Date: 2000-02-29

ISO/IEC FDIS 13249-2:2000 (E)

ISO/IEC JTC 1/SC 32/WG 4

Secretariat: U.S.A.

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.

Attention is drawn to the possibility that some elements of this part of ISO/IEC 13249 may be the subject of patent rights. ISO and IEC shall not be held responsible for identifying any or all such patent rights.

International Standard ISO/IEC 13249-2 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information Technology*, Subcommittee SC32, *Data Management and interchange*.

ISO/IEC 13249 consists of the following parts, under the general title *Information technology — Database languages — SQL Multimedia and Application Packages*:

- *Part 1: Framework*
- *Part 2: Full-Text*
- *Part 3: Spatial*
- *Part 5: Still Image*

Annexes A and B of this part of ISO/IEC 13249 are for information only.

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.

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

The organization of this part of ISO/IEC 13249 is as follows:

1) Clause 1, "Scope", specifies the scope of this part of ISO/IEC 13249.

2) Clause 2, "Normative references", identifies additional standards that, through reference in this part of ISO/IEC 13249, constitute provisions of this part of ISO/IEC 13249.

3) Clause 3, "Definitions, notations, and conventions", defines the notations and conventions used in this part of ISO/IEC 13249.

4) Clause 4, "Concepts", presents concepts used in the definition of this part of ISO/IEC 13249.

5) Clause 5, "Full-Text Types", defines the full-text user-defined types and associated routines.

6) Clause 6, "Structured Search Pattern Types", defines a family of user-defined types to provide for the construction of structured search patterns.

7) Clause 7, "FullText_Token Type and Routines", defines the user-defined FullText_Token type.

8) Clause 8, "SQL/MM Full-Text Thesaurus Schema", defines the SQL/MM Full-Text thesaurus schema used to define the thesaurus related routines.

9) Clause 9, "SQL/MM Full-Text Information Schema", defines the SQL/MM Full-Text Information Schema.

10) Clause 10, "SQL/MM Full-Text Definition Schema", defines the SQL/MM Full-Text Definition Schema.

11) Clause 11, "Status Codes", defines the SQLSTATE codes used in this part of ISO/IEC 13249.

12) Clause 12, "Conformance", defines the criteria for conformance to this part of ISO/IEC 13249.

13) Annex A, "Implementation-defined elements", is an informative Annex. It lists those features for which the body of this part of ISO/IEC 13249 states that the syntax or meaning or effect on the database is partly or wholly implementation-defined, and describes the defining information that an implementor shall provide in each case.

14) Annex B, "Implementation-dependent elements", is an informative Annex. It list those features which the body of this part of ISO/IEC 13249 states explicitly that the syntax or meaning or effect on the database is implementation-dependent.

In the text of this part of ISO/IEC 13249, Clauses begin a new odd-numbered page, and in Clause 5, "Full-Text Types", through Clause 12, "Conformance", Subclauses begin a new page. Any resulting blank space is not significant.

Information technology — Database languages — SQL Multimedia and Application Packages — Part 2: Full-Text

1 Scope

This part of ISO SQL/MM:

a) introduces the Full-Text part of ISO/IEC 13249,

b) gives the references necessary for this part of ISO/IEC 13249,

c) defines notations and conventions specific to this part of ISO/IEC 13249,

d) defines concepts specific to this part of ISO/IEC 13249,

e) defines the full-text user-defined 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 part of ISO/IEC 13249. For dated references, subsequent amendments to, or revisions of, any of these publications do not apply. However, parties to agreements based on this part of ISO/IEC 13249 are encouraged to investigate the possibility of applying the most recent editions of the normative documents indicated below. For undated references, the latest edition of the normative document referred to applies. Members of IEC and ISO maintain registers of currently valid International Standards.

2.1 International standards

ISO/IEC 9075-1:1999, *Information technology — Database languages — SQL — Part 1: Framework (SQL/Framework).*

ISO/IEC 9075-2:1999, *Information technology — Database languages — SQL — Part 2: Foundation (SQL/Foundation).*

ISO/IEC 9075-4:1999, *Information technology — Database languages — SQL — Part 4: Persistent Stored Modules (SQL/PSM).*

ISO/IEC 13249-1:1999, *Information technology — Database languages — SQL Multimedia and Application Packages — Part 1: Framework.*

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

For the purpose of this part of ISO/IEC 13249, the following definitions apply.

3.1.1 Definitions provided in Part 1

This part of ISO/IEC 13249 makes use of all terms defined ISO/IEC 13249-1.

3.1.2 Definitions provided in Part 2

3.1.2.1
broader term
A superordinate term in a hierarchical relation (e.g. a broader term for "SQL" is "Database Language").

3.1.2.2
coordinate relation
A formal relation juxtaposing terms or classes of terms.

3.1.2.3
hierarchical relation
A formal relation between two terms or classes in which one term is subordinate to the other term.

3.1.2.4
narrower term
A subordinate term in a hierarchical relation (e.g. a narrower term for "SQL" is "SQL/MM").

3.1.2.5
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").

3.1.2.6
related term
A term connected to another term by a coordinate relation (e.g. a related term for "SQL" is "DB2").

3.1.2.7
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").

3.1.2.8
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").

3.1.2.9
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".

3.1.3 Definitions taken from ISO/IEC 9075

This part of ISO/IEC 13249 makes use of the following terms defined in ISO/IEC 9075:

3.1.3.1
contain

3.1.3.2
immediately contain

3.1.3.3
simply contain

3.1.3.4
SQL-invoked routine

3.1.4 Definitions taken from ANSI/NISO Z39.19

This part of ISO/IEC 13249 makes use of the following terms defined in ANSI/NISO Z39.19:

3.1.4.1
thesaurus

3.2 Notations

The notations used in this part of ISO/IEC 13249 are defined in ISO/IEC 13249-1.

3.3 Conventions

The conventions used in this part of ISO/IEC 13249 are defined in ISO/IEC 13249-1.

4 Concepts

4.1 Text model

Text as modeled by the types and routines of this part of ISO/IEC 13249 is any sequence of characters which represents one of the following:

- a single word,
- a sequence of words,
- a single sentence,
- a sequence of sentences,
- a single paragraph,
- a sequence of paragraphs.

A sentence consists of one or more words. A paragraph consists of one or more sentences.

When modeled as a value of the *FullText* type of this part of ISO/IEC 13249 a text value is associated with a specific language. The recognition of word, sentence and paragraph boundaries is largely governed by language specific rules, conventions, and heuristics. It is implementation-defined which of these rules, conventions, and heuristics are applied by a given implementation.

4.2 Text identification facilities

For identifying specific *FullText* values in collections of *FullText* values this part of ISO/IEC 13249 provides facilities for testing whether a text represented by a given *FullText* value matches a certain pattern (i.e. whether that pattern occurs in that text).

Like text, patterns are sequences of characters, representing one of the following:

- a single word (patterns of the form <word>),
- a set of words (patterns of the form <word> with wild card characters, patterns of the form <token list>, patterns of the form <stemmed word>, patterns of the form <expansion function invocation>, or certain patterns of the form <text literal list>),
- a phrase, i.e. a representation of a sequence of words (patterns of the form <phrase>),
- a set of phrases (patterns of the form <phrase> with wild card characters, patterns of the form <stemmed phrase>, patterns of the form <expansion function invocation>, or certain patterns of the form <text literal list>),
- a set of words and/or phrases (patterns of the form <text literal list> or patterns of the form <expansion function invocation>),
- sets of two or more patterns, each either consisting of a single word or phrase, or a set composed of context patterns (patterns of the form <Proximity expansion>, or patterns of the form <context condition>),
- patterns formed by patterns and Boolean operators for negation, conjunction, or disjunction (patterns of the form <search expression> | <search term>, patterns of the form <search term> & <search factor>, or patterns of the form NOT <search primary>).

Each word pattern and single phrase pattern is either explicitly or implicitly associated with a specific language.

To illustrate the effects of patterns the following text samples represented by values of the *FullText* type (to be referred to as *firstSample*, *secondSample*, and *thirdSample*) will be used:

firstSample:

```
As assumed by this International Standard, every text value is
associated with a specific language. The recognition of word,
sentence and paragraph boundaries is largely governed by language
specific rules, conventions, and heuristics; it is implementation-
defined which of these rules, conventions, and heuristics are applied
by a given implementation.
```

secondSample:

```
The test
   firstSample.Contains(’ "International" ’) = 1
succeeds since the word International is contained in this text
sample.
```

thirdSample:

```
die   ≅   Würfel
```

4.2.1 Single word patterns (patterns of the form <word>)

Single word patterns are the most basic pattern and they consist of a sequence of characters which are for a given language admissible in words. That sequence of characters is decorated by a leading and trailing double quote character, as in the following example:

```
' "International" '
```

NOTE 1 - The blank characters outside of double quote characters in the above example are not significant. They have been added simply to ensure readability of the example text.

NOTE 2 - A list of <key word>s that can be used in patterns is given in Subclause 5.3.2, "FT_Pattern Key Words". Although these <key word>s are shown in upper case in subsequent examples, methods that accept arguments containing these <key word>s are invariant to the case of these <key word>s.

A text value matches a word pattern if it contains at least one word which matches that pattern. Thus, the test:

```
firstSample.Contains(' "International" ') = 1
```

succeeds since the word *International* is contained in *firstSample*.

4.2.2 Single phrase patterns (patterns of the form <phrase>)

Single phrase patterns represent a sequence of words. Each such word is represented in the same way as the word in a single word pattern. Where needed by a given language an implementation-defined word separator is used. In the following example the word separator is a blank character. Like single word patterns, single phrase patterns are decorated by a leading and trailing double quote character, as in the following example:

```
' "International Standard" '
```

A text value matches a single phrase pattern if it contains at least one sequence of words, such that, for every word in that sequence, the *i*-th word of the sequence matches the *i*-th word of the phrase pattern. Thus, the test:

```
firstSample.Contains(’ "International Standard" ’) = 1
```

succeeds since the word sequence *International Standard* is contained in *firstSample*.

4.2.3 Patterns representing sets of single words

Patterns representing a set of words can be specified in one of the following ways:

4.2.3.1 Patterns of the form <word> with wild card characters

By using the wild card characters underscore (_) or percent (%) in any character position of a single word pattern a possibly unlimited number of single word patterns are effectively specified. For instance, in the following example:

```
’ "Standard_" ’
```

the underscore stands for any single character. Accordingly this pattern represents as many words (not all of them necessarily meaningful) as there are characters. When the percent wild card is used, the number of virtually represented single word patterns is infinite since this wild card character represents any sequence of characters (including the empty one). A text value matches such a pattern if it contains at least one word which matches one word out of the set of word patterns effectively represented by that pattern. Thus the test:

```
firstSample.Contains(’ "Standard%" ’) = 1
```

succeeds since the pattern matches the word *Standard* (note that the word *Standards* would also be matched). The test:

```
firstSample.Contains(’ "Standard_" ’) = 1
```

fails since there is no word in *firstSample* which starts with *Standard* and ends with some other character (such as "s").

4.2.3.2 Expansion facility patterns

Expansion facility patterns enable one to effectively generate a set composed of single word (and/or single phrase) patterns from a starting term which represents a single word such as *database* (note that a single phrase is also admissible as the starting term). Depending on the specific generation being specified the generated terms (i.e. single word or single phrase patterns) may be:

- terms which sound similar to the generating term,
- terms which are broader terms for the generating term,
- terms which are narrower terms for the generating term,
- terms which are synonyms of the generating term,
- terms which are preferred terms for the generating term,
- terms which are related to the generating term,
- terms which are top terms of the generating term.

A text value matches such a pattern if it contains at least one word which matches the single word patterns effectively represented by that pattern. Thus if the thesaurus *computer science* has been set up in such a way that *list* and *sequence* are synonyms to each other the test:

```
firstSample.Contains(’ THESAURUS "computer science"
                       EXPAND SYNONYM TERM OF "list" ’) = 1
```

(which uses a synonym expansion pattern) will succeed.

4.2.3.3 Enumeration of single word patterns (<token list> and certain <text literal list> patterns)

An enumeration of single word patterns consists of a comma separated list of single word patterns, as in the following example:

```
’ ( "Standard", "International", "method" ) ’
```

Any of the single word patterns may contain wild card characters, as in the following example:

```
’ ( "Standard", "International_", "method" ) ’
```

When wild card characters are used the number of words effectively represented by a pattern is larger than the number of its constituent single word patterns.

A text value matches such a <token list> pattern if it matches at least one of its constituent patterns. <token list> patterns can only be used as constituent patterns of <Proximity expansion> patterns.

4.2.3.4 Patterns representing sets of words with a common base reduced form (patterns of the form <stemmed word>)

Patterns of the form [STEMMED] FORM OF <word> are effectively treated as a set of <word> patterns, such that all elements of that set have the same base reduced form. For example:

```
STEMMED FORM OF ’ "mice" ’
```

will be treated as if

```
’ ( "mouse" , "mice" ’ )
```

had been specified.

Therefore a text value matches a <stemmed word> pattern if it matches the equivalent <token list> pattern. A text value matches a <stemmed word> pattern if it contains at least one word which when replaced by its base reduced form matches the base reduced form word pattern represented by that pattern. Thus, the test:

```
firstSample.Contains(’STEMMED FORM OF "Standards" ’) = 1
```

succeeds since the base reduced form of *Standards* is *Standard* which in turn is contained in *firstSample*.

4.2.4 Patterns formed by sets of single phrases

Patterns representing a set of phrases can be specified in one of the following ways:

4.2.4.1 Patterns of the form <phrase> with wild card characters

Within single phrase patterns wild card characters may be used as follows:

- within every constituent word representation any wild card character may be used as in the following example:

  ```
  ’ "International Standard%" ’
  ```

  Effectively a multitude of single phrase patterns is generated this way such that every possible combination of generated word representations and word representations without wild card characters (taking the proper word positions into account) are reflected by one of the resulting single phrase patterns.

- instead of a word representation a single percent (%) wild card character may be used as in the following example:

  ```
  ’ "this % Standard" ’
  ```

  Used this way the wild card character represents an arbitrary optional word. Thus the above pattern effectively represents a two word phrase, i.e.:

  ```
  ’ "this Standard" ’
  ```

  and an infinite number of three word phrases each having *this* and *Standard* as its first and last word, respectively.

The two styles of using wild card characters can be combined.

A text value matches a single phrase pattern with wild card characters if it matches at least one of the patterns effectively generated from that pattern. Thus the test:

```
firstSample.Contains(’ "this % Standard%" ’) = 1
```

succeeds since the pattern represents (among others) the word sequence *this International Standard* which is contained in *firstSample*.

4.2.4.2 Expansion facility patterns

Expansion facility patterns enable one to effectively generate a set composed of single phrase (and/or single word) patterns given a starting term which represents a phrase such as *data base* (note that a single word is also admissible as the starting term). Depending on the specific generation being specified the generated terms (i.e. single word or single phrase patterns) may be:

- terms which sound similar to the generating term,
- terms which are broader terms for the generating term,
- terms which are narrower terms for the generating term,
- terms which are synonyms of the generating term,
- terms which are preferred terms for the generating term,

- terms which are related to the generating term,
- terms which are top terms of the generating term.

A text value matches such a pattern if it contains at least one phrase which matches one of the single phrase patterns effectively represented by that pattern. Thus, if the thesaurus *computer science* has been set up in such a way that *rule of thumb* and *heuristics* are synonyms to each other then the test:

```
firstSample.Contains(’ THESAURUS "computer science"
                       EXPAND SYNONYM TERM OF "rule of thumb" ’) = 1
```

(which uses a synonym expansion pattern) will succeed.

4.2.4.3 Enumeration of single phrase patterns (certain <text literal list> patterns)

An enumeration of single phrase patterns consists of a comma separated list of single phrase patterns as in the following example:

```
’ ( "this % Standard", "International Standards" ) ’
```

If one of the constituent patterns contains wild card symbols then the number of phrase patterns effectively represented by this pattern is larger than the number of its constituent single phrase patterns.

A text value matches such a <text literal list> pattern if it matches at least one of its constituent patterns. <text literal list> patterns can only be used as constituent patterns of <context condition> patterns.

Note that a <text literal list> pattern may contain both single word patterns and single phrase patterns.

4.2.4.4 Patterns representing phrases with common base reduced forms (patterns of the form <stemmed phrase>)

Patterns of the form [STEMMED] FORM OF <phrase> are effectively treated as a set *SPP* of <phrase> patterns, which is constructed as follows:

Let *N* be the number of <phrase part representation>s *PPRi* simply contained in <stemmed phrase>. Let *Ni* be 1 (one) if *PPRi* represents an optional word. Otherwise, let *Ni* be the number of <phrasepart representation>s *WPij* that share the base reduced form of *PPRi*. Let *SPP* be such that *SPP* contains *N1* * ... * *NN* <phrase> patterns.

For a given <phrasepart representation> *i* there are only occurrences of *WPij* and every *WPij* occurs in that position. For example,

```
’ STEMMED FORM OF GERMAN "Internationale Standards" ’
```

is treated as

```
’ ( GERMAN "International Standard",
    GERMAN "Internationaler Standard",
    GERMAN "Internationales Standard",
    GERMAN "Internationalem Standard",
    GERMAN "Internationale Standard",
    GERMAN "Internationalen Standard",
    ...) ’
```

Therefore a text matches a <stemmed phrase> pattern if it matches one of the <phrase> patterns of *SPP*. This condition can be rephrased as: A text value matches a <stemmed phrase> pattern if it contains at least one phrase which when replacing each contained word by its base reduced form matches the transformed phrase pattern that is obtained from the <stemmed phrase> by replacing each contained <phrasepart representation> by one of its base reduced forms. Thus, the test:

```
firstSample.Contains(’ STEMMED FORM OF GERMAN
                       "Internationale Standards" ’) = 1
```

succeeds since the phrase *International Standards* is contained in *firstSample*.

4.2.5 Patterns specifying context conditions

Patterns for context conditions specify first a set of two or more subpatterns each effectively specifying a set of single word and/or single phrase patterns, and second a window inside of which all subpatterns must be matched.

4.2.5.1 <Proximity expansion> patterns

A <Proximity expansion> pattern is characterized by:

1. a first and second pattern each representing either a single word pattern or a set of single word patterns,
2. a window width which is specified by an integral number of structural units; predefined units are characters, words, sentences, and paragraphs.
3. an indication whether the matches are required to occur in order or not.

For reference purposes these constituents are marked in the example below:

```
’ ( "Standards", "International" )       -- first pattern
  NEAR "language"                        -- second pattern
  WITHIN 0                               -- number of units
  SENTENCES                              -- kind of unit
  IN ORDER                               -- matches to occur in order
)’
```

A text value matches a <Proximity expansion> pattern if all the conditions below are met:

1. The text value matches the first pattern.

2. Let *SubS* be a substring of the text value such that:
   - it starts with the first specified unit (character, word, etc.) that is or contains the first character of the portion that matches the first pattern,
   - its length is 1 (one) plus the number of units as specified in the <Proximity expansion> pattern.

3. *SubS* matches the second pattern.

4. If order has been specified then the portion matching the second pattern must not precede the portion matching the first pattern.

Thus the test:

```
firstSample.Contains(’ ("Standards", "International" )
                       NEAR "language"
                       WITHIN 0 SENTENCES
                       IN ORDER ’) = 1
```

succeeds since the first sentence of *firstSample* contains the words *International* and *language* such that the first one occurs prior to the second one. Note that the text matches the pattern although it does not contain the word *Standards*.

4.2.5.2 <context condition> patterns

A <context condition> pattern is characterized by:

1. two or more patterns *Si* each representing either a single word pattern, a set of single word patterns, a single phrase pattern, a set of single phrase patterns, or a set the elements which are single word and/or single phrase patterns.
2. a specification of a window which may be 1 (one) SENTENCE or 1 (one) PARAGRAPH wide.

Using this notation the example pattern of the previous Subclause is respecified as:

```
' ( "Standards", "International" ) -- first pattern
  IN SAME SENTENCE AS              -- window specification
  "language" '                     -- second pattern
```

<context condition> and <Proximity expansion> patterns complement each other. The <Proximity expansion> pattern is more flexible with respect to the window and order specifications but allows for two subpatterns only. In contrast, the <context condition> pattern is more restrictive with respect to the windows that can be specified but allows for more than two patterns to be matched within a given window.

A text value matches such a <context condition> pattern if it contains at least one sentence (paragraph) which matches every pattern *Si*.

4.2.6 Patterns involving Boolean operators

4.2.6.1 Patterns involving OR operators

Subpatterns of any form can be combined into new patterns by forming an "|" separated list of those subpatterns as in the following example:

```
' "Standard" | "International" | "language" '
```

A text value matches such a pattern if it matches at least one of its subpatterns. Thus the test:

```
secondSample.Contains(' "Standard" | "International" | "language" ')=1
```

succeeds since *secondSample* contains the word *International*.

4.2.6.2 Patterns involving AND operators

Subpatterns of the form <search factor> can be combined into new patterns by forming an "&" separated list of those subpatterns as in the following example:

```
' "Standard" & "International" & "language" '
```

A text value matches such a pattern if it matches all of its subpatterns at least once. Thus the test:

```
firstSample.Contains(' "Standard" & "International" & "language" ')=1
```

succeeds since *firstSample* contains each of the words *Standard*, *International*, and *language*.

4.2.6.3 Patterns involving negation

Patterns of the form <search primary> can be negated by prefixing them with NOT as in the following example:

```
’NOT "International Standard" ’
```

A text value matches such a pattern if the pattern prefixed by NOT does not match that text. Thus the test:

```
secondSample.Contains(’NOT "International Standard" ’) = 1
```

succeeds since the text *secondSample* does not contain the phrase *International Standard*.

4.2.6.4 Precedence of Boolean operators

Boolean operators take precedence over each other in the following order:

- NOT
- &
- |

Thus the test:

```
secondSample.Contains(’ NOT "International Standard" & "test" ’) = 1
```

succeeds since the text *secondSample* contains the word *test* but not the phrase *International Standard*.

The precedence can be overridden by putting parenthesis around subpatterns. For example if the previous test is changed by putting the pattern following NOT into parenthesis:

```
secondSample.Contains(’ NOT ("International Standard" & "test") ’)=1
```

then this test will succeed since the text *secondSample* does not simultaneously contain both the word *test* and the phrase *International Standard*.

4.2.7 Identification of *FullText* values which are pertinent to a given text

Patterns of the form IS ABOUT <phrase> allow for the identification of *FullText* values which in an implementation-defined way "is about" or is pertinent to <phrase>. Depending on the criteria an implementation applies when evaluating a pattern, the test

```
firstSample.Contains(’ IS ABOUT "International Standard on text
                                 search facilities" ’) = 1
```

will succeed or not.

4.3 Text ranking facilities

When a text value matches a certain pattern there is no indication on how well the text is characterized by that pattern. For instance a text matches the pattern:

```
’ ( "Standard", "International", "method" ) ’
```

if at least one of the pattern’s words (e.g. *Standard*) occurs at least once in that text. The method *Contains* used for performing the test gives no indication about the number of matching words or about the number of occurrences of these words in the text value.

For that end this part of ISO/IEC 13249 provides a *Rank* method for the *FullText* type. This method takes any pattern that can also be used for text identification as in the following example:

```
firstSample.Rank(′ ( "Standard", "International", "method" ) ′)
```

The *Rank* method returns a relevance measure as a non-negative floating point number where larger numbers mean a better match between the text value (*firstSample* in the above example) and the given pattern. The exact relationship between a text value and a pattern and the associated rank value is implementation-defined.

4.4 Language aspects

All values of the *FullText* type are associated with a specific language. The same effectively holds for patterns of the forms:

- <word>,
- <stemmed word>,
- <phrase>,
- <stemmed phrase>.

Language information is required for:

- recognition of word, sentence, and paragraph boundaries,
- expansion of words into sets of patterns composed of similarly sounding words,
- recognition of matches using base reduced forms,
- treatment of stop words,
- word normalization.

4.4.1 Multilingual texts and patterns

Patterns may be composed of subpatterns that are associated with different languages as in the following example:

```
' ENGLISH "die" & GERMAN "Würfel" '
```

Multilingual patterns can be very useful. In a setting with German as the default language the word *die* would be ignored as a stop word while it is not when marked as an English word.

In contrast text values of the *FullText* type are associated with a single language only. However, a conforming implementation is not required to enforce that the text contents of a *FullText* value is strictly monolingual. Instead any language specific processing of this text is performed according to the rules, conventions, and heuristics (which in turn are implementation-defined) of the language associated with the given *FullText* value.

When matching text values against patterns differing text and pattern languages may be appropriate as in the test:

```
thirdSample.Contains(' ENGLISH "die" & GERMAN "Würfel" ') = 1
```

This test will succeed if the language of *thirdSample* happens to be English and *Würfel* is accepted as a word according to structural criteria. The test will not succeed if the language is German and *die* is recognized as a stop word. Note that *die* (i.e. the feminine form of *the*) is most likely to be one of the implementation-defined stop words.

4.4.2 Treatment of stop words

Stop words are words that occur in text values at a probability which makes these words useless for text identification purposes. It primarily depends on the language whether some word (e.g. *die*) is to be treated as a stop word or not. Other factors such as the universe of discourse may also be taken into account.

The set of stop words for a given language is implementation-defined.

