT_PhraseList();
      SET temp>>Phrases = phra;
      RETURN temp;
   END
```

Definitional Rules

1) *FT_MaxArrayLength* is the implementation-dependent maximum length for an array.

Description

1) The function *FT_PhraseList(FT_Phrase ARRAY)* takes the following input parameters:

   a) an array *phra* with elements of type *FT_Phrase*.

6.18.3 Contains Function

Purpose

Search a *FullText* instance for an *FT_PhraseList*.

Definition

```
CREATE FUNCTION Contains
   (text FullText,
    phrl FT_PhraseList)
   RETURNS BOOLEAN
   BEGIN
      DECLARE i INTEGER ;
      DECLARE result BOOLEAN;
      DECLARE TokArray FullText_Token ARRAY[FT_MaxArrayLength];
      DECLARE lenp INTEGER;
      DECLARE lent INTEGER;

      IF phrl IS NULL THEN
         SET lenp = CAST(NULL AS INTEGER);
      ELSE
         SET lenp = CARDINALITY(phrl>>Phrases);
      END IF;

      SET TokArray = Tokenize(text);
      IF TokArray IS NULL THEN
         SET lent = CAST(NULL AS INTEGER);
      ELSE
         SET lenp = CARDINALITY(TokArray);
      END IF;

      IF lent IS NULL AND lenp IS NULL THEN
         RETURN UNKNOWN;
      ELSEIF lent = 0 OR lenp = 0 THEN
         SET result = FALSE;
      ELSEIF lent <> 0 AND lenp IS NULL
            OR lent IS NULL AND lenp <> 0 THEN
         RETURN UNKNOWN;
      ELSE SET result =

      (WITH RECURSIVE phrlTab(ind, phr) AS
         (VALUES(1, phrl>>Phrases[1])
            UNION
          SELECT ind + 1, phrl>>Phrases[ind + 1]
          FROM phrlTab
          WHERE ind < lenp
         )
       VALUES
         (FOR SOME phrlTab pt(ind,phr)
            (Contains(text, phr))
         )
      );
      END IF;
   END
```

Definitional Rules

1) *FT_MaxArrayLength* is the implementation-dependent maximum length for an array.

Description

1) The function *Contains (FullText, FT_PhraseList)* takes the following input parameters:

   a) a *FullText* value *text*,
   b) an *FT_PhraseList* value *phrl*.

2) The result of *Contains (FullText, FT_PhraseList)* is:

   Case:

   a) false, if *phrl>>Phrases* is empty or if for every element *P* of *phrl>>Phrases* the result of

      ```
      Contains(text, P)
      ```

      is false.

   b) true, if there exists at least one element *P* of *phrl>>Phrases* such that the result of

      ```
      Contains(text, P)
      ```

      is true.

   c) Otherwise, unknown,.

      In particular, this result is obtained if:

      i) Any of *text* or *Tokenize(text)* is the null value, and *phrl* or *phrl>>Phrases* is the null value.

      ii) *text* or *Tokenize(text)* is the null value, but *phrl>>Phrases* is an non-empty array.

      iii) *phrl* or *phrl>>Phrases* is the null value, but *Tokenize(text)* is an non-empty array.

6.18.4 StrctPattern_to_FT_Pattern Function

Purpose

Convert an *FT_PhraseList* value to an *FT_Pattern* value.

Definition

```
CREATE FUNCTION StrctPattern_to_FT_Pattern
   (phrl FT_PhraseList)
   RETURNS FT_Pattern
   BEGIN
      DECLARE i INTEGER;
      DECLARE result CHARACTER VARYING(FT_MaxPatternLength);
      DECLARE len INTEGER;

      IF phrl IS NULL THEN
         RETURN CAST(NULL AS FT_Pattern);
      ELSEIF phrl>>Phrases IS NULL THEN
         RETURN CAST(NULL AS FT_Pattern);
      ELSE
         SET len = CARDINALITY(phrl>>Phrases);
      END IF;

      SET i = 1;
      SET result = '(';

      L1: WHILE(i <= len) DO
         SET result = result
            || CAST(
               StrctPattern_to_FT_Pattern(phrl>>Phrases[i])
                  AS CHARACTER VARYING(FT_MaxPatternLength))
            || ',';
         SET i = i + 1;
      END WHILE L1;