Stop words in patterns affect the identification of text values according to the following rules:

- A pattern of the form <word> or <stemmed word> must not represent a stop word unless it is part of a pattern of the form <phrase> or the form <token list>.

- If a pattern of the form <token list> or <text literal list> simply contains a subpattern of the form <word> or <stemmed word> that represents a stop word then it is implementation-defined whether the stop word is ignored or causes an error.

- It is implementation-defined whether the distance separating two words *W1* and *W2* in a pattern of the form <phase> or <stemmed phrase> is exactly or at most one more than the number of consecutive stop words between *W1* and *W2*. In the latter case, the stop words effectively behave like optional words.

- Let *P* be a pattern of the form <phrase > or <stemmed phrase> simply containing *n* <phrasepart representation>s some of which represent stop words. If stops words do not behave like optional words, then a text value *text* matches *P* if *text* contains a contiguous sequence of *n* words starting at some position (*j*+1) such that every (*j*+*i*)-th word of *text* is a stop word if the *i*-th word of *P* is a stop word, or otherwise is matched by the *i*-th word of *P*.

  Thus the test:

  ```
  firstSample.Contains(′ ( "sentence or paragraph" ) ′) = 1
  ```

  succeeds since *firstSample* contains the phrase *sentence and paragraph*.

  It is implementation-defined whether phrases are admissible that have a stop word as their first or last word or that consist of stop words only. If the latter case is supported then the test:

  ```
  firstSample.Contains(′ ( "this and that" ) ′) = 1
  ```

  would succeed if *firstSample* contained three consecutive stop words (which is actually not the case).

4.5 Word normalization

When evaluating *Rank* or *Contains* method invocations conforming implementations are allowed to normalize word patterns in an implementation-defined way provided that the words contained in the text values being tested or ranked by the *Rank* or *Contains* methods are effectively processed in the same way. For instance, the word pattern

```
' "Müller" '
```

may be replaced by

```
’ "Mueller" ’
```

This pattern will be matched by any text value containing at least one occurrence of *Müller* since this word is effectively replaced by *Mueller* before performing the test.

Normalization can possibly result in more matches than would be observed without normalization. In German texts the word *Mueller* (as opposed to *Müller*) has a low occurrence probability. If text values containing *Mueller* are to be identified then unwanted texts (i.e. those containing *Müller* but not *Mueller*) will eventually be identified as well.

4.6 Types and routines provided by this part of ISO/IEC 13249

The types and routines provided by this part of ISO/IEC 13249 are divided into two major Categories:

1. types and routines which are for public use,
2. definition oriented types and routines that are used to capture the semantics of the Category 1 types and routines, except for Rank methods (see Subclause 5.3.1, "FT_Pattern type").

4.6.1 Types and routines intended for public use

The following types and routines are intended for public use:

- *FullText* type with
  - methods *Language*,
  - methods *Contains*,
  - methods *Rank*,
  - methods *FullText*,
  - function *FullText_to_Character* to cast a *FullText* value into a character string,

- *FT_Pattern* type.

4.6.2 Types and routines for definition

All other types and routines that are not covered by Subclause 4.6.1, "Types and routines intended for public use" are used to specify the semantics of the Category 1 types and routines. Implementations conforming to this part of ISO/IEC 13249 do not need to provide these types or routines for public use.

4.6.3 Technique for defining the semantics of Category 1 Contains methods

As far as possible, types and routines of this part of ISO/IEC 13249 are defined by the facilities ISO/IEC 9075. For the Category 1 *Contains* methods this is done in an indirect way. Using the definitional facilities of ISO/IEC 9075, *Contains* methods are defined for the structural patterns of Clause 6. "Structured Search Pattern Types". For a given pattern accepted by a Category 1 Contains method, the meaning is defined in terms of an equivalent structural pattern. For example the following pattern:

```
'ENGLISH "die" & GERMAN "Würfel" '
```

is equivalent to the structural pattern:

```
NEW FT_Term(ARRAY[NEW FT_TextLiteral('die', 'ENGLISH'),
                  NEW FT_TextLiteral('Würfel', 'GERMAN')])
```

which in turn is captured by the fact that:

```
NEW FT_Term(ARRAY[NEW FT_TextLiteral(’die’, ’ENGLISH’),
NEW FT_TextLiteral('Würfel', 'GERMAN')]).StrctPattern_to_FT_Pattern()
```

returns a pattern which is equal except for some white space characters to the pattern:

```
' ENGLISH "die" & GERMAN "Würfel" '
```

under question. Finally the meaning of

```
thirdSample.Contains(' ENGLISH "die" & GERMAN "Würfel" ') = 1
```

is defined by the meaning of:

```
NEW FT_Term(ARRAY[NEW FT_TextLiteral('die', 'ENGLISH'),
      NEW FT_TextLiteral('Würfel', 'GERMAN')]).Contains(thirdSample)
```

5 Full-Text 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
   AS (
      Contents CHARACTER VARYING(FT_MaxTextLength),
      Language CHARACTER VARYING(FT_MaxLanguageLength)
         DEFAULT FT_DefaultLanguage
   )
   INSTANTIABLE
   NOT FINAL

   METHOD Contains(pattern FT_Pattern)
      RETURNS INTEGER
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      CALLED ON NULL INPUT,

   METHOD Contains(pattern CHARACTER VARYING(FT_MaxPatternLength))
      RETURNS INTEGER
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      CALLED ON NULL INPUT,

   METHOD Rank(pattern FT_Pattern)
      RETURNS DOUBLE PRECISION
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      CALLED ON NULL INPUT,

   METHOD Rank(pattern CHARACTER VARYING(FT_MaxPatternLength))
      RETURNS DOUBLE PRECISION
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      CALLED ON NULL INPUT,

   METHOD Tokenize()
      RETURNS FullText_Token ARRAY[FT_MaxArrayLength]
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      RETURNS NULL ON NULL INPUT,

   METHOD TokenizePosition(unit FullText_Token)
      RETURNS FT_TokenPosition ARRAY[FT_MaxArrayLength]
```

```
         LANGUAGE SQL
         DETERMINISTIC
         CONTAINS SQL
         RETURNS NULL ON NULL INPUT,

      METHOD Segmentize(unit FullText_Token)
         RETURNS FullText ARRAY[FT_MaxArrayLength]
         LANGUAGE SQL
         DETERMINISTIC
         CONTAINS SQL
         RETURNS NULL ON NULL INPUT,

      METHOD TokenizeAndStem()
         RETURNS FullText ARRAY[FT_MaxArrayLength]
         LANGUAGE SQL
         DETERMINISTIC
         CONTAINS SQL
         RETURNS NULL ON NULL INPUT,

      METHOD TokenizePositionAndStem()
         RETURNS FullText ARRAY[FT_MaxArrayLength]
         LANGUAGE SQL
         DETERMINISTIC
         CONTAINS SQL
         RETURNS NULL ON NULL INPUT,

      METHOD FullText(string CHARACTER VARYING(FT_MaxTextLength))
         RETURNS FullText
         SELF AS RESULT
         LANGUAGE SQL
         DETERMINISTIC
         CONTAINS SQL
         CALLED ON NULL INPUT,

      METHOD FullText
         (string CHARACTER VARYING(FT_MaxTextLength),
          Language CHARACTER VARYING(FT_MaxLanguageLength)
         )
         RETURNS FullText
         SELF AS RESULT
         LANGUAGE SQL
         DETERMINISTIC
         CONTAINS SQL
         CALLED ON NULL INPUT

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

Definitional Rules

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

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

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

4) *FT_MaxLanguageLength* is the implementation-defined maximum length for the character representation of a language specification.

5) *FT_DefaultLanguage* is an implementation-defined character string literal which denotes the implementation-defined default language. The length of *FT_DefaultLanguage* does not exceed *FT_MaxLanguageLength*.

Description

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

    a) an attribute *Language*,
    b) a method *Contains(FT_Pattern)*,
    c) a method *Contains(CHARACTER VARYING)*,
    d) a method *Rank(FT_Pattern)*,
    e) a method *Rank(CHARACTER VARYING)*,
    f) a method *FullText(CHARACTER VARYING)* to initialize a *FullText* value from a character string,
    g) a method *FullText(CHARACTER VARYING, CHARACTER VARYING)* to initialize a *FullText* value from a character string and a language specification,
    h) a function *FullText_to_Character(FullText)* to cast a *FullText* value into a character string.

2) The attribute *Contents* is not for public use. There are no GRANT statements granting EXECUTE privilege to the observer or mutator method for *Contents*.

5.1.2 Contains Methods

Purpose

Search a *FullText* value for a linear search pattern.

Definition

```
CREATE METHOD Contains(pattern FT_Pattern)
   RETURNS INTEGER
   FOR FullText
   BEGIN
      --
      -- !! See Description
      --
   END

CREATE METHOD Contains
   (pattern CHARACTER VARYING(FT_MaxPatternLength))
   RETURNS INTEGER
   FOR FullText
   RETURN SELF.Contains(CAST(pattern AS FT_Pattern))
```

Definitional Rules

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

Description

1) The method *Contains(FT_Pattern)* takes the following input parameters:

   a) an *FT_Pattern* value *pattern*.

2) The method *Contains(CHARACTER VARYING)* takes the following input parameters:

   a) a *CHARACTER VARYING* value *pattern*.

3) The result of the invocation *Contains(CHARACTER VARYING)* or *Contains(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*.

      NOTE 4 - <search expression> is defined in Subclause 5.3.1, “FT_Pattern Type”.

   b) If *pattern* contains a pattern that meets one of the following conditions, then it is implementation-defined whether an exception condition is raised: *SQL/MM Full-Text - invalid search expression*:

      i) A pattern of the form <word> or <stemmed word> specifies a stop word.

      ii) A pattern of the form <phrase> or <stemmed phrase> contains only stop words, or contains leading or trailing stop words.

      iii) A pattern of the form <text literal list> contains only stop words.

NOTE 3 - The subrules i), ii), and iii) reflect the behavior of the *Contains* methods for the types *FT_TextLiteral*, *FT_StemmedWord*, *FT_Phrase*, *FT_StemmedPhrase*, and *FT_Any*.

c) Otherwise:

Case:

i) If *SELF, SELF.Contents*, or *pattern* is the null value, then the null value.

ii) If the length of *SELF.Contents* is 0 (zero), then 0 (zero).

iii) Otherwise, let *s_pattern* be the structured pattern of type *FT_Expr*, such that

`pattern = s_pattern.StrctPattern_to_FT_Pattern()`

Then the result of

`SELF.Contains(pattern)`

is

Case:

A) 1 (one), if

`s_pattern.Contains(SELF)`

is <u>true</u>.

B) 0 (zero), if

`s_pattern.Contains(SELF)`

is <u>false</u>.

C) Otherwise, the null value.

4) The result of invocation of *Contains(FT_Pattern)* is invariant to the case of the <key word>s in *FT_Pattern*.

NOTE 5 - A list of *FT_Pattern* <key word>s is given in Subclause 5.3.2, "FT_Pattern Key Words".

5.1.3 Rank Methods

Purpose

Search a *FullText* value for a linear search pattern and give the relevance of the pattern.

Definition

```
CREATE METHOD Rank(pattern FT_Pattern)
   RETURNS DOUBLE PRECISION
   FOR FullText
   BEGIN
      --
      -- !! See Description
      --
   END

CREATE METHOD Rank
   (pattern CHARACTER VARYING(FT_MaxPatternLength))
   RETURNS DOUBLE PRECISION
   FOR FullText
   RETURN SELF.Rank(CAST(pattern AS FT_Pattern))
```

Definitional Rules

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

Description

1) The method *Rank(FT_Pattern)* takes the following input parameters:

   a) an *FT_Pattern* value *pattern*.

2) The method *Rank(CHARACTER VARYING)* takes the following input parameters:

   a) a *CHARACTER VARYING* value *pattern*.

3) The result of the invocation *Rank(CHARACTER VARYING)* or *Rank(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*.

      NOTE 6 - <search expression> is defined in Subclause 5.3.1, “FT_Pattern Type”.

   b) Otherwise:

      Case:

      i) If *SELF*, *SELF.Contents*, or *pattern* is the null value, the null value.

      ii) Otherwise, an implementation-dependent *DOUBLE PRECISION* value constrained by implementation-defined minimum and maximum values. The size of this value is an indication of how relevant *SELF* is for the given pattern.

4) The result of invocation of *Rank(FT_Pattern)* is invariant to the case of the <key word>s in *FT_Pattern*.

   NOTE 7 - A list of *FT_Pattern* <key word>s is given in Subclause 5.3.2, "FT_Pattern Key Words".

5.1.4 Tokenize Method

Purpose

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

Definition

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

Definitional Rules

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

Description

1) The method *Tokenize()* has no input parameters.

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

3) If the length of *SELF.Contents* is 0 (zero), then *Tokenize()* returns an empty array.

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

5.1.5 TokenizePosition Method

Purpose

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

Definition

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

Definitional Rules

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

Description

1) The method *TokenizePosition(FullText_Token)* takes the following input parameters:

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

2) The unit information supported is ’CHARACTERS’, ’WORDS’, ’SENTENCE’, ’SENTENCES’, ’PARAGRAPH’ and ’PARAGRAPHS’.

3) If the length of *SELF.Contents* is 0 (zero), then *TokenizePosition(FullText_Token)* returns an empty array.

4) *TokenizePosition(FullText_Token)* returns an array representing a set of *FT_TokenPosition* items with the attributes:

   a) A *FullText_Token* value *token* representing a normalized word occurring in *SELF*.
   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 identified by two *FT_TokenPosition* items. *corrVal* is zero for the distance units 'WORDS', 'SENTENCES' and 'PARAGRAPHS'; its value is implementation-defined for distance unit 'CHARACTERS'. 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 the 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*).

5) The result of *TokenizePosition(FullText_Token)* shall be the null value if:

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

6) It is implementation-defined whether no stop words of *SELF.Contents*, all stop words of *SELF.Contents*, or all stop words of *SELF.Contents* other than leading and trailing stop words are effectively included in the result of *SELF.TokenizePosition(FullText_Token).* If stop words are included, then it is implementation-defined how they are effectively represented, provided their representation is such that the result of comparing any two stop words is <u>true</u>.

7) Let *W1* and *W2* be two words contained in *SELF.Contents* and let *TLE1* and *TLE2* be the corresponding elements in the result of *TokenizePosition(FullText_Token)*. The distance between *W1* and *W2* shall be properly captured by *TLE1* and *TLE2* regardless of whether some word between *W1* and *W2* is a stop word and regardless of whether stop words are included in the result of *TokenizePosition(FullText_Token)* or not.

8) For all words adopted from *SELF* (whether they are stop words or not) their position relative to each other shall be properly reflected in the result of *TokenizePosition(FullText_Token)*.

9) Let *TLE* be the element of *SELF.TokenizePosition(FullText_Token)* with the lowest *Position* value. The value of *TLE.Position* shall be 1 (one). In particular this means that the result of *TokenizePosition(FullText_Token)* shall be the same for two values *SELF1* and *SELF2* which are different with one respect only: the *Contents* attribute of *SELF1* has leading stop words while *SELF2.Contents* has none.

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

5.1.6 Segmentize Method

Purpose

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

Definition

```
CREATE METHOD Segmentize(unit FullText_Token)
   RETURNS FullText ARRAY[FT_MaxArrayLength]
   FOR FullText
   BEGIN
      --
      -- !! See Description
      --
   END
```

Definitional Rules

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

Description

1) The method *Segmentize(FullText_Token)* takes the following input parameters:

   a) a *FullText_Token* value *unit*.

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

   NOTE 8 - If an implementation does not support the distance unit ’SENTENCE’ and ’PARAGRAPH’, then it is not required to support the method *Segmentize(FullText_Token).* If any of these distance units is supported, the method *Segmentize(FullText_Token)* shall effectively be supported with that distance unit.

3) If the length of *SELF.Contents* is 0 (zero), then *Segmentize(FullText_Token)* returns an empty array.

4) *Segmentize(FullText_Token)* returns an array of *FullText* values, 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) Further details of the relationship between input and output are implementation-defined. Though not enforced by this standard, it is intended that *Segmentize(FullText_Token)* reflects the language structure of the input text being processed. That language is denoted by *SELF.Language*.

5.1.7 TokenizeAndStem Method

Purpose

Convert a *FullText* value into a sequence of normalized and stem-reduced *FullText_Token* values.

Definition

```
CREATE METHOD TokenizeAndStem()
   RETURNS FullText_Token ARRAY[FT_MaxArrayLength]
   FOR FullText
   BEGIN
      --
      -- !! See Description
      --
   END
```

Definitional Rules

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

Description

1) The method *TokenizeAndStem()* has no input parameters.

2) *TokenizeAndStem()* returns an array representing a sequence of normalized and stem-reduced *FullText_Token* values. The result of *TokenizeAndStem()* is the null value if *SELF* or *SELF.Contents* is the null value.

3) If the length of *SELF.Contents* is 0 (zero), then *TokenizeandStem()* returns an empty array.

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

5.1.8 TokenizePositionAndStem Method

Purpose

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

Definition

```
CREATE METHOD TokenizePositionAndStem()
   RETURNS FT_TokenPosition ARRAY[FT_MaxArrayLength]
   FOR FullText
   BEGIN
      --
      -- !! See Description
      --
   END
```

Definitional Rules

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

Description

1) The method *TokenizePositionAndStem()* has no input parameters.

2) *TokenizePositionAndStem()* returns an array representing a set of *FT_TokenPosition* values with the attributes:

   a) A *FullText_Token* value *token* representing a word occurring in *SELF*; that word is represented in a normalized way and is reduced to its stemmed form.
   b) An INTEGER value *position* identifying the position of an occurrence of *token* in terms of words.
   c) An INTEGER value *corrVal* set to zero.

3) The result of *TokenizePositionAndStem()* shall be the null value if *SELF* or *SELF.Contents* is the null value.

4) If the length of *SELF.Contents* is 0 (zero), then *TokenizePositionAndStem()* returns an empty array.

5) It is implementation-defined whether no stop words of *SELF.Contents*, all stop words of *SELF.Contents,* or all stop words of *SELF.Contents* other than leading and trailing stop words are effectively included in the result of *SELF.TokenizePositionAndStem()*. If stop words are included, then it is implementation-defined how they are effectively represented, provided their representation is such that the result of comparing any two stop words is true.

6) Let *W1* and *W2* be two words contained in *SELF.Contents* and let *TLE1* and *TLE2* be the corresponding elements in the result of *TokenizePositionAndStem()*. The distance between *W1* and *W2* shall be properly captured by *TLE1* and *TLE2*, regardless of whether some word between *W1* and *W2* is a stop word and regardless of whether stop words are included in the result of *TokenizePositionAndStem()* or not.

7) Let *TLE* be the element of *SELF.TokenizePositionAndStem()* with the lowest *Position* value. The value of *TLE.Position* shall be 1 (one). In particular this means that the result of *TokenizePositionAndStem()* shall be the same for two values *SELF1* and *SELF2*, which are different with one respect only: the *Contents* attribute of *SELF1* has leading stop words while *SELF2.Contents* has none.

8) Further details of the relationship between input and output are implementation-defined. Though not enforced by this standard, it is intended that *TokenizePositionAndStem(FullText_Token)* reflects the language structure of the input text being processed. That language is denoted by *SELF.Language*.

5.1.9 FullText Methods

Purpose

Return a specified *FullText* value.

Definition

```
CREATE METHOD FullText(string CHARACTER VARYING(FT_MaxTextLength))
   RETURNS FullText
   FOR FullText
   RETURN SELF.Contents(string)

CREATE METHOD FullText
   (string CHARACTER VARYING(FT_MaxTextLength),
    Language CHARACTER VARYING(FT_MaxLanguageLength)
   )
   RETURNS FullText
   FOR FullText
   BEGIN
      DECLARE InvalidLanguage CONDITION FOR SQLSTATE ’XXF02";

      IF Language IS NULL OR
         Language = ’’    OR
         --
         -- if Language does not specify a supported language
         --
      THEN
         SIGNAL InvalidLanguage
            SET MESSAGE_TEXT = ’invalid language specification’;
      END IF;

      RETURN SELF.Contents(string).Language(Language);
   END
```

Definitional Rules

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

2) *FT_MaxLanguageLength* is the implementation-defined maximum length for the character representation of a language specification.

Description

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

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

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

   a) a *CHARACTER VARYING* value *string*,
   b) a *CHARACTER VARYING* value *Language*.

3) If the value of *Language* is the empty string or the null value or *Language* does not specify a supported language, then the method *FullText(CHARACTER VARYING, CHARACTER VARYING)* raises an exception condition: *SQL/MM Full-Text - invalid language specification*.

5.1.10 FullText_to_Character Function

Purpose

Return the character representation of a *FullText* value.

Definition

```
CREATE FUNCTION FullText_to_Character (text FullText)
   RETURNS CHARACTER VARYING(FT_MaxTextLength)
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT
   STATIC DISPATCH
   RETURN text.Contents
```

Definitional Rules

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

Description

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

   a) a *FullText* value *text*.

5.1.11 StrctPattern_to_FT_Pattern Function

Purpose

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

Definition

```
CREATE FUNCTION StrctPattern_to_FT_Pattern
   (woparray FT_WordOrPhrase ARRAY[FT_MaxArrayLength])
   RETURNS FT_Pattern
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   RETURNS NULL ON NULL INPUT
   STATIC DISPATCH
   BEGIN
      DECLARE result CHARACTER VARYING(FT_MaxPatternLength);
      DECLARE i INTEGER;

      SET i = 1;
      SET result = ’(’;
      WHILE i <= CARDINALITY(woparray) DO
         SET result = result
            || CAST(woparray[i].StrctPattern_to_FT_Pattern()
                  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_MaxPatternLength* is the implementation-dependent maximum length for the character representation of an *FT_Pattern* value.

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

Description

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

   a) an array *woparray* whose elements are *FT_WordOrPhrase* values.

2) *StrctPattern_to_FT_Pattern(FT_WordOrPhrase ARRAY)* returns an *FT_Pattern* value of the form <token list>.

3) If the input argument *woparray* is the null value, then *StrctPattern_to_FT_Pattern(FT_WordOrPhrase ARRAY)* returns the null value.

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
   AS (
      token FullText_Token,
      position INTEGER,
      corrVal INTEGER
   )
   INSTANTIABLE
   NOT FINAL
```

Description

1) The *FT_TokenPosition* type provides:

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

2) The purpose of the *FT_TokenPosition* attributes is described in Subclause 5.1.5, “TokenizePosition Method” 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)
   FINAL
```

Definitional Rules

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

Description

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

2) Values of *FT_Pattern* type are meant as input to the method *Contains(FT_Pattern)* of the *FullText* type.

   NOTE 9 - The *FullText* type is described in Subclause 5.1.1, "FullText Type".

3) Values 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>
   | <stemmed word>
   | <stemmed phrase>

<word> ::=
   [ <language specification> ] <double quote>
```

```
         <word representation> <double quote>
         [ <escape specification> ]

   <language specification> ::= !! See Description

   <double quote> ::=
      !! See Subclause 5.1, <SQL terminal character>,
      !!   in part 2 of ISO/IEC 9075

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

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

   <phrase> ::=
      [ <language specification> ] <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/IEC 9075

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

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

   <optional word representation> ::=  %

   <word separator> ::= !! See Description

   <stemmed word> ::=
      [ STEMMED ] FORM OF <word>

   <stemmed phrase> ::=
      [ STEMMED ] FORM OF <phrase>

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

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

```
<Proximity expansion> ::=
   <token list1> NEAR <token list2> WITHIN <distance> <unit> <order>

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

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

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

<left paren> ::= (

<right paren> ::= )

<comma> ::= ,

<word specification> ::=
     <word>
   | <stemmed word>

<distance> ::= <unsigned integer>

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

<unit> ::=
     CHARACTERS
   | WORDS
   | SENTENCES
   | PARAGRAPHS

<order> ::=
     ANY ORDER
   | IN ORDER

<Soundex expansion> ::=
   SOUNDS LIKE <word>

<Broader_Term expansion> ::=
   THESAURUS <thesaurus specification>
      EXPAND BROADER TERM OF <text literal>
      [FOR <thesaurus expansion count> { LEVEL | LEVELS }]

<thesaurus specification> ::=
   <double quote> <thesaurus name representation> <double quote>

<thesaurus name representation> ::= <thesaurus name character>...

<thesaurus name character> ::= !! See Description

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

<Narrower_Term expansion> ::=
   THESAURUS <thesaurus specification>
      EXPAND NARROWER TERM OF <text literal>
      [FOR <thesaurus expansion count> { LEVEL | LEVELS }]

<Synonym expansion> ::=
      THESAURUS <thesaurus specification>
         EXPAND SYNONYM TERM OF <text literal>

   <Preferred_Term expansion> ::=
      THESAURUS <thesaurus specification>
         EXPAND PREFERRED TERM OF <text literal>

   <Related_Term expansion> ::=
      THESAURUS <thesaurus specification>
         EXPAND RELATED TERM OF <text literal>

   <Top_Term expansion> ::=
      THESAURUS <thesaurus specification>
         EXPAND TOP TERM OF <text literal>

   <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>

   <about expansion> ::=
      IS ABOUT <word or phrase>

   <word or phrase> ::=
        <word>
      | <phrase>
```

NOTE 10 - A list of *FT_Pattern* <key word>s is given in Subclause 5.3.2, "FT_Pattern Key Words". Although these <key word>s are shown above in upper case, methods that accept *FT_Pattern* arguments are invariant to the case of these <key word>s.

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. Wild card characters '_' and '%' shall be among the admissible characters; however, a <word representation> shall contains at least one character that is not treated as a wild card 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*. *E* must be followed by either *E*, ‘%’, or ‘_’. If *WR* contains either a '%' or an '_' that is preceded by *E*, those characters represent a '%' or an '_', and not a wild card character. If an *E* is preceded by an *E*, the second *E* does not represent an <escape representation character>. *E* must be followed by either *E*. ‘%’, or ‘_’.

A <Broader_Term expansion>, <Narrower_Term expansion>, <Synonym_Term expansion>, <Preferred_Term expansion>, <Related_Term expansion>, or <Top_Term expansion> shall not contain a <stemmed word> or <stemmed phrase>.

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 11 - If a <phrasepart representation> *PPR* is simply contained in a <phrase> 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.

c) Each <word>, <stemmed word>, <phrase>, or <stemmed phrase> instance is associated with some language. This language is either explicitly or implicitly specified. The details of the <language specification>, as well as the default language is implementation-defined.

d) A <word> simply contained in a <Soundex expansion>, and a <text literal> simply contained in a <Broader_Term expansion>, <Narrower_Term expansion>, <Synonym expansion>, <Preferred_Term expansion>, <Related_Term expansion>, or <Top_Term expansion> shall not simply contain a <word representation character> that is treated as a wild card character, nor shall it simply contain an <escape specification>.

e) A <stemmed word> or <stemmed phrase> shall not simply contain a <word representation character> that is treated as a wild card character, nor shall it simply contain an <escape specification>.

f) <unit> and <context unit> instance denote document units. Document units are:

CHARACTERS
WORDS
SENTENCE
SENTENCES
PARAGRAPH
PARAGRAPHS

A conforming implementation must support at least one predefined document unit. Functionality depending on a certain document unit need only be supported if that document unit is supported. The document units supported are implementation-defined.

4) The characters <thesaurus name character> that can be used to construct thesaurus names are implementation-defined.

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

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

```
T.Contains(P)
```

is

Case:

i) 1 (one), if

`(T.Contains(SE) = 1) OR (T.Contains(ST) = 1)`

is true.

ii) 0 (zero), if

`(T.Contains(SE) = 1) OR (T.Contains(ST) = 1)`

is false.

iii) Otherwise, the null value.

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

`T.Contains(P)`

is

Case:

i) 1 (one), if

`(T.Contains(ST) = 1) AND (T.Contains(SF) = 1)`

is true.

ii) 0 (zero), if

`(T.Contains(ST) = 1) AND (T.Contains(SF) = 1)`

is false.

iii) Otherwise, the null value.

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

`T.Contains(P)`

is

Case:

i) 1 (one), if

`NOT T.Contains(SP)`

is true.

ii) 0 (zero), if

`NOT T.Contains(SP)`

is false.

iii) Otherwise, the null value.

d) If *P* is a <word> *W* or a <stemmed word> *W*, then:

i) If *W* does not immediately contain a <language specification>, then augment W with <language specification> denoting the default language.
ii) If *P* is <stemmed word> and *W* does not specify the optional key word STEMMED then augment *W* with the optional key word STEMMED.

Let *STL* be an *FT_TextLiteral* or an *FT_StemmedWord* value such that

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

The result of

```
T.Contains(W)
```

is

Case:

i) 1 (one), if

```
STL.Contains(T)
```

is <u>true</u>.

ii) 0 (zero), if

```
STL.Contains(T)
```

is <u>false</u>.

iii) Otherwise, the null value.

(i.e. *Contains* returns 1 (one) if *T* contains at least one token which matches *W* (if no <stemmed word> has been specified), or *T* contains at least one token the stem of which matches the stem of *W* (if <stemmed word> has been specified).)

NOTE 12 - A word pattern *W* may contain wild card 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>).

e) If *P* is a <phrase> *PHR* or a <stemmed phrase> *PHR*, then:

i) If *PHR* does not immediately contains a <language specification>, then augment *PHR* with <language specification> denoting the default language.
ii) If *P* is <stemmed phrase> and *PHR* does not specify the optional key word STEMMED then augment *PHR* with the optional key word STEMMED.

Let *SPP* be an *FT_Phrase* or an *FT_StemmedPhrase* value such that

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

then the result of

```
T.Contains(PHR)
```

is

Case:

i) 1 (one), if

```
SPP.Contains(T)
```

is <u>true</u>.

ii) 0 (zero), if

```
SPP.Contains(T)
```

is <u>false</u>.

iii) Otherwise, the null value.

(i.e. *Contains* returns 1 (one) if *T* contains a sequence of tokens which match *PHR*. The match condition details are given in Subclause 6.6, *FT_Phrase Type and Routines*, and in Subclause 6.7, *FT_StemmedPhrase Type and Routines*.)

NOTE 13 - A token of *PHR* may be composed of wild card characters only. If such a token consists of one or more ’%’s, then it denotes an optional word.

f) If *P* is a <Proximity expansion> *PFI*, then let *TL1* be <token list1> and *TL2* be <token list2>. Augment both *TL1* and *TL2* such that every occurrence of a <word>, <stemmed word>, <phrase>, or a <stemmed phrase> which does not specify a <language specification> is adorned by a <language specification> denoting the default language. Additionally augment both *TL1* and *TL2* such that every occurrence of <stemmed word> or <stemmed phrase> which does not specify the optional key word STEMMED is adorned by this missing optional key word.

Case:

i) If *TL1* is a <Broader_Term expansion>, let *SBT* be an *FT_BroaderTerm* value such that *TL1* is equal to *SBT.StrctPattern_to_FT_Pattern()*. 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 expansion>, let *SBT* be an *FT_BroaderTerm* value such that *TL2* is equal to *SBT.StrctPattern_to_FT_Pattern()*. 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 expansion>, let *SNT* be an *FT_NarrowerTerm* value such that *TL1* is equal to *SBT.StrctPattern_to_FT_Pattern()*. Replace *TL1* in *PFI* by the result of:

```
StrctPattern_to_FT_Pattern(GetNarrowerTerms(SNT.thesaurus,
   SNT.startingTerm, SNT.expansionCnt))
```

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

```
StrctPattern_to_FT_Pattern(GetNarrowerTerms(SNT.thesaurus,
   SNT.startingTerm, SNT.expansionCnt))
```

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

```
StrctPattern_to_FT_Pattern(GetSoundsSimilar(SPHT.spoken))
```

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

```
StrctPattern_to_FT_Pattern(GetSoundsSimilar(SPHT.spoken))
```

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

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

then the result of

```
T.Contains(PFI)
```

is

Case:

i) 1 (one), if

```
SPR.Contains(T)
```

is <u>true</u>.

ii) 0 (zero), if

```
SPR.Contains(T)
```

is <u>false</u>.

iii) Otherwise, the null value.

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 *Cai* be these <context argument>s. In every *Cai*, augment every occurrence of a <word>, <stemmed word>, <phrase>, or a <stemmed phrase> which does not specify a <language specification> by a <language specification> that denotes the default language. Additionally, in every *Cai*, augment every occurrence of <stemmed word> or <stemmed phrase> which does not specify the optional key word STEMMED by this missing optional key word. Let *CCDC* be the canonical form of *CCD*, which is obtained by replacing every *Cai* as follows:

Case:

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

```
(Cai)
```

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

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

iii) If *Cai* is a <Narrower_Term expansion>, let *SNT* be an *FT_NarrowerTerm* value such that *Cai* is equal to *SNT.StrctPattern_to_FT_Pattern()*. Replace *Cai* by the result of:

```
StrctPattern_to_FT_Pattern(GetNarrowerTerms(SNT.thesaurus,
        SNT.startingTerm, SNT.expansionCnt))
```

iv) If *Cai* is a <Synonym expansion>, let *SST* be an *FT_Synonym* value such that *Cai* is equal to *SST.StrctPattern_to_FT_Pattern()*. Replace *Cai* by the result of:

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

v) If *Cai* is a <Preferred_Term expansion>, let *SPT* be an *FT_PreferredTerm* value such that *Cai* is equal to *SPT.StrctPattern_to_FT_Pattern()*. Replace *Cai* by the result of:

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

vi) If *Cai* is a <Related_Term expansion>, let *SRT* be an *FT_RelatedTerm* value such that *Cai* is equal to *SRT.StrctPattern_to_FT_Pattern()*. Replace *Cai* by the result of:

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

vii) If *Cai* is a <Top_Term expansion>, let *STT* be an *FT_TopTerm* value such that *Cai* is equal to *STT.StrctPattern_to_FT_Pattern()*. Replace *Cai* by the result of:

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

viii) If *Cai* is a <Soundex expansion>, let *SPHT* be an *FT_Soundex* value such that *Cai* is equal to *SPHT.StrctPattern_to_FT_Pattern()*. Replace *Cai* by the result of:

```
StrctPattern_to_FT_Pattern(GetSoundsSimilar(SPHT.spoken))
```

ix) Otherwise, *Cai* is left unchanged.

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

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

Then the result of

```
T.Contains(CCDC)
```

is

Case:

i) 1 (one), if

```
SCR.Contains(T)
```

is true.

ii) 0 (zero), if

```
SCR.Contains(T)
```

is false.

iii) Otherwise, the null value.

h) If *P* is of the form <left paren> <search expression> <right paren>, augment *P* such that every occurrence of a <word>, <stemmed word>, <phrase>, or a <stemmed phrase> which does not specify a <language specification> is adorned by a <language specification> denoting the default language. Additionally augment *P* such that every occurrence of <stemmed word> or <stemmed phrase> which does not specify the optional key word STEMMED is adorned by this missing optional key word. Let *SPSE* be an *FT_ParExpr* value such that

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

Then the result of

```
T.Contains(P)
```

is

Case:

i) 1 (one), if

```
SPSE.Contains(T)
```

is <u>true</u>.

ii) 0 (zero), if

```
SPSE.Contains(T)
```

is <u>false</u>.

iii) Otherwise, the null value.

i) If *P* is a <Soundex expansion> *SFI*, augment *SFI* such that every occurrence of a <word> or a <phrase> which does not specify a <language specification> is adorned by a <language specification> denoting the default language. Let *SSO* be an *FT_Soundex* value such that

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

then the result of

```
T.Contains(SFI)
```

is

Case:

i) 1 (one), if

```
SSO.Contains(T)
```

is <u>true</u>.

ii) 0 (zero), if

```
SSO.Contains(T)
```

is <u>false</u>.

iii) Otherwise, the null value.

j) If *P* is a <Broader_Term expansion> *BFI*, augment *BFI* such that every occurrence of a <word> or a <phrase> which does not specify a <language specification> is adorned by a <language specification> denoting the default language. Let *SBT* be an *FT_BroaderTerm* value such that

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

then the result of

```
T.Contains(BFI)
```

is

Case:

i) 1 (one), if

```
SBT.Contains(T)
```

is <u>true</u>.

ii) 0 (zero), if

```
SBT.Contains(T)
```

is <u>false</u>.

iii) Otherwise, the null value.

NOTE 14 - If FOR <thesaurus expansion count> LEVELS has not been specified, then according to the specification of Subclause 6.10.3, "StrctPattern_to_FT_Pattern Method" and Subclause 6.10.5 "GetBroaderTerms Function" the expansion of <text literal> immediately contained in <Broader_Term expansion > is carried on until no further expansion term can be found.

k) If *P* is a <Narrower_Term expansion> *NFI*, augment *NFI* such that every occurrence of a <word> or a <phrase> which does not specify a <language specification> is adorned by a <language specification> denoting the default language. Let *SNT* be an *FT_NarrowerTerm* value such that

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

then the result of

```
T.Contains(NFI)
```

is

Case:

i) 1 (one), if

```
SNT.Contains(T)
```

is true.

ii) 0 (zero), if

```
SNT.Contains(T)
```

is false.

iii) Otherwise, the null value.

NOTE 15 - If FOR <thesaurus expansion count> LEVELS has not been specified, then according to the specification of Subclause 6.11.3, "StrctPattern_to_FT_Pattern Method" and Subclause 6.11.5 "GetNarrowerTerms Function" the expansion of <text literal> immediately contained in <Narrower_Term expansion > is carried on until no further expansion term can be found.

l) If *P* is a <Synonym expansion> *SYFI*, augment *SYFI* such that every occurrence of a <word> or a <phrase> which does not specify a <language specification> is adorned by a <language specification> denoting the default language. Let *SST* be an *FT_Synonym* value such that

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

then the result of

```
T.Contains(SYFI)
```

is

Case:

i) 1 (one), if

```
SST.Contains(T)
```

is true.

ii) 0 (zero), if

```
SST.Contains(T)
```

is false.

iii) Otherwise, the null value.

m) If *P* is a <Related_Term expansion> *RTFI*, augment *RTFI* such that every occurrence of a <word> or a <phrase> which does not specify a <language specification> is adorned by a <language specification> denoting the default language. Let *SRT* be an *FT_RelatedTerm* value such that

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

then the result of

```
T.Contains(RTFI)
```

is

Case:

  - i) 1 (one), if

    ```
    SRT.Contains(T)
    ```

    is true.

  - ii) 0 (zero), if

    ```
    SRT.Contains(T)
    ```

    is false.

  - iii) Otherwise, the null value.

n) If *P* is a <Preferred_Term expansion> *PTFI*, augment *PTFI* such that every occurrence of a <word> or a <phrase> which does not specify a <language specification> is adorned by a <language specification> denoting the default language. Let *SPT* be an *FT_PreferredTerm* value such that

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

then the result of

```
T.Contains(PTFI)
```

is

Case:

  - i) 1 (one), if

    ```
    SPT.Contains(T)
    ```

    is true.

  - ii) 0 (zero), if

    ```
    SPT.Contains(T)
    ```

    is false.

  - iii) Otherwise, the null value.

o) If *P* is a <Top_Term expansion> *TTFI*, augment *TTFI* such that every occurrence of a <word> or a <phrase> which does not specify a <language specification> is adorned by a <language specification> denoting the default language. Let *STT* be an *FT_TopTerm* value such that

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

then the result of

```
T.Contains(TTFI)
```

is

Case:

i) 1 (one), if

```
STT.Contains(T)
```

is true.

ii) 0 (zero), if

```
STT.Contains(T)
```

is false.

iii) Otherwise, the null value.

p) If *P* is an <about expansion> *IAFI*, augment *IAFI* such that the contained <word> or <phrase>, if it does not specify a <language specification>, is adorned by a <language specification> denoting the default language. Let *SIA* be an *FT_IsAbout* value such that

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

then the result of

```
T.Contains(IAFI)
```

is

Case:

i) 1 (one), if

```
SIA.Contains(T)
```

is true.

ii) 0 (zero), if

```
SIA.Contains(T)
```

is false.

iii) Otherwise, the null value.

5.3.2 FT_Pattern Key Words

Purpose

This subclause contains a list of all the <key word>s allowed in the *FT_Pattern* type. They are provided here for easy reference.

Definition

```
<key word> ::=
     ABOUT | AND | ANY | AS | BROADER | CHARACTERS | ESCAPE
   | EXPAND | FOR |FORM | FROM | IN | IS | LEVEL | LEVELS
   | LIKE | NARROWER | NEAR | NOT | OF | ORDER | PARAGRAPH
   | PARAGRAPHS | PREFERRED | PROXIMITY | RELATED | SAME
   | SENTENCE | SENTENCES | SOUNDS | STEMMED | SYNONYM | TERM
   | THESAURUS | TOP | WITHIN | WORDS
```

6 Structured Search Pattern Types

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

- FT_Any
- FT_Primary (not instantiable)
  - FT_WordOrPhrase (not instantiable)
    - FT_TextLiteral
      - FT_StemmedWord
    - FT_Phrase
      - FT_StemmedPhrase
  - 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
- FT_PhraseList

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 of *FT_WordOrPhrase* values and for testing whether at least one member of such a multiset occurs in a given *FullText* value.

Definition

```
CREATE TYPE FT_Any
   AS (
      Tokens FT_WordOrPhrase ARRAY[FT_MaxArrayLength]
   )
   INSTANTIABLE
   NOT FINAL

   METHOD Contains(text FullText)
      RETURNS BOOLEAN
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      CALLED ON NULL INPUT,

   METHOD FT_Any(tokens FT_WordOrPhrase ARRAY[FT_MaxArrayLength])
      RETURNS FT_Any
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      CALLED ON NULL INPUT
```

Definitional Rules

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

Description

1) The *FT_Any* type provides:

   a) an attribute *Tokens*,
   b) a method *Contains(FullText)*,
   c) a method *FT_Any(FT_WordOrPhrase ARRAY)*.

6.1.2 Contains Method

Purpose

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

Definition

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

      SET TokArray = text.Tokenize();

      IF TokArray IS NULL THEN
         SET lent = CAST(NULL AS INTEGER);
      ELSE
         SET lent = CARDINALITY(TokArray);
      END IF;
      IF SELF IS NULL THEN
         SET lena = CAST(NULL AS INTEGER);
      ELSEIF SELF.Tokens IS NULL THEN
         SET lena = CAST(NULL AS INTEGER);
      ELSE SET lena = CARDINALITY(SELF.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 Tab2(ind, wop) AS
            (VALUES(1, SELF.Tokens[1])
               UNION
             SELECT ind + 1, SELF.Tokens[ind + 1]
             FROM   Tab2
             WHERE  ind < lena
            ),
         Temp(BasI) AS
            (SELECT MAX(BasI)
             FROM (VALUES(1) UNION
               SELECT CASE ta.wop.Contains(text)
                        WHEN FALSE THEN 1
                        WHEN TRUE  THEN 3
                        ELSE            2
                     END
               FROM Tab2 ta) AS TT(BasI)
            )
         SELECT ARRAY[FALSE, UNKNOWN, TRUE][BasI] FROM Temp
         );
      END IF;
      RETURN result;
   END
```

Definitional Rules

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

Description

1) The method *Contains(FullText)* takes the following input parameters:

   a) a *FullText* item *text*.

2) If *SELF.Tokens* meets all of the following conditions, then it is implementation-defined whether an exception condition is raised: *SQL/MM Full-Text - invalid search expression*:

   i) Every contained pattern of the form <word> or <stemmed word> specifies a stop word.

   ii) Every contained pattern of the form <phrase> or <stemmed phrase> contains only stop words, or contains leading or trailing stop words.

3) *Contains(FullText)* returns:

   Case:

   a) false, if either *text.Tokenize()* or *SELF.Tokens* is empty, or for every element *B* of *SELF.Tokens*

      ```
      B.Contains(text)
      ```

      is false.

   b) true, if there exists one element *B* of *SELF.Tokens*, such that

      ```
      B.Contains(text)
      ```

      is true;

   c) Otherwise, unknown. In particular, this result is obtained if:

      i) Any of *text* or *text.Tokenize()* is the null value, and *SELF* or *SELF.Tokens* is the null value.

      ii) *text* or *text.Tokenize()* is the null value, but *SELF.Tokens* is an non-empty array.

      iii) *SELF* or *SELF.Tokens* is the null value, but *text.Tokenize()* is an non-empty array.

6.1.3 FT_Any Method

Purpose

Return a specified *FT_Any* value.

Definition

```
CREATE METHOD FT_Any
   (tokens FT_WordOrPhrase ARRAY[FT_MaxArrayLength])
   RETURNS FT_Any
   FOR FT_Any
   RETURN SELF.Tokens(tokens)
```

Definitional Rules

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

Description

1) The method FT*_Any(FT_WordOrPhrase ARRAY)* takes the following input parameters:

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

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
   AS (
      NOT_tag BOOLEAN
   )
   NOT INSTANTIABLE
   NOT FINAL

   METHOD Contains(text FullText)
      RETURNS BOOLEAN
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      CALLED ON NULL INPUT,

   METHOD StrctPattern_to_FT_Pattern()
      RETURNS FT_Pattern
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      CALLED ON NULL INPUT
```

Description

1) The *FT_Primary* type provides:

   a) an attribute *NOT_tag*,
   b) a method *Contains(FullText)*,
   c) a method *StrctPattern_to_Pattern()*.

2) Values of *FT_Primary* cannot be created. Only values of instantiable subtypes of *FT_Primary* can be created.

6.2.2 Contains Method

Purpose

Search a *FullText* value for an *FT_Primary* value.

Definition

```
CREATE METHOD Contains
   (text FullText)
   RETURNS BOOLEAN
   FOR FT_Primary
   RETURN TRUE
```

Description

1) The method *Contains(FullText)* takes the following input parameters:

   a) a *FullText* value *text*.

2) The method *Contains(FullText)* is a dummy method that will never be called since there are no *FT_Primary* values which are not values of a subtype of *FT_Primary*.

6.2.3 StrctPattern_to_FT_Pattern Method

Purpose

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

Definition

```
CREATE METHOD StrctPattern_to_FT_Pattern()
   RETURNS FT_Pattern
   FOR FT_Primary
   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 *FT_Pattern* value.

Description

1) The method *StrctPattern_to_FT_Pattern()* has no input parameters.

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

6.3 FT_WordOrPhrase Type and Routines

6.3.1 FT_WordOrPhrase Type

Purpose

The *FT_WordOrPhrase* type is the root type for the types *FT_TextLiteral* and *FT_Phrase*; it is not instantiable.

Definition

```
CREATE TYPE FT_WordOrPhrase
   UNDER FT_Primary
   NOT INSTANTIABLE
   NOT FINAL

   OVERRIDING METHOD Contains(text FullText)
      RETURNS BOOLEAN,

   OVERRIDING METHOD StrctPattern_to_FT_Pattern()
      RETURNS FT_Pattern,

   METHOD getWordArray()
      RETURNS FullText_Token ARRAY[FT_MaxArrayLength]
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      CALLED ON NULL INPUT
```

Definitional Rules

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

Description

1) The *FT_WordOrPhrase* type provides:

   a) a method *Contains(FullText)*,
   b) a method *StrctPattern_to_Pattern()*,
   c) a method *getWordArray()*.

2) *FT_WordOrPhrase* values cannot be created. Only values of the subtypes of *FT_WordOrPhrase* can be created.

6.3.2 Contains Method

Purpose

Search a *FullText* value for an *FT_WordOrPhrase* value.

Definition

```
CREATE METHOD Contains
   (text FullText)
   RETURNS BOOLEAN
   FOR FT_WordOrPhrase
   RETURN TRUE
```

Description

1) The method *Contains(FullText)* takes the following input parameters:

   a) a *FullText* value *text.*

2) The method *Contains(FullText)* is a dummy method that will never be called since there are no *FT_WordOrPhrase* values which are not values of a subtype of *FT_WordOrPhrase*.

6.3.3 StrctPattern_to_FT_Pattern Method

Purpose

Convert an *FT_WordOrPhrase* value to an *FT_Pattern* value.

Definition

```
CREATE METHOD StrctPattern_to_FT_Pattern()
   RETURNS FT_Pattern
   FOR FT_WordOrPhrase
   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 *FT_Pattern* value.

Description

1) The method *StrctPattern_to_FT_Pattern()* has no input parameters.

2) The method *StrctPattern_to_FT_Pattern()* is a dummy method that will never be called since there are no *FT_WordOrPhrase* values which are not values of a subtype of *FT_WordOrPhrase*.

6.3.4 getWordArray Method

Purpose

Generate an array representation while preserving ordering from an *FT_WordOrPhrase* value where each array element contains a *FullText_Token* value representing a word.

Definition

```
CREATE METHOD getWordArray()
   RETURNS FullText_Token ARRAY[FT_MaxArrayLength]
   FOR FT_WordOrPhrase
   RETURN CAST(ARRAY[] AS FullText_Token ARRAY[FT_MaxArrayLength])
```

Definitional Rules

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

Description

1) The method *getWordArray()* has no input parameters.

2) The method *getWordArray()* is a dummy method that will never be called since there are no *FT_WordOrPhrase* values which are not values of a subtype of *FT_WordOrPhrase*.

6.4 FT_TextLiteral Type and Routines

6.4.1 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_WordOrPhrase
   AS (
      LitPart FullText_Token,
      Language CHARACTER VARYING(FT_MaxLanguageLength),
      EscapeSpec CHARACTER(1)
   )
   INSTANTIABLE
   NOT FINAL

   OVERRIDING METHOD Contains(text FullText)
      RETURNS BOOLEAN,

   OVERRIDING METHOD StrctPattern_to_FT_Pattern()
      RETURNS FT_Pattern,

   METHOD matches(tok FullText_Token)
      RETURNS BOOLEAN
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      CALLED ON NULL INPUT,

   METHOD Tokenize()
      RETURNS FT_TextLiteral
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      CALLED ON NULL INPUT,

   METHOD FT_TextLiteral
      (w FullText_Token,
       Language CHARACTER VARYING(FT_MaxLanguageLength))
      RETURNS FT_TextLiteral
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      CALLED ON NULL INPUT,

   METHOD FT_TextLiteral
      (w FullText_Token
       Language CHARACTER VARYING(FT_MaxLanguageLength),
       EscapeChar CHARACTER(1))
      RETURNS FT_TextLiteral
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      CALLED ON NULL INPUT,
      OVERRIDING METHOD getWordArray()
         RETURNS FullText_Token ARRAY[FT_MaxArrayLength]
```

Definitional Rules

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

2) *FT_MaxLanguageLength* is the implementation-defined maximum length for the character representation of a language specification.