      SET result = TRIM(TRAILING ',' FROM result);
      RETURN CAST(result AS FT_Pattern);
   END
```

Definitional Rules

1) *FT_MaxPatternLength* is the implementation-dependent maximum length for the character representation of an instance of *FT_Pattern*.

Description

1) The function *StrctPattern_to_FT_Pattern(FT_PhraseList)* takes the following input parameters:

   a) an *FT_PhraseList* value *phrl*.

2) *StrctPattern_to_FT_Pattern(FT_PhraseList)* returns an *FT_Pattern* of the form <text literal list> except for the following cases:

   a) If *phrl* or *phrlr>>Phrases* is the null value, then the result is the null value.

b) If any element of *phrl>>Phrases* is the null value, then the result is the null value.

7 FullText_Token Type and Routines

7.1 FullText_Token Type

Purpose

The *FullText_Token* domain is used to define valid tokens.

Definition

```
CREATE DOMAIN FullText_Token
   AS CHARACTER VARYING(FT_MaxTokenLength)
   CHECK(p(VALUE))
```

Definitional Rules

1) *FT_MaxTokeLength* is the implementation-dependent maximum length for the character representation of an instance of *FullText_Token*.

Description

1) The function *p* returns *true* if and only if the character string *VALUE* is a valid token. It is implementation-defined whether a character string is a valid token.

8 SQL/MM Thesaurus Schema

8.1 Introduction

The only purpose of the SQL/MM thesaurus schema is to provide a data model to support understanding of the thesaurus related functions.

The base tables are all defined in a <schema definition> for the schema named SQLMM_THESAURUS. The table definitions are as complete as the definitional power of SQL allows. The table definitions are supplemented with assertions where appropriate. Each description comprises three parts:

1. The function of the definition is stated.

2. The SQL definition of the object is presented as a <table definition>.

3. An explanation of the object.

The specification provides only a model of the base tables that are required, and does not imply that an implementation shall provide the functionality in the manner described in this clause.

8.2 SQLMM_THESAURUS Schema

Purpose

Create the schema that is to contain the base tables that underlie the SQL/MM Thesaurus Schema.

Definition

```
CREATE SCHEMA SQLMM_THESAURUS
   AUTHORIZATION SQLMM_THESAURUS
```

8.3 TERM_DICTIONARY base table

Purpose

The TERM_DICTIONARY base table has one row for each term referenced in the SQL/MM Thesaurus Schema of the catalog. These are all those terms that can be found in the TERM_HIERARCHY, TERM_SYNONYM and TERM_RELATE tables.

Definition

```
CREATE TABLE TERM_DICTIONARY
   (
   TERMID     INTEGER NOT NULL DEFAULT 0,
   EXPR       CHARACTER VARYING(FT_ThesTermLength),
   THNAME_DIC CHARACTER VARYING(FT_ThesNameLength),

   PRIMARY KEY (TERMID, THNAME_DIC)
   )
```

Definitional Rules

1) *FT_ThesNameLength* is the implementation-dependent maximum length for the character representation of a thesaurus name.

2) *FT_ThesTermLength* is the implementation-dependent maximum length for the character representation of a thesaurus term.

Description

1) The number of available thesauri and their names are implementation-defined.

8.4 TERM_HIERARCHY base table

Purpose

The TERM_HIERARCHY base table has one row for each pair of terms that form a broader-narrower term pair.

Definition

```
CREATE TABLE TERM_HIERARCHY
   (
   TERMID            INTEGER NOT NULL,
   NARROWER_TERMID   INTEGER,
   THNAME_HRR        CHARACTER VARYING(FT_ThesNameLength),

   PRIMARY KEY(TERMID, NARROWER_TERMID, THNAME_HRR),

   FOREIGN KEY(NARROWER_TERMID, THNAME_HRR) REFERENCES TERM_DICTIONARY,
   FOREIGN KEY(TERMID, THNAME_HRR) REFERENCES TERM_DICTIONARY
```

Definitional Rules

1) *FT_ThesNameLength* is the implementation-dependent maximum length for the character representation of a thesaurus name.

Description

1) The number of available thesauri and their names are implementation-defined.

8.5 TERM_SYNONYM base table

Purpose

The TERM_SYNONYM base table has one row for each pair of terms that form a synonym term pair. Each row also indicates the preferred term for the synonym term pair.

Definition