Description

1) The *FT_TextLiteral* type provides:

   a) an attribute *LitPart*,
   b) an attribute *Language*,
   c) an attribute *EscapeSpec*,
   d) a method *Contains(FullText)*,
   e) a method *StrctPattern_to_FT_Pattern()*,
   f) a method *matches(FullText_Token)*,
   g) a method *Tokenize()*,
   h) a method *getWordArray()*,
   i) a method *FT_TextLiteral(FullText_Token, CHARACTER VARYING)* and a method *FT_TextLiteral(FullText_Token, CHARACTER VARYING, CHARACTER)*,
   j) a function *EliminateDQS(FullText_Token)*,
   k) a function *InsertDQS(FullText_Token)*.

6.4.2 Contains Method

Purpose

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

Definition

```
CREATE METHOD Contains
   (text FullText)
   RETURNS BOOLEAN
   FOR FT_TextLiteral
   BEGIN
      DECLARE result BOOLEAN;

      IF text.Tokenize() IS NULL THEN
         RETURN UNKNOWN;
      END IF;
      IF CARDINALITY(text.Tokenize()) = 0 THEN
         SET result = FALSE;
      ELSE
         SET result = (WITH RECURSIVE tempTab(pos, token) AS
            (VALUES(1, text.Tokenize()[1])
                  UNION
             SELECT tt.pos + 1, text.Tokenize()[tt.pos + 1]
             FROM   tempTab tt
             WHERE  tt.pos < CARDINALITY(text.Tokenize())
            ),
         Temp(BasI) AS
            (SELECT MAX(BasI)
             FROM (VALUES(1) UNION
               SELECT CASE SELF.Tokenize().matches(tt.token)
                         WHEN FALSE THEN 1
                         WHEN TRUE  THEN 3
                         ELSE            2
                      END
               FROM TempTab tt) AS TT(BasI)
            )
         SELECT ARRAY[FALSE, UNKNOWN, TRUE][BasI] FROM Temp
         );
      END IF;
      RETURN (SELF.NOT_tag = result);
   END
```

Description

1) The method *Contains(FullText)* takes the following input parameters:

   a) a *FullText* item *text*.

2) Let *TL* be the result of the invocation of *text.Tokenize()* and *TLE* be elements of *TL*, normalized in an implementation-defined way, and with leading and trailing blanks removed. Let *T* be *SELF.LitPart*, normalized in an implementation-defined way and with leading and trailing blanks removed. If *SELF.EscapeSpec* is the null value, let *TT* be *T*; otherwise, let *TT* be *T* ESCAPE *SELF.EscapeSpec*.

   a) Case:

      i) If *T* contains a stop word, an exception condition is raised: *SQL/MM Full-Text - invalid search expression.*

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

iii) 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.

iv) 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.

v) Otherwise, let *R* be unknown.

b) *Contains(FullText)* returns:

Case:

i) unknown, if *SELF.NOT_tag* is the null value.

ii) *R*, if *SELF.NOT_tag* is true.

iii) Otherwise, NOT *R*.

6.4.3 StrctPattern_to_FT_Pattern Method

Purpose

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

Definition

```
CREATE METHOD StrctPattern_to_FT_Pattern()
   RETURNS FT_Pattern
   FOR FT_TextLiteral
   BEGIN
      DECLARE result CHARACTER VARYING(FT_MaxPatternLength);

      SET result = SELF.Language || ’ "’
         || TRIM(BOTH ’ ’ FROM InsertDQS(SELF.LitPart))
         || CASE WHEN SELF.EscapeSpec IS NULL THEN
               ’"’
            ELSE
               ’" ESCAPE "’ || SELF.EscapeSpec || ’"’
            END;
      IF SELF.NOT_tag IS UNKNOWN THEN
         SET result = NULL;
      ELSEIF NOT SELF.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 *FT_Pattern* value.

Description

1) The method *StrctPattern_to_FT_Pattern()* has no input parameters.

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

3) In the course of initializing an *FT_Pattern* value, <double quote>s appearing in *SELF.LitPart* 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 *SELF* is the null value or *SELF.NOT_tag* is unknown, then the result is the null value.

6.4.4 matches Method

Purpose

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

Definition

```
CREATE METHOD matches
   (tok FullText_Token)
   RETURNS BOOLEAN
   FOR FT_TextLiteral
   RETURN (
      CASE WHEN SELF.EscapeSpec IS NULL THEN
         TRIM(BOTH ’ ’ FROM tok) LIKE
            TRIM(BOTH ’ ’ FROM SELF.LitPart)
      ELSE
         TRIM(BOTH ’ ’ FROM tok) LIKE
            TRIM(BOTH ’ ’ FROM SELF.LitPart) ESCAPE SELF.EscapeSpec
      END
   )
```

Description

1) The method *matches(FullText_Token)* takes the following input parameters:

   a) a *FullText_Token* item *tok*.

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

6.4.5 Tokenize Method

Purpose

Normalize the *LitPart* attribute of an *FT_TextLiteral* value.

Definition

```
CREATE METHOD Tokenize()
   RETURNS FT_TextLiteral
   FOR FT_TextLiteral
   BEGIN
      --
      -- !! See Description
      --
   END
```

Description

1) The method *Tokenize()* has no input parameters.

2) *Tokenize()* normalizes *SELF.LitPart* in an implementation-defined way.

6.4.6 getWordArray Method

Purpose

Return a one element *FullText_Token* array from an *FT_TextLiteral* value representing a single word.

Definition

```
CREATE METHOD getWordArray()
   RETURNS FullText_Token ARRAY[FT_MaxArrayLength]
   FOR FT_TextLiteral
   RETURN ARRAY[TRIM(BOTH ’ ’ FROM SELF.LitPart)]
```

Definitional Rules

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

Description

1) The method *getWordArray()* has no input parameters.

2) The method *getWordArray()* returns *SELF.LitPart* as a one element *FullText_Token* array.

6.4.7 FT_TextLiteral Methods

Purpose

Return a specified *FT_TextLiteral* value.

Definition

```
CREATE METHOD FT_TextLiteral
   (w FullText_Token,
    Language CHARACTER VARYING(FT_MaxLanguageLength))
   RETURNS FT_TextLiteral
   FOR FT_TextLiteral
   RETURN SELF.LitPart(EliminateDQS(w)).
      Language(Language).NOT_tag(TRUE)

CREATE METHOD FT_TextLiteral
   (w FullText_Token,
   Language CHARACTER VARYING(FT_MaxLanguageLength),
    EscapeChar CHARACTER(1))
   RETURNS FT_TextLiteral
   FOR FT_TextLiteral
   RETURN NEW FT_TextLiteral(w, Language).EscapeSpec(EscapeChar)
```

Description

1) The method *FT_TextLiteral(FullText_Token, CHARACTER VARYING)* takes the following input parameters:

   a) a *FullText_Token* value *w*,
   b) a *CHARACTER VARYING* value *Language*.

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

   a) a *FullText_Token* value *w*,
   b) a *CHARACTER VARYING* value *Language*,
   c) a *CHARACTER* value *EscapeChar*.

3) In the process of initializing an *FT_TextLiteral* value, 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>.

6.4.8 EliminateDQS Function

Purpose

Eliminate a double quote symbol from a *FullText_Token* value.

Definition

```
CREATE FUNCTION EliminateDQS
   (w FullText_Token)
   RETURNS FullText_Token
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT
   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>.

6.4.9 InsertDQS Function

Purpose

Insert a double quote symbol in a *FullText_Token* value.

Definition

```
CREATE FUNCTION InsertDQS
   (w FullText_Token)
   RETURNS CHARACTER VARYING(FT_MaxPatternLength)
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT
   BEGIN
      --
      -- !! See Description
      --
   END
```

Definitional Rules

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

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>.

6.5 FT_StemmedWord Type and Routines

6.5.1 FT_StemmedWord Type

Purpose

The *FT_StemmedWord* type provides facilities for the construction of stemmed word search patterns and for searching of occurrences of stemmed words in text.

Definition

```
CREATE TYPE FT_StemmedWord
   UNDER FT_TextLiteral
   INSTANTIABLE
   NOT FINAL

   OVERRIDING METHOD Contains(text FullText)
      RETURNS BOOLEAN,

   OVERRIDING METHOD StrctPattern_to_FT_Pattern()
      RETURNS FT_Pattern,

   METHOD TokenizeAndStem()
      RETURNS FT_TextLiteral
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      CALLED ON NULL INPUT,

   METHOD FT_StemmedWord
      (sw FullText_Token,
       Language CHARACTER VARYING(FT_MaxLanguageLength))
      RETURNS FT_StemmedWord
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      CALLED ON NULL INPUT

   METHOD FT_StemmedWord
      (sw FullText_Token,
       Language CHARACTER VARYING(FT_MaxLanguageLength),
       EscapeChar CHARACTER(1))
      RETURNS FT_StemmedWord
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      CALLED ON NULL INPUT
```

Definitional Rules

1) *FT_MaxLanguageLength* is the implementation-defined maximum length for the character representation of a language specification.

Description

1) The *FT_StemmedWord* type provides:

   a) a method *Contains(FullText),*

b) a method *StrctPattern_to_FT_Pattern(),*
c) a method *TokenizeAndStem(),*
d) a method *FT_StemmedWord(FullText_Token, CHARACTER VARYING)* and a method *FT_StemmedWord(FullText_Token, CHARACTER VARYING, CHARACTER).*

6.5.2 Contains Method

Purpose

Search a *FullText* value for an *FT_StemmedWord* value.

Definition

```
CREATE METHOD Contains
   (text FullText)
   RETURNS BOOLEAN
   FOR FT_StemmedWord
   BEGIN
      DECLARE result BOOLEAN;

      IF text.TokenizeAndStem() IS NULL THEN
         RETURN UNKNOWN;
      END IF;
      IF CARDINALITY(text.TokenizeAndStem()) = 0 THEN
         SET result = FALSE;
      ELSE
         SET result = (WITH RECURSIVE tempTab(pos, token) AS
            (VALUES(1, text.TokenizeAndStem()[1])
               UNION
            SELECT tt.pos + 1, text.TokenizeAndStem()[tt.pos + 1]
            FROM   tempTab tt
            WHERE  tt.pos < CARDINALITY(text.TokenizeAndStem())
            ),
         Temp(BasI) AS
            (SELECT MAX(BasI)
             FROM (VALUES(1) UNION
               SELECT CASE SELF.TokenizeAndStem().matches(tt.token)
                       WHEN FALSE THEN 1
                       WHEN TRUE  THEN 3
                       ELSE            2
                    END
               FROM TempTab tt) AS TT(BasI)
            )
         SELECT ARRAY[FALSE, UNKNOWN, TRUE][BasI] FROM Temp
         );
      END IF;
      RETURN (SELF.NOT_tag = result);
   END
```

Description

1) The method *Contains*(*FullText*) takes the following input parameters:

   a) a *FullText* item *text*.

2) Let *TL* be the result of the invocation of *text.TokenizeAndStem()*. Let *TLE* be elements of *TL*, normalized and reduced to stems in an implementation-defined way, and with leading and trailing blanks removed. Let *T* be *SELF.LitPart*, normalized and reduced to stems in an implementation-defined way, and with leading and trailing blanks removed. If *SELF.EscapeSpec* is the null value, let *TT* be *T*; otherwise, let *TT* be *T* ESCAPE *SELF.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*, then let *R* be false.

iii) If *TL* contains at least one element *TLE*, such that

TLE LIKE *TT*

is true, then let *R* be true.

iv) Otherwise, let *R* be unknown.

b) *Contains(FullText)* returns:

Case:

i) unknown, if *SELF.NOT_tag* is the null value.

ii) *R*, if *SELF.NOT_tag* is true.

iii) Otherwise, NOT *R*.

6.5.3 StrctPattern_to_FT_Pattern Method

Purpose

Convert an *FT_StemmedWord* value to an *FT_Pattern* value.

Definition

```
CREATE METHOD StrctPattern_to_FT_Pattern()
   RETURNS FT_Pattern
   FOR FT_StemmedWord
   BEGIN
      DECLARE result CHARACTER VARYING(FT_MaxPatternLength);

      SET result = ’STEMMED FORM OF’ || SELF.Language
         ||’ "’ || TRIM(BOTH ’ ’ FROM InsertDQS(SELF.LitPart))
         || CASE WHEN SELF.EscapeSpec IS NULL THEN
               ’"’
            ELSE
               ’" ESCAPE "’ || SELF.EscapeSpec || ’"’
            END;
      IF SELF.NOT_tag IS UNKNOWN THEN
         SET result = NULL;
      ELSEIF NOT SELF.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 *FT_Pattern* value.

Description

1) The method *StrctPattern_to_FT_Pattern(FT_StemmedWord)* has no input parameters.

2) *StrctPattern_to_FT_Pattern()* converts an *FT_StemmedWord* value into an *FT_Pattern* value of the form <stemmed word> or of the form NOT <stemmed word>.

3) In the course of initializing an *FT_Pattern* value, <double quote>s appearing in *SELF.LitPart* 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 *SELF* is the null value or *SELF.NOT_tag* is unknown, then the result is the null value.

6.5.4 TokenizeAndStem Method

Purpose

Normalize and stem-reduce the *LitPart* attribute of an *FT_StemmedWord* value.

Definition

```
CREATE METHOD TokenizeAndStem()
   RETURNS FT_TextLiteral
   FOR FT_StemmedWord
   BEGIN
      --
      -- !! See Description
      --
   END
```

Description

1) The method *TokenizeAndStem()* has no input parameters.

2) *TokenizeAndStem()* normalizes and stem-reduces *SELF.LitPart* in an implementation-defined way.

6.5.5 FT_StemmedWord Methods

Purpose

Return a specified *FT_StemmedWord* value.

Definition

```
CREATE METHOD FT_StemmedWord
   (sw FullText_Token,
    Language CHARACTER VARYING(FT_MaxLanguageLength))
   RETURNS FT_StemmedWord
   FOR FT_StemmedWord
   RETURN SELF.LitPart(EliminateDQS(sw)).
      Language(Language).NOT_Tag(TRUE)

CREATE METHOD FT_StemmedWord
   (sw FullText_Token,
    Language CHARACTER VARYING(FT_MaxLanguageLength),
    EscapeChar CHARACTER(1))
   RETURNS FT_StemmedWord
   FOR FT_StemmedWord
   RETURN NEW FT_StemmedWord(sw, Language).EscapeSpec(EscapeChar)
```

Definitional Rules

1) *FT_MaxLanguageLength* is the implementation-defined maximum length for the character representation of a language specification.

Description

1) The method *FT_StemmedWord*(*FullText_Token*, *CHARACTER VARYING*) takes the following input parameters:

   a) a *FullText_Token* value *sw*,
   b) a *CHARACTER VARYING* value *Language*.

2) The method *FT_StemmedWord(FullText_Token, CHARACTER VARYING, CHARACTER)* takes the following input parameters:

   a) a *FullText_Token* value *sw*,
   b) a *CHARACTER VARYING* value *Language*,
   c) a *CHARACTER* value *EscapeChar*.

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

6.6 FT_Phrase Type and Routines

6.6.1 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_WordOrPhrase
   AS (
      PhrasePart FullText_Token ARRAY[FT_MaxArrayLength],
      Language CHARACTER VARYING(FT_MaxLanguageLength),
      EscapeSpec CHARACTER(1)
   )
   INSTANTIABLE
   NOT FINAL

   OVERRIDING METHOD Contains(text FullText)
      RETURNS BOOLEAN,

   OVERRIDING METHOD StrctPattern_to_FT_Pattern()
      RETURNS FT_Pattern,

   OVERRIDING METHOD getWordArray()
      RETURNS FullText_Token ARRAY[FT_MaxArrayLength],

   METHOD TokenizePosition()
      RETURNS FT_TokenPosition ARRAY[FT_MaxArrayLength]
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      CALLED ON NULL INPUT,

   METHOD FT_Phrase
      (wl FullText_Token ARRAY[FT_MaxArrayLength],
       Language CHARACTER VARYING(FT_MaxLanguageLength))
      RETURNS FT_Phrase
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      CALLED ON NULL INPUT,

   METHOD FT_Phrase
      (wl FullText_Token ARRAY[FT_MaxArrayLength],
       Language CHARACTER VARYING(FT_MaxLanguageLength),
       EscapeChar CHARACTER(1))
      RETURNS FT_Phrase
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      CALLED ON NULL INPUT
```

Definitional Rules

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

2) *FT_MaxLanguageLength* is the implementation-defined maximum length for the character representation of a language specification.

Description

1) The *FT_Phrase* type provides:

   a) an attribute *PhrasePart*,
   b) an attribute *Language*,
   c) an attribute *EscapeSpec*,
   e) a method *Contains(FullText),*
   f) a method *StrctPattern_to_FT_Pattern()*,
   g) a method *getWordArray(),*
   h) a method *TokenizePosition()*,
   i) a method *FT_Phrase(FullText_Token ARRAY, CHARACTER VARYING)* and a method *FT_Phrase(FullText_Token ARRAY, CHARACTER VARYING, CHARACTER)*,
   j) a function *matches(FT_TokenPosition ARRAY, INTEGER, INTEGER, FT_TokenPosition ARRAY, INTEGER, INTEGER, CHARACTER, CHARACTER VARYING),*
   k) a function *prune(FT_TokenPosition ARRAY, INTEGER, INTEGER).*

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

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

   A token may be the null value.

   NOTE 16 - *FT_Phrase* values 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 ’%’ wild card characters, then it denotes an optional word.

6.6.2 Contains Method

Purpose

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

Definition

```
CREATE METHOD Contains
   (text FullText)
   RETURNS BOOLEAN
   FOR FT_Phrase
   BEGIN
      DECLARE tokarray FT_TokenPosition ARRAY[FT_MaxArrayLength];
      DECLARE result BOOLEAN;
      DECLARE lent INTEGER;
      DECLARE tlen INTEGER;
      DECLARE lenp INTEGER;
      DECLARE plen INTEGER;
      DECLARE canonicphr FT_TokenPosition ARRAY[FT_MaxArrayLength];
      DECLARE nmsk INTEGER;
      DECLARE i    INTEGER;

      SET tokarray = text.TokenizePosition('WORDS');
      IF tokarray IS NULL THEN
         RETURN UNKNOWN;
      END IF;
      SET lent = CARDINALITY(tokarray);
      SET canonicphr = SELF.TokenizePosition();
      IF (SELF IS NULL OR canonicphr IS NULL) AND
            lent <> 0 THEN
         RETURN UNKNOWN;
      END IF;
      SET lenp = CARDINALITY(canonicphr);
      SET nmsk = 0;
      SET i    = 1;
      --------------------------------------------
      -- find tokens representing an optional word
      --------------------------------------------
      L1: WHILE (i <= lenp) DO
         IF canonicphr[i].token SIMILAR '$%+' ESCAPE '$' THEN
            SET nmsk = nmsk + 1;
         END IF;
         SET i = i + 1;
      END WHILE L1;
      IF lent = 0 THEN
         RETURN (FALSE = SELF.NOT_tag);
      END IF;
      IF lenp = 0 THEN
         RETURN (TRUE = SELF.NOT_tag);
      END IF;

      SET tlen = tokarray[lent].position;
      SET plen = canonicphr[lenp].position;

      IF tlen < plen - nmsk THEN
         RETURN (FALSE = SELF.NOT_tag);
      END IF;
      IF plen - nmsk = 0 THEN
         RETURN (TRUE = SELF.NOT_tag);
      END IF;

      SET result = (WITH RECURSIVE textrange(i) AS
            (VALUES (1)
               UNION
             SELECT i + 1
             FROM   textrange
             WHERE  i < lent
            ),
            Temp(BasI) AS
               (SELECT MAX(BasI)
                FROM (VALUES(1) UNION
                  SELECT
                     CASE tokarray[i].position <=
                                 tlen + 1 - (plen - nmsk)
                      AND
                        matches(tokarray, i, lent, canonicphr, 1, lenp,
                              SELF.EscapeSpec, SELF.Language)
                        WHEN FALSE THEN 1
                        WHEN TRUE  THEN 3
                        ELSE            2
                     END
                  FROM textrange AS tr(i)) AS TT(BasI)
               )
            SELECT ARRAY[FALSE, UNKNOWN, TRUE][BasI] FROM Temp
            );
         RETURN (SELF.NOT_tag = result);
      END
```

Definitional Rules

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

Description

1) The method *Contains(FullText)* takes the following input parameters:

   a) a *FullText* argument *text*.

2) If the first element of *SELF.PhrasePart* or the last element of *SELF.PhrasePart* is a stop word, or all elements of *SELF.PhrasePart* are stop words, then it is implementation-defined whether an exception condition is raised: *SQL/MM Full-Text - invalid search expression.*

3) Let *TL* be the result of the invocation of *text.TokenizePosition('WORDS')*and *TLE* be elements of *TL*. Every *TLE* represents some word of *text* in an implementation-defined normalized way, with leading and trailing blanks removed. It is implementation-defined whether no stop word of *text*, all stop words of *text*, or all stop words of *text* except for leading and trailing stop words are represented by some *TLE*. If stop words are included, then it is implementation-defined how they are effectively represented, provided their representation is such that the result of comparing any two stop words is true.

4) If no stop words are included in *TL* or no leading and trailing stop words are included in *TL*, then let *TL* be the result of *text.TokenizePosition('WORDS'),*with all leading stop words removed from *text* (i.e. the *TLE.Position* numbers start with 1 (one)).

5) Let *TPL* be the result of *SELF.TokenizePosition ()* and let *TPLE* be the elements of *TPL*. Every *TPLE* represents some word of *SELF.PhrasePart* in an implementation-defined normalized way with leading and trailing blanks removed. It is implementation-defined whether no stop word of *SELF.PhrasePart*, all stop words of *SELF.PhrasePart*, or all stop words of *SELF.PhrasePart* except for leading and trailing stop words are represented by some *TPLE* in an implementation-defined way, provided stop word are dealt with in the same fashion by the *TokenizePosition* methods of the *FullText* and *FT_Phrase* types.

6) Case:

   a) If *TL* is empty or if *TLE.position* of the last *TLE* is less than *TPLE.position* of the last *TPLE*, not counting the *TPLE*s representing optional words, then let *R* be <u>false</u>.

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

   c) If the cardinality of *TPL* is zero or *TPL* represents optional words only, then let *R* be <u>true</u>.

   d) Otherwise:

      i) Let *n* be the number of elements of *TPL*. Let *now* be the number of optional words. Let *STS* be a set of *m* arrays of *FT_TokenPosition* values where *m* is 2 to the power of *n* such that:

         A) *TPL* is an element of *STS*.

         B) Every other element of *STS* (if *m* > 1 (one)) is obtained from *TPL* as follows:

            1) Remove one of the possible combinations of *TPLE*s representing optional words.

            2) For each removed *TPLE*, for each subsequent *TPLE*, say *t*, reduce the value *t.position* by 1 (one).

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

      ii) Let *S1* be a sequence of *L* *TLE*s of *TL* 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 *SELF.EscapeSpec* is the null value, then let *TT* be $S2_j$ *.token*. Otherwise, let *TT* be $S2_j$*.token* ESCAPE *SELF.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 <u>true</u>.

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

            $S1_j$ `LIKE` *`TT`*

            is false for at least one *j*, then let *R* be <u>false</u>.

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

7) *Contains(FullText)* returns:

   Case:

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

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

   c) Otherwise, *R*.

8) It is implementation-defined whether the distance separating two words *W1* and *W2* in a pattern of the form <phase> or <stemmed phrase> is exactly or at most one more than the number of consecutive stop words between *W1* and *W2*. In the latter case, the stop words effectively behave like optional words.

6.6.3 StrctPattern_to_FT_Pattern Method

Purpose

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

Definition

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

      IF SELF.PhrasePart IS NULL THEN
         RETURN CAST(NULL AS FT_Pattern);
      END IF;
      SET i = 1;
      SET len = CARDINALITY(SELF.PhrasePart);
      SET result = SELF.Language || '"';
      WHILE (i <= len) DO
         SET result = result
            || InsertDQS(SELF.PhrasePart[i])
            || ' ';
         SET i = i + 1;
      END WHILE;

      SET result = TRIM(TRAILING ' ' FROM result)
         || CASE WHEN SELF.EscapeSpec IS NULL THEN
               '"'
            ELSE
               '" ESCAPE "' || SELF.EscapeSpec || '"';
            END

      IF SELF.NOT_tag IS UNKNOWN THEN
         SET result = NULL;
      ELSEIF NOT SELF.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 *FT_Pattern* value.

Description

1) The method *StrctPattern_to_FT_Pattern()* has no input parameters.

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

3) If *SELF* or *SELF.PhrasePart* is the null value or *SELF.NOT_tag* is unknown, then the result is the null value.

6.6.4 getWordArray Method

Purpose

Return a *FullText_Token* array from an *FT_Phrase* value representing a term consisting of a sequence of words (phrases).

Definition

```
CREATE METHOD getWordArray()
   RETURNS FullText_Token ARRAY[FT_MaxArrayLength]
   FOR FT_Phrase
   BEGIN
      DECLARE ret FullText_Token ARRAY[FT_MaxArrayLength];
      DECLARE len INTEGER;
      DECLARE i   INTEGER;

      SET len = CARDINALITY(SELF.PhrasePart);
      SET i   = 1;
      SET ret = CAST(ARRAY[] AS
            FullText_Token ARRAY[FT_MaxArrayLength];
      L1: WHILE (i <= len) DO
         SET ret = ret ||
            ARRAY[TRIM(BOTH ’ ’ FROM SELF.PhrasePart[i])];
         SET i = i + 1;
      END WHILE L1;
      RETURN ret;
   END
```

Definitional Rules

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

Description

1) The method *getWordArray()* has no input parameters.

2) The method *getWordArray()* returns *SELF.PhrasePart* as a *FullText_Token* array such that the *i*-th array element corresponds to the *i*-th element of *SELF.PhrasePart*. Leading and trailing blanks are removed from the array elements.

6.6.5 TokenizePosition Method

Purpose

Normalize the *PhrasePart* attribute of an *FT_Phrase* value.

Definition

```
CREATE METHOD TokenizePosition()
   RETURNS FT_TokenPosition ARRAY[FT_MaxArrayLength]
   FOR FT_Phrase
   BEGIN
      --
      -- !! See Description
      --
   END
```

Description

1) The method *TokenizePosition()* has no input parameters.

2) *TokenizePosition()* normalizes *SELF.PhrasePart* in an implementation-defined way. In addition, it is implementation-dependent whether stop words are effectively included in the result, and if so, how they are represented. However, a conforming implementation must treat stop words in this method and in the *FullText* method *TokenizePosition(FullText_Token)* in the same way.

6.6.6 FT_Phrase Methods

Purpose

Return a specified *FT_Phrase* value.

Definition

```
CREATE METHOD FT_Phrase
   (wl FullText_Token ARRAY[FT_MaxArrayLength],
    Language CHARACTER VARYING(FT_MaxLanguageLength))
   RETURNS FT_Phrase
   FOR FT_Phrase
   BEGIN
      DECLARE i INTEGER;

      IF wl IS NULL THEN
         RETURN SELF;
      END IF;
      SET SELF.Language = Language;
      SET SELF.NOT_tag = TRUE;
      SET SELF.PhrasePart =
         CAST(ARRAY[] AS FullText_Token ARRAY[FT_MaxArrayLength]);
      -- This method 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 SELF.PhrasePart = SELF.PhrasePart
            || ARRAY[EliminateDQS(wl[i + 1])];
         SET i = i + 1;
      END WHILE L1;
      RETURN SELF;
   END

CREATE METHOD FT_Phrase
   (wl FullText_Token ARRAY[FT_MaxArrayLength],
    Language CHARACTER VARYING(FT_MaxLanguageLength),
    EscapeChar CHARACTER(1))
   RETURNS FT_Phrase
   FOR FT_Phrase
   RETURN NEW FT_Phrase(w1, Language).EscapeSpec(EscapeChar)
```

Definitional Rules

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

Description

1) The method *FT_Phrase(FullText_Token ARRAY, CHARACTER VARYING)* takes the following input parameters:

   a) an array *wl* of *FullText_Token*s, representing a sequence of words,
   b) a *CHARACTER VARYING* value *Language*.

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

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

6.6.7 matches Function

Purpose

Compare two *FT_TokenPosition* array values.

Definition

```
   CREATE FUNCTION matches
      (canonictext FT_TokenPosition ARRAY[FT_MaxArrayLength],
       post     INTEGER,
       lent     INTEGER,
       canonicphr FT_TokenPosition ARRAY[FT_MaxArrayLength],
       posp     INTEGER,
       lenp     INTEGER
       EscapeChar CHARACTER(1),
       Language CHARACTER VARYING(FT_MaxLanguageLength))
      RETURNS BOOLEAN
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      CALLED ON NULL INPUT
      STATIC DISPATCH
      BEGIN
         RETURN
         CASE
            -- pattern exhausted, match found
            WHEN (posp > lenp) THEN
               TRUE
            -- text to be tested exhausted, no match found
            WHEN (post + posp - 1 > lent) THEN
               FALSE
            ELSE -- test successful so far; continue
               CASE
                  WHEN canonicphr[posp].token NOT SIMILAR ’$%+’
                        ESCAPE ’$’ THEN
                     canonictext[post+posp-1].position -
                         canonictext[post].position =
                      canonicphr[posp].position -
                         canonicphr[1].position
                     AND
                        NEW FT_TextLiteral(canonicphr[posp].token,
                          Language,EscapeChar).
                          matches(canonictext[post+posp-1].token)
                     AND
                     matches(canonictext, post, lent, canonicphr,
                       posp+1, lenp, EscapeChar, Language)
                  ELSE matches(canonictext, post, lent,
                       prune(canonicphr, posp, lenp),
                       posp, lenp-1, EscapeChar, Language)
                     OR
                     matches(canonictext, post, lent, canonicphr,
                       posp+1, lenp, EscapeChar, Language)
               END
            END
      END
```

Definitional Rules

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

2) *FT_MaxLanguageLength* is the implementation-defined maximum length for the character representation of a language specification.

Description

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

   a) an array *canonictext* of *FT_TokenPosition* items, representing a sequence of words.
   b) an INTEGER value *post*,
   c) an INTEGER value *lent*,
   d) an array *canonicphr* of *FT_TokenPosition* items, representing a sequence of words,
   e) an INTEGER value *posp*,
   f) an INTEGER value *lenp*,
   g) a CHARACTER value *EscapeChar*,
   h) a CHARACTER VARYING value *Language*.

6.6.8 prune Function

Purpose

Return an *FT_TokenPosition* array from an *FT_TokenPosition* array by removing an indicated element and adjusting the position value of subsequent elements.

Definition

```
CREATE FUNCTION prune
   (canonicphr FT_TokenPosition ARRAY[FT_MaxArrayLength],
    posp INTEGER, lenp INTEGER)
   RETURNS FT_TokenPosition ARRAY[FT_MaxArrayLength]
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT
   STATIC DISPATCH
   BEGIN
      DECLARE result FT_TokenPosition ARRAY[FT_MaxArrayLength];
      DECLARE i      INTEGER;

      SET i = 1;

      L1: WHILE (i < posp) DO
         SET result[i] = canonicphr[i];
         SET i = i + 1;
      END WHILE L1;

      L2: WHILE (i < lenp) DO
         SET result[i] = canonicphr[i+1];
         SET result[i] = result[i].position(result[i].position - 1);
         SET i = i + 1;
      END WHILE L2;

      RETURN result;
   END
```

Definitional Rules

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

Description

1) The function *prune(FT_TokenPosition ARRAY, INTEGER, INTEGER)* takes the following input parameters:

   a) an array *canonicphr* of *FT_TokenPosition* values representing a sequence of words,

   b) an *INTEGER* value *posp* which points to the element to be removed,

   c) an *INTEGER* value *lenp* which is the cardinality of *canonicphr*.

2) From *canonicphr*, the function *prune(FT_TokenPosition ARRAY, INTEGER, INTEGER)* removes the element at position *posp*. In the elements following *posp* the value of the attribute *position* is reduced by 1 (one).

6.7 FT_StemmedPhrase Type and Routines

6.7.1 FT_StemmedPhrase Type

Purpose

The *FT_StemmedPhrase* type provides facilities for the construction of stemmed phrase search patterns and for searching of occurrences of stemmed phrases in text.

Definition

```
CREATE TYPE FT_StemmedPhrase
UNDER FT_Phrase
INSTANTIABLE
NOT FINAL

OVERRIDING METHOD Contains(text FullText)
   RETURNS BOOLEAN,

OVERRIDING METHOD StrctPattern_to_FT_Pattern()
   RETURNS FT_Pattern,

METHOD TokenizePositionAndStem()
   RETURNS FT_TokenPosition ARRAY[FT_MaxArrayLength]
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT,

METHOD FT_StemmedPhrase
   (wl FullText_Token ARRAY[FT_MaxArrayLength],
    Language CHARACTER VARYING(FT_MaxLanguageLength))
   RETURNS FT_StemmedPhrase
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT,

METHOD FT_StemmedPhrase
   (wl FullText_Token ARRAY[FT_MaxArrayLength],
    Language CHARACTER VARYING(FT_MaxLanguageLength),
    EscapeChar CHARACTER(1))
   RETURNS FT_StemmedPhrase
   SELF AS RESULT
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT
```

Definitional Rules

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

2) *FT_MaxLanguageLength* is the implementation-defined maximum length for the character representation of a language specification.

Description

1) The *FT_StemmedPhrase* type provides:

a) an attribute *PhrasePart*,
b) an attribute *Language*,
c) an attribute *EscapeSpec*,
d) a method *Contains(FullText),*
e) a method *StrctPattern_to_FT_Pattern()*,
f) a method *TokenizePositionAndStem(),*
g) a method *FT_StemmedPhrase(FullText_Token ARRAY, CHARACTER VARYING)* and a method *FT_StemmedPhrase(FullText_Token ARRAY, CHARACTER VARYING, CHARACTER).*

2) An *FT_StemmedPhrase* value denotes an array of *FullText_Token* tokens which in turn represents a sequence of words. When used for searching, each such word is to be replaced by its stemmed form. The array may be empty or the null value.

A token may be the null value.

NOTE 17 - *FT_StemmedPhrase* values are intentionally more general than <phrase>s, the latter containing at least two <word representation>s, none of which may be a NULL string.

3) If a token exclusively consists of ’%’ wild card characters, then it denotes an optional word.

6.7.2 Contains Method

Purpose

Search a *FullText* value for an *FT_StemmedPhrase* value.

Definition

```
CREATE METHOD Contains
   (text FullText)
   RETURNS BOOLEAN
   FOR FT_StemmedPhrase
   BEGIN
      DECLARE tokarray FT_TokenPosition ARRAY[FT_MaxArrayLength];
      DECLARE result BOOLEAN;
      DECLARE lent INTEGER;
      DECLARE tlen INTEGER;
      DECLARE lenp INTEGER;
      DECLARE plen INTEGER;
      DECLARE nmsk INTEGER;
      DECLARE canonicphr FT_TokenPosition ARRAY[FT_MaxArrayLength];
      DECLARE i    INTEGER;

      SET tokarray = text.TokenizePositionAndStem();
      IF tokarray IS NULL THEN
         RETURN UNKNOWN;
      END IF;

      SET lent = CARDINALITY(tokarray);
      SET canonicphr = SELF.TokenizePositionAndStem();
      IF (SELF IS NULL OR canonicphr IS NULL) AND lent <> 0 THEN
         RETURN UNKNOWN;
      END IF;

      SET lenp = CARDINALITY(canonicphr);
      SET nmsk = 0;
      SET i    = 1;
      --------------------------------------------
      -- find tokens representing an optional word
      --------------------------------------------
   L1: WHILE (i <= lenp) DO
         IF canonicphr[i].token SIMILAR '$%+' ESCAPE '$' THEN
            SET nmsk = nmsk + 1;
         END IF;
         SET i = i + 1;
      END WHILE L1;
      IF lent = 0 THEN
         RETURN (FALSE = SELF.NOT_tag);
      END IF;
      IF lenp = 0 THEN
         RETURN (TRUE = SELF.NOT_tag);
      END IF;

      SET tlen = tokarray[lent].position;
      SET plen = canonicphr[lenp].position;
      IF tlen < plen - nmsk THEN
         RETURN (FALSE = SELF.NOT_tag);
      END IF;
      IF plen - nmsk = 0 THEN
         RETURN (TRUE = SELF.NOT_tag);
      END IF;

      SET result = (WITH RECURSIVE textrange (i) AS
            (VALUES (1)
               UNION
             SELECT i + 1
             FROM   textrange
             WHERE  i < lent
            ),
            Temp(BasI) AS
               (SELECT MAX(BasI)
                FROM (VALUES(1) UNION
                  SELECT
                     CASE tokarray[i].position <=
                             tlen + 1 - (plen - nmsk)
                      AND
                         matches(tokarray, i, lent, canonicphr, 1, lenp,
                              SELF.EscapeSpec, SELF.Language)
                        WHEN FALSE THEN 1
                        WHEN TRUE  THEN 3
                        ELSE            2
                     END
                  FROM textrange AS tr(i)) AS TT(BasI)
               )
               SELECT ARRAY[FALSE, UNKNOWNN, NULL][BaseI]
               FROM Temp
            );
         RETURN (SELF.NOT_tag = result);
      END
```

Definitional Rules

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

Description

1) The method *Contains(FullText)* takes the following input parameters:

   a) a *FullText* argument *text*.

2) If the first element of *SELF.PhrasePart* or the last element of *SELF.PhrasePart* is a stop word, or all elements of *SELF.PhrasePart* are stop words, then it is implementation-defined whether an exception condition is raised: *SQL/MM Full-Text - invalid search expression.*

3) Let *TL* be the result of the invocation of *text.TokenizePositionAndStem()* and *TLE* be elements of *TL*. Every *TLE* represents some word of *text* reduced to its base reduced form and in an implementation-defined normalized way, with leading and trailing blanks removed. It is implementation-defined whether no stop word of *text*, all stop words of *text*, or all stop words of *text* except for leading and trailing stop words are represented by some *TLE*. If stop words are included, then it is implementation-defined how they are effectively represented, provided their representation is such that the result of comparing any two stop words is <u>true</u>.

4) If no stop words are included in *TL* or no leading and trailing stop words are included in *TL*, then let *TL* be the result of *text.TokenizePositionAndStem()* with all leading stop words removed from *text* (i.e. the *TLE*.Position numbers start with 1 (one)).

5) Let *TPL* be the result of *SELF.TokenizePositionAndStem()*. Every element *TPLE* of *TPL* represents some word of *SELF.PhrasePart* reduced to its base reduced form and represented in an implementation-defined normalized way, with leading and trailing blanks removed. It is implementation-defined whether no stop word of *SELF.PhrasePart*, all stop words of *SELF.PhrasePart*, or all stop words of *SELF.PhrasePart* except for leading and trailing stop words are

represented by some *TPLE* in an implementation-defined way, provided stop word are dealt with in the same fashion by the *TokenizePositionAndStem* methods of the *FullText* and *FT_StemmedPhrase* types.

6) Case:

   a) If *TL* is empty or *TLE.position* of the last *TLE* is less than *TPLE.position* of the last *TPLE*, not counting the *TPLE*s representing optional words, then let *R* be false.

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

   c) If the cardinality of *TPL* is zero or *TPL* represents optional words only, then let *R* be true.

   d) Otherwise:

      i) Let *n* be the number of elements of *TPL*. Let *now* be the number of optional words. Let *STS* be a set of *m* arrays of *FT_TokenPosition* values, where *m* is 2 to the power of *n*, such that:

         A) *TPL* is an element of *STS*.

         B) Every other element of *STS* (if *m* > 1 (one)) is obtained from *TPL* as follows:

            1) Remove one of the possible combinations of *TPLE*s representing optional words.

            2) For each removed *TPLE*, for each subsequent *TPLE*, say *t*, reduce the value *t.position* by 1 (one).

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

      ii) Let *S1* be a sequence of *L TLE*s of *TL* and let *S2* be 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. Let *TT* be $S2_j$*.token.*

      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.

         C) Otherwise, let *R* be unknown.

7) *Contains(FullText)* returns:

   Case:

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

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

   c) Otherwise, *R*.

8) It is implementation-defined whether the distance between two words *W1* and *W2* in a pattern of the form <phase> or <stemmed phrase> is exactly or at most one more than the number of consecutive stop words between *W1* and *W2*. In the latter case, the stop words effectively behave like optional words.

6.7.3 StrctPattern_to_FT_Pattern Method

Purpose

Convert an *FT_StemmedPhrase* value to an *FT_Pattern* value.

Definition

```
CREATE METHOD StrctPattern_to_FT_Pattern()
   RETURNS FT_Pattern
   FOR FT_StemmedPhrase
   BEGIN
      DECLARE result CHARACTER VARYING(FT_MaxPatternLength);
      DECLARE len INTEGER;
      DECLARE i INTEGER;

      IF SELF.PhrasePart IS NULL THEN
         RETURN CAST(NULL AS FT_Pattern);
      END IF;

      SET i = 1;
      SET len = CARDINALITY(SELF.PhrasePart);
      SET result = 'STEMMED FORM OF ' || SELF.Language ||'"';
      WHILE (i <= len) DO
         SET result = result
            || InsertDQS(SELF.PhrasePart[i])
            || ' ';
         SET i = i + 1;
      END WHILE;

      SET RESULT = TRIM(TRAILING ' ' FROM result)
         || CASE WHEN SELF.EscapeSpec IS NULL THEN
            '"'
         ELSE
            '" ESCAPE "' || SELF.EscapeSpec || '"';
         END

      IF SELF.NOT_tag IS UNKNOWN THEN
         SET result = NULL;
      ELSEIF NOT SELF.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 method *StrctPattern_to_FT_Pattern()* has no input parameters.

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

3) If *SELF* or *SELF.PhrasePart* is the null value or *SELF.NOT_tag* is unknown, then the result is the null value.

6.7.4 TokenizePositionAndStem Method

Purpose

Normalize and stem-reduce the *PhrasePart* attribute of an *FT_StemmedPhrase* value.

Definition

```
CREATE METHOD TokenizePositionAndStem()
   RETURNS FT_TokenPosition ARRAY[FT_MaxArrayLength]
   FOR FT_StemmedPhrase
   BEGIN
      --
      -- !! See Description
      --
   END
```

Definitional Rules

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

Description

1) The method *TokenizePositionAndStem()* has no input parameters.

2) *TokenizePositionAndStem()* normalizes and stem-reduces the sequence of words represented by *SELF.PhrasePart* in an implementation-dependent way. In addition, it is implementation-dependent whether stop words are effectively included in the result, and if so, how they are represented. However, a conforming implementation must treat stop words in this method and in the *FullText* method *TokenizePositionAndStem()* in the same way.

6.7.5 FT_StemmedPhrase Methods

Purpose

Return a specified *FT_StemmedPhrase* value.

Definition

```
CREATE METHOD FT_StemmedPhrase
   (wl FullText_Token ARRAY[FT_MaxArrayLength],
    Language CHARACTER VARYING(FT_MaxLanguageLength))
   RETURNS FT_StemmedPhrase
   FOR FT_StemmedPhrase
   BEGIN
      DECLARE i INTEGER;

      IF wl IS NULL THEN
         RETURN SELF;
      END IF;
      SET SELF.NOT_tag = TRUE;
      SET SELF.Language = Language;
      SET SELF.PhrasePart =
         CAST(ARRAY[] AS FullText_Token ARRAY[FT_MaxArrayLength]);
         -- This method 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 SELF.PhrasePart = SELF.PhrasePart
            || ARRAY[EliminateDQS(wl[i + 1])];
         SET i = i + 1;
      END WHILE L1;
      RETURN SELF;
   END

CREATE METHOD FT_StemmedPhrase
   (wl FullText_Token ARRAY[FT_MaxArrayLength],
    Language CHARACTER VARYING(FT_MaxLanguageLength),
    EscapeChar CHARACTER(1))
   RETURNS FT_StemmedPhrase
   FOR FT_StemmedPhrase
   RETURN NEW FT_StemmedPhrase(wl, Language).EscapeSpec(EscapeChar)
```

Definitional Rules

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

2) *FT_MaxLanguageLength* is the implementation-defined maximum length for the character representation of a language specification.

Description

1) The method *FT_StemmedPhrase(FullText_Token ARRAY, CHARACTER VARYING*) takes the following input parameters:

    a) an array *wl* of *FullText_Token*s, representing a sequence of words,
    b) a *CHARACTER VARYING* value *Language*.

2) The method *FT_StemmedPhrase(FullText_Token ARRAY, CHARACTER VARYING, CHARACTER)* takes the following input parameters:

    a) an array *wl* of *FullText_Token*s, representing a sequence of words,
    b) a *CHARACTER VARYING* value *Language*,
    c) a *CHARACTER* value *EscapeChar*.

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

6.8 FT_Proxi Type and Routines

6.8.1 FT_Proxi Type

Purpose

FT_Proxi values represent proximity search patterns.

Definition

```
CREATE TYPE FT_Proxi
   UNDER FT_Primary
   AS (
      TL1 FT_TextLiteral ARRAY[FT_MaxArrayLength],
      TL2 FT_TextLiteral ARRAY[FT_MaxArrayLength],
      dv  INTEGER,        -- distance value
      du  FullText_Token, -- distance unit
      oi  FullText_Token  -- order indicator
   )
   INSTANTIABLE
   NOT FINAL

   OVERRIDING METHOD Contains(text FullText)
      RETURNS BOOLEAN,

   OVERRIDING METHOD StrctPattern_to_FT_Pattern()
      RETURNS FT_Pattern,

   METHOD FT_Proxi
      (TokList1 FT_TextLiteral ARRAY[FT_MaxArrayLength],
       TokList2 FT_TextLiteral ARRAY[FT_MaxArrayLength],
       DistanceValue INTEGER,
       DistanceUnit FullText_Token,
       OrderIndicator FullText_Token)
      RETURNS FT_Proxi
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      CALLED ON NULL INPUT
```

Definitional Rules

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

Description

1) The *FT_Proxi* type provides:

   a) an attribute *TL1*,
   b) an attribute *TL2*,
   c) an attribute *dv*,
   d) an attribute *du*,
   e) an attribute *oi*,
   f) a method *Contains(FullText)*,
   g) a method *StrctPattern_to_FT_Pattern()*,
   h) a method *FT_Proxi(FT_TextLiteral ARRAY, FT_TextLiteral ARRAY, INTEGER, FullText_Token, FullText_Token)*.

6.8.2 Contains Method

Purpose

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

Definition

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

      IF SELF.du <> ’CHARACTERS’ AND
         SELF.du <> ’WORDS’ AND
         SELF.du <> ’SENTENCES’ AND
         SELF.du <> ’PARAGRAPHS’ THEN
         RETURN -- !! See Description ;
      END IF;

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

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

      IF SELF IS NULL OR SELF.TL2 IS NULL THEN
         SET lentl2 = CAST(NULL AS INTEGER)
      ELSE
         SET lentl2 = CARDINALITY(SELF.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 =
            (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, SELF.TL1[1])
                     UNION
                   SELECT ind + 1, SELF.TL1[ind + 1]
```

```
                FROM   tl1Tab
                WHERE  ind < lentl1
               ),
            tl2Tab(ind, tok) AS
               (VALUES(1, SELF.TL2[1])
                  UNION
                SELECT ind + 1, SELF.TL2[ind + 1]
                FROM   tl2Tab
                WHERE  ind < lentl2
               ),
             Temp[BasI] AS
            (SELECT MAX(BasI)
             FROM (VALUES(1) UNION
               SELECT
                  CASE l1.tok.Contains(
                        NEW FullText(tt1.tp.token, text.Language))
                      AND l2.tok.Contains(
                        NEW FullText(tt2.tp.token, text.Language))
                      AND tt2.tp.position
                         BETWEEN tt1.tp.position
                            - (SELF.dv + tt2.tp.corrVal) *
                            (CASE SELF.oi
                               WHEN ’IN ORDER’ THEN 0
                               ELSE 1
                             END)
                      AND tt1.tp.position
                        + SELF.dv + tt1.tp.corrVal
                     WHEN FALSE THEN 1
                     WHEN TRUE  THEN 3
                     ELSE            2
                  END
               FROM ttTab tt1, tl1Tab l1, ttTab tt2, tl2Tab l2)
                  AS TT(BasI)
            )
            SELECT ARRAY[FALSE, UNKNOWN, TRUE][BasI] FROM Temp
         );
      END IF;
   RETURN (SELF.NOT_tag = result);
END
```

Definitional Rules

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

Description

1) The method *Contains(FullText)* takes the following input parameters:

   a) a *FullText* value *text*.

2) Case:

   i) If *SELF.TL1*, *SELF.TL2* or the result of *text.TokenizePosition(SELF.du)* is empty, then let *R* be <u>false</u>.

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

   iii) Otherwise, let *TPS1* be the result of *text.TokenizePosition(SELF.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 *SELF.oi* has the value ’IN ORDER’ and the difference

    `tp2.pos - tp1.pos`

    is not negative and not greater than the distance value *SELF.dv*.

B) The order indication *SELF.oi* has the value 'ANY ORDER’ and the absolute value of the difference

    `tp2.pos - tp1.pos`

    is not greater than the distance value *SELF.dv*.

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

Case:

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

    `w1.Contains(NEW FullText(tp1.token), text.Language))`

    and

    `w2.Contains(NEW FullText(tp2.token), text.Language))`

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

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

    `w1.Contains(NEW FullText(tp1.token), text.Language))`

    and

    `w2.Contains(NEW FullText(tp2.token), text.Language))`

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

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

NOTE 18 - The method *Contains* is described in Subclause 6.4.2, “Contains Method” and Subclause 6.4.4, “Contains Method”.

3) *Contains(FullText)* returns:

   Case:

   a) <u>unknown</u>, if *SELF.NOT_tag* is the null value.

   b) NOT *R*, if *SELF.NOT_tag* is <u>false</u>.

   c) Otherwise, *R*.