```
CREATE TABLE TERM_SYNONYM
   (
   TERMID            INTEGER NOT NULL,
   SYNONYM_TERMID    INTEGER NOT NULL,
   PREFERRED_TERMID  INTEGER,
   THNAME_SYN        CHARACTER VARYING(FT_ThesNameLength),

   PRIMARY KEY(TERMID, SYNONYM_TERMID, THNAME_SYN),

   FOREIGN KEY(SYNONYM_TERMID, THNAME_SYN)   REFERENCES TERM_DICTIONARY,
   FOREIGN KEY(PREFERRED_TERMID, THNAME_SYN) REFERENCES TERM_DICTIONARY,
   FOREIGN KEY(TERMID, THNAME_SYN)           REFERENCES TERM_DICTIONARY
   )
```

Definitional Rules

1) *FT_ThesNameLength* is the implementation-dependent maximum length for the character representation of a thesaurus name.

Description

1) The number of available thesauri and their names are implementation-defined.

8.6 TERM_RELATED base table

Purpose

The TERM_RELATED base table has one row for each pair of terms that form a related term pair.

Definition

```
CREATE TABLE TERM_RELATED
   (
   TERMID            INTEGER NOT NULL,
   RELATED_TERMID    INTEGER NOT NULL,
   THNAME_REL        CHARACTER VARYING(FT_ThesNameLength),


   PRIMARY KEY(TERMID, RELATED_TERMID, THNAME_REL),

   FOREIGN KEY(RELATED_TERMID, THNAME_REL)   REFERENCES TERM_DICTIONARY,
   FOREIGN KEY(TERMID, THNAME_REL)           REFERENCES TERM_DICTIONARY
   )
```

Definitional Rules

1) *FT_ThesNameLength* is the implementation-dependent maximum length for the character representation of a thesaurus name.

Description

1) The number of available thesauri and their names are implementation-defined.

9 Status Codes

The character string value returned in an SQLSTATE parameter comprises a 2-character class value followed by a 3-character subclass value. The class value for each condition and the subclass value or values for each class value are specified in Table 1 - SQLSTATE class and subclass values.

The "Category" column has the following meanings: "S" means that the class value given corresponds to successful completion and is a completion condition; "W" means that the class value given corresponds to a successful completion but with a warning and is a completion condition; "N" means that the class value corresponds to a no-data situation and is a completion condition; "X" means that the class value given corresponds to an exception condition.

Table 1 – SQLSTATE class and subclass values

Category	Condition	Class	Subcondition	Subclass
X	SQL/MM Full-Text	H2	invalid search expression	F01

10 Conformance

10.1 Introduction

This part of ISO/IEC 13249 specifies conforming SQL/MM Full-Text implementations.

A conforming SQL/MM Full-Text implementation shall support the public Full-Text data types and functions according to the associated Definitions and Description Rules specified in this part of ISO/IEC 13249.

A conforming SQL/MM Full-Text implementation shall supply <SQL-invoked function>s whose <routine body> is either a <SQL routine body> or an <external body reference> that specifies PARAMETER STYLE SQL as defined in Subclause 12.5, "<SQL-invoked routine>" in part 2 of ISO 9075.

A conforming SQL/MM Full-Text implementation is not required to perform the exact sequence of actions defined in the Description Rules or in the <SQL routine body>s contained this International Standard, but shall achieve the same effect as that sequence.

10.2 Relationship to other International Standards

****Editor's Note 2-040****
Relationships to other International Standards to be supplied. This section needs to explain the dependency on ISO/IEC 9075.

10.3 Claims of conformance

Claims of conformance to this part of ISO/IEC 13249 shall state:

****Editor's Note 2-041****
Claims of conformance are to be supplied.

1) The definitions for all elements and actions that this part of ISO/IEC 13049 specifies as implementation-defined.

10.4 Extensions and options

A conforming implementation may provide support for additional implementation-defined routines defined using the Full-Text data types.

An implementation remains conforming even if it provides user options to process Full-Text routines in a nonconforming manner.

Index