6.8.3 StrctPattern_to_FT_Pattern Method

Purpose

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

Definition

```
CREATE METHOD StrctPattern_to_FT_Pattern()
   RETURNS FT_Pattern
   FOR FT_Proxi
   BEGIN
      DECLARE result CHARACTER VARYING(FT_MaxPatternLength);

      SET result = CAST(StrctPattern_to_FT_Pattern(SELF.TL1)
            AS CHARACTER VARYING(FT_MaxPatternLength))
         || ’ NEAR ’
         || CAST(StrctPattern_to_FT_Pattern(SELF.TL2)
            AS CHARACTER VARYING(FT_MaxPatternLength))
         || ’ WITHIN ’
         || CAST(SELF.dv AS CHARACTER VARYING(FT_MaxPatternLength))
         || ’ ’ || TRIM(BOTH ’ ’ FROM SELF.du)
         || ’ ’ || TRIM(BOTH ’ ’ FROM SELF.oi);

      IF SELF.NOT_tag IS UNKNOWN THEN
         SET result = NULL;
      ELSEIF NOT SELF.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 *FT_Pattern* value.

Description

1) The method *StrctPattern_to_FT_Pattern()* has no input parameters.

2) *StrctPattern_to_FT_Pattern()* returns an *FT_Pattern* value of the form <Proximity expansion> or the form NOT <Proximity expansion>.

3) If *SELF* or any of the attributes *SELF.TL1*, *SELF.du*, *SELF.dv*, *SELF.oi* are the null value or *SELF.NOT_tag* is unknown, then the result is the null value.

6.8.4 FT_Proxi Method

Purpose

Return a specified *FT_Proxi* value.

Definition

```
CREATE METHOD FT_Proxi
   (TokList1 FT_TextLiteral ARRAY[FT_MaxArrayLength],
    TokList2 FT_TextLiteral ARRAY[FT_MaxArrayLength],
    DistanceValue INTEGER,
    DistanceUnit FullText_Token,
    OrderIndicator FullText_Token)
   RETURNS FT_Proxi
   FOR FT_Proxi
   RETURN SELF.TLI(TokList1).TL2(Toklist2).
      dv(DistanceValue).du(DistanceUnit).
      oi(OrderIndicator).NOT_tag(TRUE)
```

Definitional Rules

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

Description

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

   a) an array *TokList1* of *FT_TextLiteral* elements, which represents a set of words,
   b) an array *TokList2* of *FT_TextLiteral* 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 the null value. *TokList1* and *TokList2* may be empty.

6.9 FT_Soundex Type and Routines

6.9.1 FT_Soundex Type

Purpose

FT_Soundex values represent a search token to be matched in text due to phonetic criteria.

Definition

```
CREATE TYPE FT_Soundex
   UNDER FT_Primary
   AS (
      spoken FT_TextLiteral
   )
   INSTANTIABLE
   NOT FINAL

   OVERRIDING METHOD Contains(text FullText)
      RETURNS BOOLEAN,

   OVERRIDING METHOD StrctPattern_to_FT_Pattern()
      RETURNS FT_Pattern,

   METHOD FT_Soundex(snd FT_TextLiteral)
      RETURNS FT_Soundex
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      CALLED ON NULL INPUT
```

Description

1) The *FT_Soundex* type provides:

   a) an attribute *spoken*,
   b) a method *Contains(FullText)*,
   c) a method *StrctPattern_to_FT_Pattern()*,
   d) a method *FT_Soundex(FT_TextLiteral)*,
   e) a function *GetSoundsSimilar(FT_TextLiteral)*.

6.9.2 Contains Method

Purpose

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

Definition

```
CREATE METHOD Contains
   (text FullText)
   RETURNS BOOLEAN
   FOR FT_Soundex
   RETURN (SELF.NOT_tag =
      NEW FT_Any(GetSoundsSimilar(SELF.spoken)).Contains(text))
```

Description

1) The method *Contains(FullText)* takes the following input parameters:

   a) a *FullText* value *text*.

2) Let *R* be the result of

   ```
   NEW FT_Any(GetSoundsSimilar(SELF.spoken)).Contains(text)
   ```

   Case:

   a) If *SELF.NOT_tag* is unknown, then *Contains(FullText)* returns unknown.

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

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

6.9.3 StrctPattern_to_FT_Pattern Method

Purpose

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

Definition

```
CREATE METHOD StrctPattern_to_FT_Pattern()
   RETURNS FT_Pattern
   FOR FT_Soundex
   BEGIN
      DECLARE result CHARACTER VARYING(FT_MaxPatternLength);

      IF SELF.NOT_tag IS UNKNOWN THEN
         RETURN CAST(NULL AS FT_Pattern);
      END IF;

      SET result = 'SOUNDS LIKE '
            || CAST(SELF.spoken.StrctPattern_to_FT_Pattern()
               AS CHARACTER VARYING(FT_MaxPatternLength));

      IF NOT SELF.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 *FT_Pattern* value.

Description

1) The method *StrctPattern_to_FT_Pattern()* has no input parameters.

2) *StrctPattern_to_FT_Pattern()* returns an *FT_Pattern* value of the form <Soundex expansion> or the form NOT <Soundex expansion>.

3) If *SELF*, *SELF.spoken* or *SELF.spoken.LitPart* is the null value or *SELF.NOT_tag* is unknown, then the result is the null value.

6.9.4 FT_Soundex Method

Purpose

Return a specified *FT_Soundex* value.

Definition

```
CREATE METHOD FT_Soundex
   (snd FT_TextLiteral)
   RETURNS FT_Soundex
   FOR FT_Soundex
   RETURN SELF.spoken(snd).NOT_tag(TRUE)
```

Description

1) The method *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.

6.9.5 GetSoundsSimilar Function

Purpose

Return an array of words that sound like a given word.

Definition

```
CREATE FUNCTION GetSoundsSimilar
   (spoken FT_TextLiteral)
   RETURNS FT_TextLiteral ARRAY[FT_MaxArrayLength]
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT
   STATIC DISPATCH
   BEGIN
      --
      -- !! See Description
      --
   END
```

Definitional Rules

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

Description

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

   a) an *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. The mechanism for generating this array, taking into account the language as specified in *spoken.Language*, is implementation-dependent.

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.

6.10 FT_BroaderTerm Type and Routines

6.10.1 FT_BroaderTerm Type

Purpose

FT_BroaderTerm values 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
   AS (
      thesaurus  CHARACTER VARYING(FT_ThesNameLength),
      startingTerm FT_WordOrPhrase,
      expansionCnt INTEGER
   )
   INSTANTIABLE
   NOT FINAL

   OVERRIDING METHOD Contains(text FullText)
      RETURNS BOOLEAN,

   OVERRIDING METHOD StrctPattern_to_FT_Pattern()
      RETURNS FT_Pattern,

   METHOD FT_BroaderTerm
      (thes_name CHARACTER VARYING(FT_ThesNameLength),
       strt FT_WordOrPhrase,
       thes_exp_count INTEGER)
      RETURNS FT_BroaderTerm
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      CALLED ON NULL INPUT
```

Definitional Rules

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

Description

1) The *FT_BroaderTerm* type provides:

   a) an attribute *thesaurus*,
   b) an attribute *startingTerm*,
   c) an attribute *expansionCnt*,
   d) a method *Contains(FullText)*,
   e) a method *StrctPattern_to_FT_Pattern()*,
   f) a method *FT_BroaderTerm(CHARACTER VARYING, FT_WordOrPhrase, INTEGER)*,
   g) a function *GetBroaderTerms(CHARACTER VARYING, FT_WordOrPhrase, INTEGER)*.

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.

6.10.2 Contains Method

Purpose

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

Definition

```
CREATE METHOD Contains
   (text FullText)
   RETURNS BOOLEAN
   FOR FT_BroaderTerm
   BEGIN
      DECLARE BrdArray FT_WordOrPhrase ARRAY[FT_MaxArrayLength];
      DECLARE result BOOLEAN;

      SET BrdArray = GetBroaderTerms(SELF.thesaurus ,
            SELF.startingTerm,
            SELF.expansionCnt);
      SET result = NEW FT_Any(BrdArray).Contains(text);

      RETURN (SELF.NOT_tag = result);
   END
```

Definitional Rules

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

Description

1) The method *Contains(FullText)* takes the following input parameters:

   a) a *FullText* value *text*.

2) Let *R* be the result of

```
NEW FT_Any(GetBroaderTerms(SELF.thesaurus, SELF.startingTerm,
   SELF.expansionCnt)).Contains(text)
```

   Case:

   a) If *SELF.NOT_tag* is unknown, then *Contains(FullText)* returns unknown.

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

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

6.10.3 StrctPattern_to_FT_Pattern Method

Purpose

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

Definition

```
CREATE METHOD StrctPattern_to_FT_Pattern()
   RETURNS FT_Pattern
   FOR FT_BroaderTerm
   BEGIN
      DECLARE result CHARACTER VARYING(FT_MaxPatternLength);

      IF SELF.NOT_tag IS UNKNOWN THEN
         RETURN CAST(NULL AS FT_Pattern);
      END IF;

      SET result = ’THESAURUS "’
            || SELF.thesaurus
            || ’" EXPAND BROADER TERM OF ’
            || CAST(SELF.startingTerm.StrctPattern_to_FT_Pattern()
                  AS CHARACTER VARYING(FT_MaxPatternLength))
            || CASE WHEN SELF.expansionCnt IS NULL THEN
                  ’’
               ELSE
                  ’FOR ’
                  || TRIM(BOTH ’ ’ FROM CAST(SELF.expansionCnt
                     AS CHARACTER VARYING(FT_MaxPatternLength))
                  || ’ LEVELS’
               END
            ;

      IF NOT SELF.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 *FT_Pattern* value.

Description

1) The method *StrctPattern_to_FT_Pattern()* has no input parameters.

2) *StrctPattern_to_FT_Pattern()* returns an *FT_Pattern* value of the form <Broader_Term expansion> or NOT <Broader_Term expansion>.

3) If *SELF*, *SELF*.thesaurus,or *SELF.startingTerm* is the null value or *SELF.NOT_tag* is unknown, then the result is the null value.

6.10.4 FT_BroaderTerm Method

Purpose

Return a specified *FT_BroaderTerm* value.

Definition

```
CREATE METHOD FT_BroaderTerm
   (thes_name CHARACTER VARYING(FT_ThesNameLength),
    strt FT_WordOrPhrase,
    thes_exp_count INTEGER)
   RETURNS FT_BroaderTerm
   FOR FT_BroaderTerm
   RETURN SELF.thesaurus(thes_name).startingTerm(strt).
      expansionCnt(thes_exp_count).NOT_tag(TRUE)
```

Definitional Rules

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

Description

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

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

6.10.5 GetBroaderTerms Function

Purpose

Get broader terms from a thesaurus.

Definition

```
CREATE FUNCTION GetBroaderTerms
   (thes_name CHARACTER VARYING(FT_ThesNameLength),
    startingTerm FT_WordOrPhrase,
    thes_exp_count INTEGER)
   RETURNS FT_WordOrPhrase ARRAY[FT_MaxArrayLength]
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT
   STATIC DISPATCH
   BEGIN
      DECLARE ret  FT_WordOrPhrase ARRAY[FT_MaxArrayLength];
      DECLARE strt FullText_Token ARRAY[FT_MaxArrayLength];
      DECLARE strt_termid INTEGER;
      DECLARE local_exp_count INTEGER;

      SET thes_name = TRIM(BOTH ’ ’ FROM thes_name);
      SET strt = startingTerm.getWordArray();

      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 EXPR.getWordArray() = strt
                AND TRIM(BOTH ’ ’ FROM THNAME_DIC) = thes_name
         );

      SET ret=CAST(ARRAY[] AS FT_WordOrPhrase
         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[TD.EXPR] AS EXPRarr1
           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 = ret || EXPRarr1;
        END FOR L1;
        RETURN ret;
     END
```

Definitional Rules

1) *FT_ThesNameLength* is the implementation-defined 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 *GetBroaderTerms(CHARACTER VARYING, FT_WordOrPhrase, INTEGER)* takes the following input parameters:

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

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

3) *GetBroaderTerms(CHARACTER VARYING, FT_WordOrPhrase, INTEGER)* returns an empty array if one of the following is true:

   a) The term *startingTerm* is not contained in column *NarrowerTerm* of *TH*.
   b) Either *startingTerm* or *thes_name* is the null value.

4) If the expansion count *thes_exp_count* is zero, *GetBroaderTerms(CHARACTER VARYING, FT_WordOrPhrase, INTEGER)* returns all terms in column *BroaderTerm* of those rows of *TH* the values of which in column *NarrowerTerm* are equivalent to *startingTermt*. 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, startingTerm, 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, Ei, 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 *startingTerm* is **not** included in the result.

7) It is implementation-defined, whether a check is made to ensure that the language specified in *startingTerm.Language* is compatible with the thesaurus as specified by *thes_name*, and if so, what kind of condition is raised in case of a language incompatibility.

6.11 FT_NarrowerTerm Type and Routines

6.11.1 FT_NarrowerTerm Type

Purpose

FT_NarrowerTerm values 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
   AS (
      thesaurus CHARACTER VARYING(FT_ThesNameLength),
      startingTerm FT_WordOrPhrase,
      expansionCnt INTEGER
   )
   INSTANTIABLE
   NOT FINAL

   OVERRIDING METHOD Contains(text FullText)
      RETURNS BOOLEAN,

   OVERRIDING METHOD StrctPattern_to_FT_Pattern()
      RETURNS FT_Pattern,

   METHOD FT_NarrowerTerm
      (thes_name CHARACTER VARYING(FT_ThesNameLength),
       strt FT_WordOrPhrase,
       thes_exp_count INTEGER)
      RETURNS FT_NarrowerTerm
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      CALLED ON NULL INPUT
```

Definitional Rules

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

Description

1) The *FT_NarrowerTerm* type provides:

   a) an attribute *thesaurus*,
   b) an attribute *startingTerm*,
   c) an attribute *expansionCnt*,
   d) a method *Contains(FullText),*
   e) a method *StrctPattern_to_FT_Pattern()*,
   f) a method *FT_NarrowerTerm(CHARACTER VARYING, FT_WordOrPhrase, INTEGER)*,
   g) a function *GetNarrowerTerms(CHARACTER VARYING, FT_WordOrPhrase, INTEGER).*

2) For the purpose of this 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.

6.11.2 Contains Method

Purpose

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

Definition

```
CREATE METHOD Contains
   (text FullText)
   RETURNS BOOLEAN
   FOR FT_NarrowerTerm
   BEGIN
      DECLARE NrwArray FT_WordOrPhrase ARRAY[FT_MaxArrayLength];
      DECLARE result BOOLEAN;

      SET NrwArray = GetNarrowerTerms(SELF.thesaurus ,
            SELF.startingTerm, SELF.expansionCnt);
      SET result = NEW FT_Any(NrwArray).Contains(text);

      RETURN (SELF.NOT_tag = result);
   END
```

Definitional Rules

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

Description

1) The method *Contains(FullText)* takes the following input parameters:

   a) a *FullText* value *text*.

2) Let *R* be the result of

```
NEW FT_Any(GetNarrowerTerms(SELF.thesaurus, SELF.startingTerm,
   SELF.expansionCnt)).Contains(text)
```

   Case:

   a) If *SELF.NOT_tag* is unknown, then *Contains(FullText)* returns unknown.

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

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

6.11.3 StrctPattern_to_FT_Pattern Method

Purpose

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

Definition

```
CREATE METHOD StrctPattern_to_FT_Pattern()
   RETURNS FT_Pattern
   FOR FT_NarrowerTerm
   BEGIN
      DECLARE result CHARACTER VARYING(FT_MaxPatternLength);

      IF SELF.NOT_tag IS UNKNOWN THEN
         RETURN CAST(NULL AS FT_Pattern)
      END IF;

      SET result = 'THESAURUS "'
            || SELF.thesaurus
            || '" EXPAND NARROWER TERM OF '
            || CAST(SELF.startingTerm.StrctPattern_to_FT_Pattern()
                  AS CHARACTER VARYING(FT_MaxPatternLength))
            || CASE WHEN SELF.expansionCnt IS NULL THEN
                  ''
               ELSE
                  'FOR '
                  || TRIM(BOTH ' ' FROM CAST(SELF.expansionCnt
                     AS CHARACTER VARYING(FT_MaxPatternLength))
                  || ' LEVELS'
               END
            ;
      IF NOT SELF.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 *FT_Pattern* value.

Description

1) The method *StrctPattern_to_FT_Pattern(FT_NarrowerTerm)* has no input parameters:

2) *StrctPattern_to_FT_Pattern()* returns an *FT_Pattern* value of the form <Narrower_Term expansion> or NOT <Narrower_Term expansion>.

3) If *SELF*, *SELF.thesaurus*, or *SELF.startingTerm* is the null value or *SELF.NOT_tag* is unknown, then the result is the null value.

6.11.4 FT_NarrowerTerm Method

Purpose

Return a specified *FT_NarrowerTerm* value.

Definition

```
CREATE METHOD FT_NarrowerTerm
   (thes_name CHARACTER VARYING(FT_ThesNameLength),
    strt FT_WordOrPhrase,
    thes_exp_count INTEGER)
   RETURNS FT_NarrowerTerm
   FOR FT_NarrowerTerm
   RETURN SELF.thesaurus(thes_name).
      startingTerm(strt).expansionCnt(thes_exp_count).
      NOT_tag(TRUE)
```

Definitional Rules

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

Description

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

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

6.11.5 GetNarrowerTerms Function

Purpose

Get narrower terms from a thesaurus.

Definition

```
CREATE FUNCTION GetNarrowerTerms
   (thes_name CHARACTER VARYING(FT_ThesNameLength),
    startingTerm FT_WordOrPhrase,
    thes_exp_count INTEGER)
   RETURNS FT_WordOrPhrase ARRAY[FT_MaxArrayLength]
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT
   STATIC DISPATCH
   BEGIN
      DECLARE ret FT_WordOrPhrase ARRAY[FT_MaxArrayLength];
      DECLARE strt FullText_Token ARRAY[FT_MaxArrayLength];
      DECLARE strt_termid INTEGER;
      DECLARE local_exp_count INTEGER;

      SET thes_name = TRIM(BOTH ' ' FROM thes_name);
      SET strt = startingTerm.getWordArray();

      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 EXPR.getWordArray() = strt
                AND TRIM(BOTH ' ' FROM THNAME_DIC) = thes_name
         );

      SET ret = CAST(ARRAY[] AS FT_WordOrPhrase
         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[TD.EXPR] AS EXPRarr1
      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 = ret || EXPRarr1;
   END FOR L1;
   RETURN ret;
END
```

Definitional Rules

1) *FT_ThesNameLength* is the implementation-defined 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 *GetNarrowerTerms(CHARACTER VARYING, FT_WordOrPhrase, INTEGER)* takes the following input parameters:

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

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

3) *GetNarrowerTerms(CHARACTER VARYING, FT_WordOrPhrase, INTEGER)* returns an empty array if one of the following is true:

   a) The term *startingTerm* is not contained in column *BroaderTerm* of *TH*.
   b) Either *startingTermt* 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_WordOrPhrase, INTEGER)* returns all terms in column *NarrowerTerm* of those rows of *TH* the values of which in column *BroaderTerm* are equivalent to *startingTerm*. 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, startingTerm, 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,` E_i`, 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 *startingTerm* is **not** included in the result.

7) It is implementation-defined, whether a check is made to ensure that the language specified in *startingTerm.Language* is compatible with the thesaurus as specified by *thes_name*, and if so, what kind of condition is raised in case of a language incompatibility.

6.12 FT_Synonym Type and Routines

6.12.1 FT_Synonym Type

Purpose

FT_Synonym values 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
   AS (
      thesaurus CHARACTER VARYING(FT_ThesNameLength),
      startingTerm FT_WordOrPhrase
   )
   INSTANTIABLE
   NOT FINAL

   OVERRIDING METHOD Contains(text FullText)
      RETURNS BOOLEAN,

   OVERRIDING METHOD StrctPattern_to_FT_Pattern()
      RETURNS FT_Pattern,

   METHOD FT_Synonym
      (thes_name CHARACTER VARYING(FT_ThesNameLength),
       strt FT_WordOrPhrase)
      RETURNS FT_Synonym
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      CALLED ON NULL INPUT
```

Definitional Rules

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

Description

1) The *FT_Synonym* type provides:

   a) an attribute *thesaurus*,
   b) an attribute *startingTerm*,
   c) a method *Contains(FullText)*,
   d) a method *StrctPattern_to_FT_Pattern(),*
   e) a method *FT_Synonym(CHARACTER VARYING, FT_WordOrPhrase)*,
   f) a function *GetSynonymTerms(CHARACTER VARYING, FT_WordOrPhrase).*

2) For the purpose of this 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.

6.12.2 Contains Method

Purpose

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

Definition

```
CREATE METHOD Contains
   (text FullText)
   RETURNS BOOLEAN
   FOR FT_Synonym
   BEGIN
      DECLARE SynArray FT_WordOrPhrase ARRAY[FT_MaxArrayLength];
      DECLARE result BOOLEAN;

      SET SynArray = GetSynonymTerms(SELF.thesaurus,
            SELF.startingTerm);
      SET result = NEW FT_Any(SynArray).Contains(text);

      RETURN (SELF.NOT_tag = result);
   END
```

Definitional Rules

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

Description

1) The method *Contains(FullText)* takes the following input parameters:

   a) a *FullText* value *text*.

2) Let *R* be the result of

```
NEW FT_Any(GetSynonymTerms(SELF.thesaurus, SELF.startingTerm)).
   Contains(text)
```

3) Case:

   a) If *SELF.NOT_tag* is unknown, then *Contains(FullText)* returns unknown.

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

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

6.12.3 StrctPattern_to_FT_Pattern Method

Purpose

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

Definition

```
CREATE METHOD StrctPattern_to_FT_Pattern()
   RETURNS FT_Pattern
   FOR FT_Synonym
   BEGIN
      DECLARE result CHARACTER VARYING(FT_MaxPatternLength);

      IF SELF.NOT_tag IS UNKNOWN THEN
         RETURN CAST(NULL AS FT_Pattern)
      END IF;

      SET result = ’THESAURUS "’
            || SELF.thesaurus
            || ’" EXPAND SYNONYM TERM OF ’
            || CAST(SELF.startingTerm.StrctPattern_to_FT_Pattern()
                  AS CHARACTER VARYING(FT_MaxPatternLength));

      IF NOT SELF.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 *FT_Pattern* value.

Description

1) The method *StrctPattern_to_FT_Pattern()* has no input parameters.

2) *StrctPattern_to_FT_Pattern()* returns an *FT_Pattern* value of the form <Synonym_Term expansion> or NOT <Synonym_Term expansion>.

3) If *SELF*, *SELF.thesaurus*, or *SELF.startingTerm* is the null value or *SELF.NOT_tag* is unknown, then the result is the null value.

6.12.4 FT_Synonym Method

Purpose

Return a specified *FT_Synonym* value.

Definition

```
CREATE METHOD FT_Synonym
   (thes_name CHARACTER VARYING(FT_ThesNameLength),
    strt FT_WordOrPhrase)
   RETURNS FT_Synonym
   FOR FT_Synonym
   RETURN SELF.thesaurus(thes_name).startingTerm(strt).
      NOT_tag(TRUE)
```

Definitional Rules

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

Description

1) The method *FT_Synonym(CHARACTER VARYING, FT_WordOrPhrase)* takes the following input parameters:

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

6.12.5 GetSynonymTerms Function

Purpose

Get synonym terms from a thesaurus.

Definition

```
CREATE FUNCTION GetSynonymTerms
   (thes_name CHARACTER VARYING(FT_ThesNameLength),
    startingTerm FT_WordOrPhrase)
   RETURNS FT_WordOrPhrase ARRAY[FT_MaxArrayLength]
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT
   STATIC DISPATCH
   BEGIN
      DECLARE ret FT_WordOrPhrase ARRAY[FT_MaxArrayLength];
      DECLARE strt FullText_Token ARRAY[FT_MaxArrayLength];
      DECLARE strt_termid INTEGER;

      SET thes_name = TRIM(BOTH ’ ’ FROM thes_name);
      SET strt = startingTerm.getWordArray();
      SET strt_termid =
         (SELECT TERMID
          FROM TERM_DICTIONARY
          WHERE EXPR.getWordArray() = strt
                AND TRIM(BOTH ’ ’ FROM THNAME_DIC) = thes_name
         );

      SET ret = CAST(ARRAY[] AS FT_WordOrPhrase
         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_SYN) = 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_SYN) = thes_name
            )
         SELECT ARRAY[TD.EXPR] AS EXPRarr1
         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 = ret || EXPRarr1;
      END FOR L1;
      RETURN ret ||
         CASE
            WHEN startingTerm IS NULL OR thes_name IS NULL THEN
               CAST(ARRAY[] AS FT_WordOrPhrase
                  ARRAY[FT_MaxArrayLength])
            ELSE
               ARRAY[startingTerm]
```

```
        END;
    END
```

Definitional Rules

1) *FT_ThesNameLength* is the implementation-defined 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 *GetSynonymTerms(CHARACTER VARYING, FT_WordOrPhrase)* takes the following input parameters:

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

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

   3) *GetSynonymTerms(CHARACTER VARYING, FT_WordOrPhrase)* returns an empty array if either *startingTerm* or *thes_name* is the null value.

4) Let R_0 be a set containing *startingTerm* as its only element, let n be the number of *Ring* values containing *startingTerm*, and let R_i denote a single element set containing such a value (if any). The result of invoking *GetSynonymTerms(CHARACTER VARYING, FT_WordOrPhrase)* represents the following set:

   R_0 UNION R_1 UNION ... R_i ... UNION R_n

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

6) It is implementation-defined, whether a check is made to ensure that the language specified in *startingTerm.Language* is compatible with the thesaurus as specified by *thes_name*, and if so, what kind of condition is raised in case of a language incompatibility.

6.13 FT_PreferredTerm Type and Routines

6.13.1 FT_PreferredTerm Type

Purpose

FT_Preferred Term values 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
   AS (
      thesaurus CHARACTER VARYING(FT_ThesNameLength),
      startingTerm FT_WordOrPhrase
   )
   INSTANTIABLE
   NOT FINAL

   OVERRIDING METHOD Contains(text FullText)
      RETURNS BOOLEAN,

   OVERRIDING METHOD StrctPattern_to_FT_Pattern()
      RETURNS FT_Pattern,

   METHOD FT_PreferredTerm
      (thes_name CHARACTER VARYING(FT_ThesNameLength),
       strt FT_WordOrPhrase)
      RETURNS FT_PreferredTerm
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      CALLED ON NULL INPUT
```

Definitional Rules

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

Description

1) The *FT_PreferredTerm* type provides:

    a) an attribute *thesaurus*,
    b) an attribute *startingTerm*,
    c) a method *Contains(FullText)*,
    d) a method *StrctPattern_to_FT_Pattern()*,
    e) a method *FT_PreferredTerm(CHARACTER VARYING, FT_WordOrPhrase)*,
    f) a function *GetPreferredTerms(CHARACTER VARYING, FT_WordOrPhrase)*.

2) For the purpose of this 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.

6.13.2 Contains Method

Purpose

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

Definition

```
CREATE METHOD Contains
   (text FullText)
   RETURNS BOOLEAN
   FOR FT_PreferredTerm
   BEGIN
      DECLARE PfdArray FT_WordOrPhrase ARRAY[FT_MaxArrayLength];
      DECLARE result BOOLEAN;

      SET PfdArray = GetPreferredTerms(SELF.thesaurus,
            SELF.startingTerm);
      SET result = NEW FT_Any(PrdArray).Contains(text);

      RETURN (SELF.NOT_tag = result);
   END
```

Definitional Rules

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

Description

1) The method *Contains(FullText)* takes the following input parameters:

   a) a *FullText* value *text*.

2) Let *R* be the result of

```
NEW FT_Any(GetPreferredTerms(SELF.thesaurus,
   SELF.startingTerm)).Contains(text)
```

3) Case:

   a) If *SELF.NOT_tag* is unknown, then *Contains(FullText)* returns unknown.

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

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

6.13.3 StrctPattern_to_FT_Pattern Method

Purpose

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

Definition

```
CREATE METHOD StrctPattern_to_FT_Pattern()
   RETURNS FT_Pattern
   FOR FT_PreferredTerm
   BEGIN
      DECLARE result CHARACTER VARYING(FT_MaxPatternLength);

      IF SELF.NOT_tag IS UNKNOWN THEN
         RETURN CAST(NULL AS FT_Pattern)
      END IF;

      SET result = 'THESAURUS "'
            || SELF.thesaurus
            || '" EXPAND PREFERRED TERM OF '
            || CAST(SELF.startingTerm.StrctPattern_to_FT_Pattern()
                  AS CHARACTER VARYING(FT_MaxPatternLength));

      IF NOT SELF.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 *FT_Pattern* value.

Description

1) The method *StrctPattern_to_FT_Pattern()* has no input parameters.

2) *StrctPattern_to_FT_Pattern()* returns an *FT_Pattern* value of the form <Preferred_Term expansion> or NOT <Preferred_Term expansion>.

3) If *SELF*, *SELF.thesaurus*, or *SELF.startingTerm* is the null value or *SELF.NOT_tag* is unknown, then the result is the null value.

6.13.4 FT_PreferredTerm Method

Purpose

Return a specified *FT_PreferredTerm* value.

Definition

```
CREATE METHOD FT_PreferredTerm
   (thes_name CHARACTER VARYING(FT_ThesNameLength),
    strt FT_WordOrPhrase)
   RETURNS FT_PreferredTerm
   FOR FT_PreferredTerm
   RETURN SELF.thesaurus(thes_name).startingTerm(strt).
      NOT_tag(TRUE)
```

Definitional Rules

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

Description

1) The method *FT_PreferredTerm(CHARACTER VARYING, FT_WordOrPhrase)* takes the following input parameters:

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

6.13.5 GetPreferredTerms Function

Purpose

Get preferred terms from a thesaurus.

Definition

```
CREATE FUNCTION GetPreferredTerms
   (thes_name CHARACTER VARYING(FT_ThesNameLength),
    startingTerm FT_WordOrPhrase)
   RETURNS FT_WordOrPhrase ARRAY[FT_MaxArrayLength]
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT
   STATIC DISPATCH
   BEGIN
      DECLARE ret  FT_WordOrPhrase ARRAY[FT_MaxArrayLength];
      DECLARE strt FullText_Token ARRAY[FT_MaxArrayLength];
      DECLARE strt_termid INTEGER;

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

      SET ret = CAST(ARRAY[] AS FT_WordOrPhrase
         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_SYN) = thes_name
            )
         SELECT ARRAY[TD.EXPR] AS EXPRarr1
         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 = ret || EXPRarr1;
      END FOR L1;
      RETURN ret ||
         CASE
            WHEN startingTerm IS NULL OR thes_name IS NULL THEN
               CAST(ARRAY[] AS FT_WordOrPhrase
                  ARRAY[FT_MaxArrayLength])
            ELSE
               ARRAY[startingTerm]
         END;
   END
```

Definitional Rules

1) *FT_ThesNameLength* is the implementation-defined 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 *GetPreferredTerms(CHARACTER VARYING, FT_WordOrPhrase)* takes the following input parameters:

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

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

3) *GetPreferredTerms(CHARACTER VARYING, FT_WordOrPhrase)* returns an empty array if either *startingTerm* 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 *startingTerm* 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 *startingTerm* **is** included in the result.

6) It is implementation-defined, whether a check is made to ensure that the language specified in *startingTerm.Language* is compatible with the thesaurus as specified by *thes_name,* and if so, what kind of condition is raised in case of a language incompatibility.

6.14 FT_RelatedTerm Type and Routines

6.14.1 FT_RelatedTerm Type

Purpose

FT_RelatedTerm values 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
   AS (
      thesaurus CHARACTER VARYING(FT_ThesNameLength),
      startingTerm FT_WordOrPhrase
   )
   INSTANTIABLE
   NOT FINAL

   OVERRIDING METHOD Contains(text FullText)
      RETURNS BOOLEAN,

   OVERRIDING METHOD StrctPattern_to_FT_Pattern()
      RETURNS FT_Pattern,

   METHOD FT_RelatedTerm
      (thes_name CHARACTER VARYING(FT_ThesNameLength),
       strt FT_WordOrPhrase)
      RETURNS FT_RelatedTerm
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      CALLED ON NULL INPUT
```

Definitional Rules

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

Description

1) The *FT_RelatedTerm* type provides:

   a) an attribute *thesaurus*,
   b) an attribute *startingTerm*,
   c) a method *Contains(FullText)*,
   d) a method *StrctPattern_to_FT_Pattern()*,
   e) a method *FT_RelatedTerm(CHARACTER VARYING, FT_WordOrPhrase)*,
   f) a function *GetRelatedTerms(CHARACTER VARYING, FT_WordOrPhrase).*

2) For the purpose of this 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.

6.14.2 Contains Method

Purpose

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

Definition

```
CREATE METHOD Contains
   (text FullText)
   RETURNS BOOLEAN
   FOR FT_RelatedTerm
   BEGIN
      DECLARE RltdArray FT_WordOrPhrase ARRAY[FT_MaxArrayLength];
      DECLARE result BOOLEAN;

      SET RltdArray = GetRelatedTerms(SELF.thesaurus,
            SELF.startingTerm);
      SET result = NEW FT_Any(RltdArray).Contains(text);

      RETURN (SELF.NOT_tag = result);
   END
```

Definitional Rules

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

Description

1) The method *Contains(FullText)* takes the following input parameters:

   a) a *FullText* value *text*.

2) Let *R* be the result of

```
NEW FT_Any(GetRelatedTerms(SELF.thesaurus, SELF.startingTerm)).
   Contains(text)
```

3) Case:

   a) If *SELF.NOT_tag* is unknown, then *Contains(FullText)* returns unknown.

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

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

6.14.3 StrctPattern_to_FT_Pattern Method

Purpose

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

Definition

```
CREATE METHOD StrctPattern_to_FT_Pattern()
   RETURNS FT_Pattern
   FOR FT_RelatedTerm
   BEGIN
      DECLARE result CHARACTER VARYING(FT_MaxPatternLength);

      IF SELF.NOT_tag IS UNKNOWN THEN
         RETURN CAST(NULL AS FT_Pattern)
      END IF;

      SET result = ’THESAURUS "’
            || SELF.thesaurus
            || ’" EXPAND RELATED TERM OF ’
            || CAST(SELF.startingTerm.StrctPattern_to_FT_Pattern()
                  AS CHARACTER VARYING(FT_MaxPatternLength));

      IF NOT SELF.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 *FT_Pattern* value.

Description

1) The method *StrctPattern_to_FT_Pattern()* has no input parameters.

2) *StrctPattern_to_FT_Pattern()* returns an *FT_Pattern* value of the form <Related_Term expansion> or NOT <Related_Term expansion>.

3) If *SELF*, *SELF.thesaurus*, or *SELF.startingTerm* is the null value or *SELF.NOT_tag* is unknown, then the result is the null value.

6.14.4 FT_RelatedTerm Method

Purpose

Return a specified *FT_RelatedTerm* value.

Definition

```
CREATE METHOD FT_RelatedTerm
   (thes_name CHARACTER VARYING(FT_ThesNameLength),
    strt FT_WordOrPhrase)
   RETURNS FT_RelatedTerm
   FOR FT_RelatedTerm
   RETURN SELF.thesaurus(thes_name).startingTerm(strt).
      NOT_tag(TRUE)
```

Definitional Rules

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

Description

1) The method *FT_RelatedTerm(CHARACTER VARYING, FT_WordOrPhrase)* takes the following input parameters:

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

6.14.5 GetRelatedTerms Function

Purpose

Get related terms from a thesaurus.

Definition

```
CREATE FUNCTION GetRelatedTerms
   (thes_name CHARACTER VARYING(FT_ThesNameLength),
    startingTerm FT_WordOrPhrase)
   RETURNS FT_WordOrPhrase ARRAY[FT_MaxArrayLength]
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT
   STATIC DISPATCH
   BEGIN
      DECLARE ret FT_WordOrPhrase ARRAY[FT_MaxArrayLength];
      DECLARE strt FullText_Token ARRAY[FT_MaxArrayLength];
      DECLARE strt_termid INTEGER;

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

      SET ret = CAST(ARRAY[] AS FT_WordOrPhrase
         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_REL) = thes_name
            )
         SELECT ARRAY[TD.EXPR] AS EXPRarr1
         FROM TERM_DICTIONARY TD, temp_related
         WHERE TD.TERMID = temp_related.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 = ret || EXPRarr1;
      END FOR L1;
      RETURN ret ||
         CASE
            WHEN startingTerm IS NULL OR thes_name IS NULL THEN
               CAST(ARRAY[] AS FT_WordOrPhrase
                  ARRAY[FT_MaxArrayLength])
            ELSE
               ARRAY[startingTerm]
         END;
   END
```

Definitional Rules

1) *FT_ThesNameLength* is the implementation-defined 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 *GetRelatedTerms(CHARACTER VARYING*, *FT_WordOrPhrase)* takes the following input parameters:

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

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

3) *GetRelatedTerms(CHARACTER VARYING, FT_WordOrPhrase)* returns an empty array either *startingTerm* 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 *startingTerm*, the term represented by the *Related_Term* value is included in the result.

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

6) It is implementation-defined, whether a check is made to ensure that the language specified in *startingTerm.Language* is compatible with the thesaurus as specified by *thes_name*, and if so, what kind of condition is raised in case of a language incompatibility.

6.15 FT_TopTerm Type and Routines

6.15.1 FT_TopTerm Type

Purpose

FT_TopTerm values 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
   AS (
      thesaurus CHARACTER VARYING(FT_ThesNameLength),
      startingTerm FT_WordOrPhrase
   )
   INSTANTIABLE
   NOT FINAL

   OVERRIDING METHOD Contains(text FullText)
      RETURNS BOOLEAN,

   OVERRIDING METHOD StrctPattern_to_FT_Pattern()
      RETURNS FT_Pattern,

   METHOD FT_TopTerm
      (thes_name CHARACTER VARYING(FT_ThesNameLength),
       strt FT_WordOrPhrase)
      RETURNS FT_TopTerm
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      CALLED ON NULL INPUT
```

Definitional Rules

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

Description

1) The *FT_TopTerm* type provides:

   a) an attribute *thesaurus*,
   b) an attribute *startingTerm*,
   c) a method *Contains(FullText)*,
   d) a method *StrctPattern_to_FT_Pattern()*,
   e) a method *FT_TopTerm(CHARACTER VARYING, FT_WordOrPhrase)*,
   f) a function *GetTopTerms(CHARACTER VARYING, FT_WordOrPhrase).*

2) For the purpose of this 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.

6.15.2 Contains Method

Purpose

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

Definition

```
CREATE METHOD Contains
   (text FullText)
   RETURNS BOOLEAN
   FOR FT_TopTerm
   BEGIN
      DECLARE TopArray FT_WordOrPhrase ARRAY[FT_MaxArrayLength];
      DECLARE result BOOLEAN;

      SET TopArray = GetTopTerms(SELF.thesaurus,
            SELF.startingTerm);
      SET result = NEW FT_Any(TopArray).Contains(text);

      RETURN (SELF.NOT_tag = result);
   END
```

Definitional Rules

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

Description

1) The method *Contains(FullText)* takes the following input parameters:

   a) a *FullText* value *text*.

2) Let *R* be the result of

```
NEW FT_Any(GetTopTerms(SELF.thesaurus,
   SELF.startingTerm)).Contains(text)
```

3) Case:

   a) If *SELF.NOT_tag* is unknown, then *Contains(FullText)* returns unknown.

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

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

6.15.3 StrctPattern_to_FT_Pattern Method

Purpose

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

Definition

```
CREATE METHOD StrctPattern_to_FT_Pattern()
   RETURNS FT_Pattern
   FOR FT_TopTerm
   BEGIN
      DECLARE result CHARACTER VARYING(FT_MaxPatternLength);

      IF SELF.NOT_tag IS UNKNOWN THEN
         RETURN CAST(NULL AS FT_Pattern)
      END IF;

      SET result = ’THESAURUS "’
            || SELF.thesaurus
            || ’" EXPAND TOP TERM OF ’
            || CAST(SELF.startingTerm.StrctPattern_to_FT_Pattern()
                  AS CHARACTER VARYING(FT_MaxPatternLength));

      IF NOT SELF.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 *FT_Pattern* value.

Description

1) The method *StrctPattern_to_FT_Pattern()* has no input parameters.

2) *StrctPattern_to_FT_Pattern()* returns an *FT_Pattern* value of the form <Top_Term expansion> or NOT <Top_Term expansion>.

3) If *SELF*, *SELF.thesaurus*, or *SELF.startingTerm* is the null value or *SELF.NOT_tag* is unknown, then the result is the null value.

6.15.4 FT_TopTerm Method

Purpose

Return a specified *FT_TopTerm* value.

Definition

```
CREATE METHOD FT_TopTerm
   (thes_name CHARACTER VARYING(FT_ThesNameLength),
    strt FT_WordOrPhrase)
   RETURNS FT_TopTerm
   FOR FT_TopTerm
   RETURN SELF.thesaurus(thes_name).startingTerm(strt).
      NOT_tag(TRUE)
```

Definitional Rules

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

Description

1) The method *FT_TopTerm(CHARACTER VARYING, FT_WordOrPhrase)* takes the following input parameters:

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

6.15.5 GetTopTerms Function

Purpose

Get top terms from a thesaurus.

Definition

```
CREATE FUNCTION GetTopTerms
   (thes_name CHARACTER VARYING(FT_ThesNameLength),
    startingTerm FT_WordOrPhrase)
   RETURNS FT_WordOrPhrase ARRAY[FT_MaxArrayLength]
   LANGUAGE SQL
   DETERMINISTIC
   CONTAINS SQL
   CALLED ON NULL INPUT
   STATIC DISPATCH
   BEGIN
      DECLARE ret FT_WordOrPhrase ARRAY[FT_MaxArrayLength];
      DECLARE strt FullText_Token ARRAY[FT_MaxArrayLength];
      DECLARE strt_termid INTEGER;

      SET thes_name = TRIM(BOTH ' ' FROM thes_name);
      SET strt = startingTerm.getWordArray();
      SET strt_termid =
         (SELECT TERMID
          FROM TERM_DICTIONARY
          WHERE EXPR.getWordArray() = strt
                AND TRIM(BOTH ' ' FROM THNAME_DIC) = thes_name
         );
      SET ret = CAST(ARRAY[] AS FT_WordOrPhrase
         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
                  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[TD.EXPR] AS EXPRarr1
         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 = ret || EXPRarr1;
      END FOR L1;
      RETURN ret;
   END
```

Definitional Rules

1) *FT_ThesNameLength* is the implementation-defined 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_WordOrPhrase)* takes the following input parameters:

   a) a CHARACTER VARYING value *thes_name*,
   b) an *FT_WordOrPhrase* value *startingTerm*.

2) *GetTopTerms(CHARACTER VARYING, FT_WordOrPhrase)* returns an array of *FT_WordOrPhrase* elements, which stands for a set of top terms.

3) *GetTopTerms(CHARACTER VARYING, FT_WordOrPhrase, INTEGER)* is equivalent to *GetBroaderTerms(CHARACTER VARYING, FT_WordOrPhrase)*, using *thes_name*, *startingTerm*, 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 *startingTerm is* **not** included in the result.

5) It is implementation-defined, whether a check is made to ensure that the language specified in *startingTerm.Language* is compatible with the thesaurus as specified by *thes_name*, and if so, what kind of condition is raised in case of a language incompatibility.

6.16 FT_IsAbout Type and Routines

6.16.1 FT_IsAbout Type

Purpose

FT_IsAbout values provide for the construction of search patterns stating a topic in form of a *FullText* value, and for testing whether a text is pertinent to this value.

Definition

```
CREATE TYPE FT_IsAbout
   UNDER FT_Primary
   AS (
      wrdorphr FT_WordOrPhrase
   )
   INSTANTIABLE
   NOT FINAL

   OVERRIDING METHOD Contains(text FullText)
      RETURNS BOOLEAN,

   OVERRIDING METHOD StrctPattern_to_FT_Pattern()
      RETURNS FT_Pattern,

   METHOD FT_IsAbout(wrdorphr FT_WordOrPhrase)
      RETURNS FT_IsAbout
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      CALLED ON NULL INPUT
```

Description

1) The *FT_IsAbout* type provides:

   a) an attribute *wrdorphr*,
   b) a method *Contains(FullText)*,
   c) a method *StrctPattern_to_FT_Pattern()*,
   d) a method *FT_IsAbout(FullText)*.

6.16.2 Contains Method

Purpose

Search a *FullText* value for an *FT_IsAbout* value.

Definition

```
CREATE METHOD Contains
   (text FullText)
   RETURNS BOOLEAN
   FOR FT_IsAbout
   BEGIN
      DECLARE result BOOLEAN;
      --
      -- !! See description
      --
      RETURN result;
   END
```

Description

1) The method *Contains(FullText)* takes the following input parameters:

   a) a *FullText* value *text*.

2) *Contains(FullText)* tests whether a given *FullText* item is pertinent to the *FT_WordOrPhrase* item of a given *FT_IsAbout* value. The result is subject to implementation-defined criteria of pertinence.

6.16.3 StrctPattern_to_FT_Pattern Method

Purpose

Convert an *FT_IsAbout* value to an *FT_Pattern* value.

Definition

```
CREATE METHOD StrctPattern_to_FT_Pattern()
   RETURNS FT_Pattern
   FOR FT_IsAbout
   BEGIN
      DECLARE result CHARACTER VARYING(FT_MaxPatternLength);

      IF SELF.NOT_tag IS UNKNOWN THEN
         RETURN CAST(NULL AS FT_Pattern)
      END IF;

      SET result = ’IS ABOUT ’
            || CAST(SELF.wrdorphr.StrctPattern_to_FT_Pattern()
                  AS CHARACTER VARYING(FT_MaxPatternLength));

      IF NOT SELF.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 *FT_Pattern* value.

Description

1) The method *StrctPattern_to_FT_Pattern()* has no input parameters.

2) *StrctPattern_to_FT_Pattern()* returns an *FT_Pattern* value of the form <about expansion> or NOT <about expansion>.

3) If *SELF* or *SELF*.wrdorphr is the null value or *SELF.NOT_tag* is unknown, then the result is the null value.

6.16.4 FT_IsAbout Method

Purpose

Return a specified *FT_IsAbout* value.

Definition

```
CREATE METHOD FT_IsAbout
   (wrdorphr FT_WordOrPhrase)
   RETURNS FT_IsAbout
   FOR FT_IsAbout
   RETURN SELF.wrdorphr(wrdorphr).NOT_tag(TRUE)
```

Description

1) The method *FT_IsAbout(FT_WordOrPhrase)* takes the following input parameters:

   a) a *FT_WordOrPhrase* value *wrdorphr*.

6.17 FT_Context Type and Routines

6.17.1 FT_Context Type

Purpose

FT_Context values represent context search patterns.

Definition

```
CREATE TYPE FT_Context
   UNDER FT_Primary
   AS (
      ArgArray FT_PhraseList ARRAY[FT_MaxArrayLength],
      du FullText_Token
   )
   INSTANTIABLE
   NOT FINAL

   OVERRIDING METHOD Contains(text FullText)
      RETURNS BOOLEAN,

   OVERRIDING METHOD StrctPattern_to_FT_Pattern()
      RETURNS FT_Pattern,

   METHOD FT_Context
      (Arg1 FT_PhraseList,
       Arg2 FT_PhraseList,
       Arg3 FT_PhraseList ARRAY[FT_MaxArrayLength],
       DistanceUnit FullText_Token)
      RETURNS FT_Context
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      CALLED ON NULL INPUT
```

Definitional Rules

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

Description

1) The *FT_Context* type provides:

   a) an attribute *ArgArray*,
   b) an attribute *du*,
   c) a method *Contains(FullText)*,
   d) a method *StrctPattern_to_FT_Pattern()*,
   e) a method *FT_Context(FT_PhraseList, FT_PhraseList, FT_PhraseList ARRAY, FullText_Token)*.

6.17.2 Contains Method

Purpose

Search a *FullText* value for an *FT_Context* value.

Definition

```
CREATE METHOD Contains
   (text FullText)
   RETURNS BOOLEAN
   FOR FT_Context
   BEGIN
      DECLARE result BOOLEAN;
      DECLARE ftl FullText ARRAY[FT_MaxArrayLength];
      DECLARE segno INTEGER;
      DECLARE argno INTEGER;

      IF SELF IS NULL THEN
         SET argno = CAST(NULL AS INTEGER);
      ELSEIF SELF.ArgArray IS NULL THEN
         SET argno = CAST(NULL AS INTEGER)
      ELSE
         SET argno = CARDINALITY(SELF.ArgArray);
      END IF;

      SET ftl = text.Segmentize(SELF.du);

      IF ftl IS NULL THEN
         SET segno = CAST(NULL AS INTEGER);
      ELSE
         SET segno = CARDINALITY(ftl);
      END IF;

      IF segno 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, SELF.ArgArray[1])
               UNION
             SELECT ind + 1, SELF.ArgArray[ind + 1]
             FROM   ContextTab
             WHERE  ind < argno
            ),
         Temp(BasI) AS
            (SELECT MAX(TTE.BasI)
             FROM (VALUES(1) UNION
               SELECT
               (SELECT MIN(TTU.BasI)
                FROM (VALUES(3) UNION
                  SELECT CASE ca.Contains(seg)
                            WHEN FALSE THEN 1
                            WHEN TRUE  THEN 3
                            ELSE            2
                         END
                  FROM ContextTab ct(ind, ca)) AS TTU(BasI))
            FROM SegTab st(ind, seg)) AS TTE(BasI)
         )
         SELECT ARRAY[FALSE, TRUE, UNKNOWN][BasI]
         FROM Temp
      );
   END IF;
   RETURN (SELF.NOT_tag = result);
END
```

Definitional Rules

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

Description

1) The method *Contains(FullText)* takes the following input parameters:

   a) a *FullText* value *text*.

2) Case:

   a) If either *text.Segmentize(SELF.du)* or *SELF* or *SELF.ArgArray* is the null value, then the result of *Contains(FullText)* is unknown.

   b) Otherwise, let *n* be the number of elements of *SELF.ArgArray*, and for *i* ranging from 1 to *n*, let CA_i be the elements of *SELF.ArgArray*. Depending on the distance unit *SELF.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* values representing these sentences (paragraphs).

      Case:

      i) If there exists some SEG_j, such that the result of

         CA_i`.Contains(`SEG_j`)`

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

      i) If for every SEG_j, such that the result of

         CA_i`.Contains(`SEG_j`)`

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

      iii) Otherwise, let *R* be unknown.

3) *Contains(FullText)* returns:

   Case:

   a) unknown, if *SELF.NOT_tag* is unknown.

b) NOT *R*, if *SELF*.*NOT_tag* is <u>false</u>.

c) Otherwise, *R*.

6.17.3 StrctPattern_to_FT_Pattern Method

Purpose

Convert an *FT_Context* value to an *FT_Pattern* value.

Definition

```
CREATE METHOD StrctPattern_to_FT_Pattern()
   RETURNS FT_Pattern
   FOR FT_Context
   BEGIN
      DECLARE result CHARACTER VARYING(FT_MaxPatternLength);
      DECLARE i INTEGER;
      DECLARE n INTEGER;

      IF SELF.ArgArray IS NULL THEN
         RETURN NULL;
      ELSEIF SELF.NOT_tag IS UNKNOWN THEN
         RETURN NULL;
      END IF;

      SET n = CARDINALITY(SELF.ArgArray);
      SET result =
         CAST(SELF.ArgArray[1].StrctPattern_to_FT_Pattern()
           AS CHARACTER VARYING(FT_MaxPatternLength)
         || 'IN SAME '
         || TRIM(BOTH ' ' FROM SELF.du)
         ||' AS ' ||
         CAST(SELF.ArgArray[2].StrctPattern_to_FT_Pattern()
           AS CHARACTER VARYING(FT_MaxPatternLength));

      SET i = 3;

      L1: WHILE (n >= i) DO
            SET result = result || ' AND '||
             CAST(SELF.ArgArray[i].StrctPattern_to_FT_Pattern()
               AS CHARACTER VARYING(FT_MaxPatternLength));
         SET i = i + 1;
      END WHILE L1;

      IF NOT SELF.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 *FT_Pattern* value.

Description

1) The method *StrctPattern_to_FT_Pattern()* has no input parameters.

2) *StrctPattern_to_FT_Pattern()* returns an *FT_Pattern* value of the form <context condition> or NOT <context condition>.

3) The result is the null value in the following cases:

   a) *SELF* or *SELF.ArgArray* is the null value or *SELF.NOT_tag* is unknown.

   b) For some element *E* of *SELF.ArgArray*, *E.StrctPattern_to_FT_Pattern()* is the null value.

6.17.4 FT_Context Method

Purpose

Return a specified *FT_Context* value.

Definition

```
CREATE METHOD FT_Context
   (Arg1 FT_PhraseList,
    Arg2 FT_PhraseList,
    Arg3 FT_PhraseList ARRAY[FT_MaxArrayLength],
    DistanceUnit FullText_Token)
   RETURNS FT_Context
   FOR FT_Context
   RETURN SELF.
      ArgArray(ARRAY[Arg1, Arg2] || Arg3).du(DistanceUnit).
      NOT_tag(TRUE)
```

Definitional Rules

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

Description

1) The method *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* values,
   d) a *FullText_Token* value *DistanceUnit*.

2) All arguments may be the null value.

6.18 FT_ParExpr Type and Routines

6.18.1 FT_ParExpr Type

Purpose

FT_ParExpr provides for the construction of *FT_Term* patterns as *FT_Primary* values of the type *FT_ParExpr*, for searching occurrences of *FT_ParExpr* patterns in *FullText* items, and for turning *FT_ParExpr* values into equivalent *FT_Pattern* values.

Definition

```
CREATE TYPE FT_ParExpr
   UNDER FT_Primary
   AS (
      Body FT_Expr
   )
   INSTANTIABLE
   NOT FINAL

   OVERRIDING METHOD Contains(text FullText)
      RETURNS BOOLEAN,

   OVERRIDING METHOD StrctPattern_to_FT_Pattern()
      RETURNS FT_Pattern,

   METHOD FT_ParExpr(expr FT_Expr)
      RETURNS FT_ParExpr
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      CALLED ON NULL INPUT
```

Description

1) The *FT_ParExpr* type provides:

   a) an attribute *Body*,
   b) a method *Contains(FullText)*,
   c) a method *StrctPattern_to_FT_Pattern()*,
   d) a method *FT_ParExpr(FT_Expr)*.

6.18.2 Contains Method

Purpose

Search a *FullText* value for an *FT_ParExpr* value.

Definition

```
CREATE METHOD Contains
   (text FullText)
   RETURNS BOOLEAN
   FOR FT_ParExpr
   RETURN SELF.Body.Contains(text)
```

Description

1) The method *Contains(FullText)* takes the following input parameters:

   a) a *FullText* value *text*.

2) The result of *SELF.Contains(text)* is the result of *SELF.Body.Contains(text)*.

6.18.3 StrctPattern_to_FT_Pattern Method

Purpose

Convert an *FT_ParExpr* value to an *FT_Pattern* value.

Definition

```
CREATE METHOD StrctPattern_to_FT_Pattern()
   RETURNS(FT_Pattern)
   FOR FT_ParExpr
   BEGIN
      DECLARE result CHARACTER VARYING(FT_MaxPatternLength);

      IF SELF.NOT_tag IS UNKNOWN THEN
         RETURN CAST(NULL AS FT_Pattern);
      END IF;

      SET result = ’(’|| CAST(SELF.Body.StrctPattern_to_FT_Pattern()
         AS CHARACTER VARYING(FT_MaxPatternLength)) || ’)’;

      IF NOT SELF.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 *FT_Pattern* value.

Description

1) The method *StrctPattern_to_FT_Pattern()* has no input parameters.

2) *StrctPattern_to_FT_Pattern()* returns an *FT_Pattern* value of the form <left paren> <search expression> <right paren> except for the following cases:

   a) If *SELF* is the null value or *SELF.NOT_tag* is the null value, then the result is the null value.

   b) If *SELF.Body.StrctPattern_to_FT_Pattern()* is the null value, then the result is the null value.

6.18.4 FT_ParExpr Method

Purpose

Return a specified *FT_ParExpr* value.

Definition

```
CREATE METHOD FT_ParExpr
   (expr FT_Expr)
   RETURNS FT_ParExpr
   FOR FT_ParExpr
   RETURN SELF.Body(expr).NOT_tag(TRUE)
```

Description

1) The method *FT_ParExpr(FT_Expr)* takes the following input parameters:

   a) an *FT_Expr* value *expr*.

6.19 FT_Term Type and Routines

6.19.1 FT_Term Type

Purpose

FT_Term values 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
   AS (
      ConjunctsArray FT_Primary ARRAY[FT_MaxArrayLength]
   )
   INSTANTIABLE
   NOT FINAL

   METHOD Contains(text FullText)
      RETURNS BOOLEAN
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      CALLED ON NULL INPUT,

   METHOD StrctPattern_to_FT_Pattern()
      RETURNS FT_Pattern
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      CALLED ON NULL INPUT,

   METHOD FT_Term
      (pArray FT_Primary ARRAY[FT_MaxArrayLength])
      RETURNS FT_Term
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      CALLED ON NULL INPUT
```

Definitional Rules

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

Description

1) The *FT_Term* type provides:

   a) an attribute *ConjunctsArray*,
   b) a method *Contains(FullText)*,
   c) a method *StrctPattern_to_FT_Pattern()*,
   d) a method *FT_Term(FT_Primary ARRAY)*.

6.19.2 Contains Method

Purpose

Search a *FullText* value for an *FT_Term* value.

Definition

```
CREATE METHOD Contains
   (text FullText)
   RETURNS BOOLEAN
   FOR FT_Term
   BEGIN
      DECLARE i INTEGER ;
      DECLARE result BOOLEAN;

      IF SELF IS NULL THEN
         RETURN UNKNOWN;
      ELSEIF SELF.ConjunctsArray IS NULL THEN
         RETURN UNKNOWN;
      END IF;
      SET i = 1 ;
      SET result = TRUE;
   L1: WHILE (i <= CARDINALITY(SELF.ConjunctsArray))
            AND (result IS TRUE OR result IS UNKNOWN) DO
         SET result = result
               AND SELF.ConjunctsArray[i].Contains(text);

         SET i = i + 1;
      END WHILE L1;
      RETURN result;
   END
```

Description

1) The method *Contains(FullText)* takes the following input parameters:

   a) a *FullText* value *text*.

2) The result of *Contains(FullText)* is:

   Case:

   a) unknown, if *SELF* or *SELF.ConjunctsArray* is the null value.

   b) true, if for all *FT_Primary* elements *P* of *SELF.ConjunctsArray*

      `P.Contains(text)`

      returns true.

   c) false, if at least one *FT_Primary* element *P* of *SELF.ConjunctsArray* is such that

      `P.Contains(text)`

      returns false.

   d) Otherwise, unknown.

6.19.3 StrctPattern_to_FT_Pattern Method

Purpose

Convert an *FT_Term* value to an *FT_Pattern* value.

Definition

```
CREATE METHOD StrctPattern_to_FT_Pattern()
   RETURNS FT_Pattern
   FOR FT_Term
   BEGIN
      DECLARE i INTEGER;
      DECLARE result CHARACTER VARYING(FT_MaxPatternLength);

      SET i = 1;
      SET result = ’’;

   L1: WHILE(i <= CARDINALITY(SELF.ConjunctsArray)) DO
         SET result = result
            || CAST(
               SELF.ConjunctsArray[i].StrctPattern_to_FT_Pattern()
                  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 *FT_Pattern* value.

Description

1) The method *StrctPattern_to_FT_Pattern()* has no input parameters.

2) *StrctPattern_to_FT_Pattern()* returns an *FT_Pattern* value of the form <search term> except for the following cases:

   a) If *SELF.ConjunctsArray* is empty, the result is represented by an empty string.

   b) If *SELF* or *SELF.ConjunctsArray* is the null value, then the result is the null value.

   c) If for any element *E* of *SELF.ConjunctsArray*, *E.StrctPattern_to_FT_Pattern()* is the null value, then the result is the null value.

6.19.4 FT_Term Method

Purpose

Return a specified *FT_Term* value.

Definition

```
CREATE METHOD FT_Term
   (pArray FT_Primary ARRAY[FT_MaxArrayLength])
   RETURNS FT_Term
   FOR FT_Term
   RETURN SELF.ConjunctsArray(pArray)
```

Definitional Rules

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

Description

1) The method *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 19 - The definition of *FT_Term* values is intentionally more general than the definition of the corresponding <search term>s.

6.20 FT_Expr Type and Routines

6.20.1 FT_Expr Type

Purpose

FT_Expr values 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
   AS (
      DisjunctsArray FT_Term ARRAY[FT_MaxArrayLength]
   )
   INSTANTIABLE
   NOT FINAL

   METHOD Contains(text FullText)
      RETURNS BOOLEAN
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      CALLED ON NULL INPUT,

   METHOD StrctPattern_to_FT_Pattern()
      RETURNS FT_Pattern
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      CALLED ON NULL INPUT,

   METHOD FT_Expr
      (tArray FT_Term ARRAY[FT_MaxArrayLength])
      RETURNS FT_Expr
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      CALLED ON NULL INPUT
```

Definitional Rules

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

Description

1) The *FT_Expr* type provides:

   a) an attribute *DisjunctsArray*,
   b) a method *Contains(FullText)*,
   c) a method *StrctPattern_to_FT_Pattern()*,
   d) a method *FT_Expr(FT_Term ARRAY)*,

6.20.2 Contains Method

Purpose

Search a *FullText* value for an *FT_Expr* value.

Definition

```
CREATE METHOD Contains
   (text FullText)
   RETURNS BOOLEAN
   FOR FT_Expr
   BEGIN
      DECLARE i INTEGER ;
      DECLARE result BOOLEAN;

      IF SELF IS NULL THEN
         RETURN UNKNOWN;
      ELSEIF SELF.DisjunctsArray IS NULL THEN
         RETURN UNKNOWN;
      END IF;

      SET i = 1 ;
      SET result = FALSE;
   L1: WHILE (i <= CARDINALITY(SELF.DisjunctsArray))
            AND (result IS FALSE OR result IS UNKNOWN) DO
         SET result = result
               OR SELF.DisjunctsArray[i].Contains(text);
         SET i = i + 1;
      END WHILE L1;
      RETURN result;
   END
```

Description

1) The method *Contains(FullText)* takes the following input parameters:

   a) a *FullText* value *text*.

2) The result of *Contains(FullText)* is:

   Case:

   a) unknown, if *SELF* or *SELF.DisjunctsArray* is the null value.

   b) true, if for at least one *FT_Term* element *T* of *SELF.DisjunctsArray*

      `T.Contains(text)`

      returns true.

   c) false, if for all *FT_Term* elements *T* of *SELF.DisjunctsArray*

      `T.Contains(text)`

      returns false.

   d) Otherwise, unknown.

6.20.3 StrctPattern_to_FT_Pattern Method

Purpose

Convert an *FT_Expr* value to an *FT_Pattern* value.

Definition

```
CREATE METHOD StrctPattern_to_FT_Pattern()
   RETURNS FT_Pattern
   FOR FT_Expr
   BEGIN
      DECLARE i INTEGER;
      DECLARE result CHARACTER VARYING(FT_MaxPatternLength);

      IF SELF IS NULL OR SELF.DisjunctsArray IS NULL THEN
         RETURN CAST(NULL AS FT_Pattern);
      END IF;

      SET i      = 1;
      SET result = '';

      L1: WHILE(i <= CARDINALITY(SELF.DisjunctsArray)) DO
         SET result = result
            || CAST(
               SELF.DisjunctsArray[i].StrctPattern_to_FT_Pattern()
                  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 *FT_Pattern* value.

Description

1) The method *StrctPattern_to_FT_Pattern()* has no input parameters.

2) *StrctPattern_to_FT_Pattern()* returns an *FT_Pattern* value of the form <search expression> except for the following cases:

   a) If *SELF.DisjunctsArray* is empty, the result is represented by an empty string.
   b) If *SELF* or *SELF.DisjunctsArray* is the null value, then the result is the null value.
   c) If for any element *E* of *SELF.DisjunctsArray*, *E.StrctPattern_to_FT_Pattern()* is the null value, then the result is the null value.

6.20.4 FT_Expr Method

Purpose

Return a specified *FT_Expr* value.

Definition

```
CREATE METHOD FT_Expr
   (tArray FT_Term ARRAY[FT_MaxArrayLength])
   RETURNS FT_Expr
   FOR FT_Expr
   RETURN SELF.DisjunctsArray(tArray)
   END
```

Definitional Rules

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

Description

1) The method *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 20 - The definition of *FT_Expr* values is intentionally more general than the definition of the corresponding <search expression>s.

6.21 FT_PhraseList Type and Routines

6.21.1 FT_PhraseList Type

Purpose

FT_PhraseList type provides facilities for the construction of a structured search pattern that represents a multiset 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 *FullText* value.

Definition

```
CREATE TYPE FT_PhraseList
   AS (
      Phrases FT_Phrase ARRAY[FT_MaxArrayLength]
   )
   INSTANTIABLE
   NOT FINAL

   METHOD Contains(text FullText)
      RETURNS BOOLEAN
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      CALLED ON NULL INPUT,

   METHOD StrctPattern_to_FT_Pattern()
      RETURNS FT_Pattern
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      CALLED ON NULL INPUT,

   METHOD FT_PhraseList
      (phrases FT_Phrase ARRAY[FT_MaxArrayLength])
      RETURNS FT_PhraseList
      SELF AS RESULT
      LANGUAGE SQL
      DETERMINISTIC
      CONTAINS SQL
      CALLED ON NULL INPUT
```

Definitional Rules

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

Description

1) The *FT_PhraseList* type provides:

   a) an attribute *Phrases,*
   b) a method *Contains(FullText),*
   c) a method *StrctPattern_to_FT_Pattern()*,
   d) a method *FT_PhraseList(FT_Phrase ARRAY)*.

6.21.2 Contains Method

Purpose

Search a *FullText* value for an *FT_PhraseList* value.

Definition

```
CREATE METHOD Contains
   (text FullText)
   RETURNS BOOLEAN
   FOR FT_PhraseList
   BEGIN
      DECLARE i INTEGER ;
      DECLARE result BOOLEAN;
      DECLARE TokArray FullText_Token ARRAY[FT_MaxArrayLength];
      DECLARE lenp INTEGER;
      DECLARE lent INTEGER;

      IF SELF IS NULL THEN
         SET lenp = CAST(NULL AS INTEGER);
      ELSE
         SET lenp = CARDINALITY(SELF.Phrases);
      END IF;

      SET TokArray = text.Tokenize();
      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, SELF.Phrases[1])
            UNION
          SELECT ind + 1, SELF.Phrases[ind + 1]
          FROM phrlTab
          WHERE ind < lenp
         ),
         Temp(BasI) AS
            (SELECT MAX(BasI)
             FROM (VALUES(1) UNION
               SELECT CASE phr.Contains(text)
                         WHEN FALSE THEN 1
                         WHEN TRUE  THEN 3
                         ELSE            2
                      END
               FROM phrlTab pt(ind, phr)) AS TT(BasI)
            )
         SELECT ARRAY[FALSE, UNKNOWN, TRUE][BasI] FROM Temp
      );
      END IF;
      RETURN result;
   END
```

Definitional Rules

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

Description

1) The method *Contains (FullText)* takes the following input parameters:

   a) a *FullText* value *text*.

2) The result of *Contains (FullText)* is:

   Case:

   a) false, if *SELF.Phrases* is empty or if for every element *P* of *SELF.Phrases* the result of

      ```
      P.Contains(text)
      ```

      is false.

   b) true, if there exists at least one element *P* of *SELF.Phrases* such that the result of

      ```
      P.Contains(text)
      ```

      is true.

   c) Otherwise, unknown,.

      In particular, this result is obtained if:

      i) Any of *text* or *text.Tokenize()* is the null value, and *SELF* or *SELF.Phrases* is the null value.

      ii) *text* or *text.Tokenize()* is the null value, but *SELF.Phrases* is an non-empty array.

      iii) *SELF* or *SELF.Phrases* is the null value, but *text.Tokenize()* is an non-empty array.

6.21.3 StrctPattern_to_FT_Pattern Method

Purpose

Convert an *FT_PhraseList* value to an *FT_Pattern* value.

Definition

```
CREATE METHOD StrctPattern_to_FT_Pattern()
   RETURNS FT_Pattern
   FOR FT_PhraseList
   BEGIN
      DECLARE i INTEGER;
      DECLARE result CHARACTER VARYING(FT_MaxPatternLength);
      DECLARE len INTEGER;

      IF SELF.Phrases IS NULL THEN
         RETURN CAST(NULL AS FT_Pattern);
      ELSE
         SET len = CARDINALITY(SELF.Phrases);
      END IF;

      SET i = 1;
      SET result = '(';

      L1: WHILE(i <= len) DO
         SET result = result
            || CAST(
               SELF.Phrases[i].StrctPattern_to_FT_Pattern()
                  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 *FT_Pattern* value.

Description

1) The method *StrctPattern_to_FT_Pattern()* has no input parameters.

2) *StrctPattern_to_FT_Pattern()* returns an *FT_Pattern* value of the form <text literal list> except for the following cases:

   a) If *SELF* or *SELF.Phrases* is the null value, then the result is the null value.

   b) If for any element *E* of *SELF.Phrases*, *E.StrctPattern_to_FT_Pattern()* is the null value, then the result is the null value.

6.21.4 FT_PhraseList Method

Purpose

Return a specified *FT_PhraseList* value.

Definition

```
CREATE METHOD FT_PhraseList
   (phrases FT_Phrase ARRAY[FT_MaxArrayLength])
   RETURNS FT_PhraseList
   FOR FT_PhraseList
   RETURN SELF.Phrases(phrases)
```

Definitional Rules

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

Description

1) The method *FT_PhraseList(FT_Phrase ARRAY)* takes the following input parameters:

   a) an array *phrases* with elements of type *FT_Phrase*.

7 FullText_Token Type

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_MaxTokenLength* is the implementation-dependent maximum length for the character representation of a *FullText_Token* value.

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 Full-Text Thesaurus Schema

8.1 Introduction

The only purpose of the SQL/MM Full-Text 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 *FT_THESAURUS*. The table definitions are as complete as the definitional power of ISO/IEC 9075 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 FT_THESAURUS Schema

Purpose

Create the schema that is to contain the base tables that underlie the SQL/MM Full-Text Thesaurus Schema.

Definition

```
CREATE SCHEMA FT_THESAURUS
   AUTHORIZATION FT_THESAURUS
```

8.3 TERM_DICTIONARY base table

Purpose

The TERM_DICTIONARY base table has one row for each term referenced in the SQL/MM Full-Text 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       FT_WordOrPhrase,
   THNAME_DIC CHARACTER VARYING(FT_ThesNameLength) NOT NULL,

   PRIMARY KEY (TERMID, THNAME_DIC)
   )
```

Definitional Rules

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

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 in a given thesaurus.

Definition

```
CREATE TABLE TERM_HIERARCHY
   (
   TERMID            INTEGER NOT NULL,
   NARROWER_TERMID   INTEGER NOT NULL,
   THNAME_HRR        CHARACTER VARYING(FT_ThesNameLength) NOT NULL,

   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-defined 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 in a given thesaurus.

Definition

```
CREATE TABLE TERM_SYNONYM
   (
   TERMID            INTEGER NOT NULL,
   SYNONYM_TERMID    INTEGER NOT NULL,
   PREFERRED_TERMID  INTEGER,
   THNAME_SYN        CHARACTER VARYING(FT_ThesNameLength) NOT NULL,

   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-defined 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 in a given thesaurus.

Definition

```
CREATE TABLE TERM_RELATED
   (
   TERMID            INTEGER NOT NULL,
   RELATED_TERMID    INTEGER NOT NULL,
   THNAME_REL        CHARACTER VARYING(FT_ThesNameLength) NOT NULL,


   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-defined maximum length for the character representation of a thesaurus name.

Description

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

9 SQL/MM Full-Text Information Schema

9.1 Introduction

The SQL/MM Full-Text Information Schema views are defined as being in a schema named *FT_INFORMATION_SCHEMA* enabling these views to be accessed in the same way as any other tables in any other schema. SELECT privilege on all of these views is granted to PUBLIC WITH GRANT OPTION so that they can be queried by any user and so that SELECT privilege can be further granted on views that reference these Information Schema views. No other privilege is granted on them so they cannot be updated.

An implementation may define objects that are associated with *FT_INFORMATION_SCHEMA* that are not defined in this Clause. An implementation may also add columns to tables that are defined in this Clause.

9.2 FT_FEATURES view

Function

Identify the optional features and the implementation-defined user-visible constants.

Definition

```
CREATE VIEW FT_FEATURES
      AS SELECT * FROM FT_DEFINITION_SCHEMA.FT_FEATURES
```

9.3 FT_Schemata view

Identify the schemata which include the descriptors of a complete set of types, methods, and functions that are necessary to support the functionality of this part of ISO/IEC 13249.

Definition

```
CREATE VIEW FT_SCHEMATA
   AS SELECT
      F.CATALOG_NAME,
      F.SCHEMA_NAME,
      S.SCHEMA_OWNER,
      S.DEFAULT_CHARACTER_SET_CATALOG,
      S.DEFAULT_CHARACTER_SET_SCHEMA,
      S.DEFAULT_CHARACTER_SET_NAME
   FROM FT_DEFINITION_SCHEMA.FT_SCHEMATA F,
      INFORMATION_SCHEMA.SCHEMATA S
   WHERE F.CATALOG_NAME = S.CATALOG_NAME AND
      F.SCHEMA_NAME = S.SCHEMA_NAME
```

Description

1) A value of SCHEMA_OWNER identifies the owner of schema identified by the CATALOG_NAME and SCHEMA_NAME values.

2) The character set identified by the DEFAULT_CHARACTER_SET_CATALOG, DEFAULT_CHARACTER_SET_SCHEMA, and DEFAULT_CHARACTER_SET_NAME values is the character set under which the types, methods, and functions of this part of ISO/IEC 13249 have been created that are included in the schema identified by the CATALOG_NAME and SCHEMA_NAME values.

3) The values of SCHEMA_NAME are the unqualified schema names of the schemata in the catalog which include the descriptors of a complete set of types, methods, and functions that are necessary to support the functionality of this part of ISO/IEC 13249.

4) *INFORMATION_SCHEMA.SCHEMATA* is defined in part 2 of ISO/IEC 9075.

10 SQL/MM Full-Text Definition Schema

10.1 Introduction

The only purpose of the SQL/MM Full-Text Definition Schema is to provide a data model to support the *FT_INFORMATION_SCHEMA* and to assist understanding.

The base tables of the SQL/MM Full-Text Definition Schema are defined as being in a schema named *FT_DEFINITION_SCHEMA*. The table definitions are as complete as the definitional power of ISO/IEC 9075 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.

10.2 FT_FEATURES base table

Function

Identify the supported features and the implementation-defined user-visible constants.

Definition

```
CREATE TABLE FT_FEATURES
    (
   FT_FEATURENAME        CHARACTER VARYING(256),
   FT_FEATUREVALUE       CHARACTER VARYING(256),

   CONSTRAINT FT_FEATURES_PRIMARY_KEY PRIMARY KEY(FT_FEATURENAME),
   CONSTRAINT FT_FEATURES_CHECK CHECK
      (CASE FT_FEATURENAME
         WHEN 'FT_MaxTextLength'
            THEN (0 < CAST(FT_FEATUREVALUE AS INTEGER))
         WHEN 'FT_MaxLanguageLength'
            THEN (0 < CAST(FT_FEATUREVALUE AS INTEGER))
         WHEN 'FT_DefaultLanguage'
            THEN (CHARACTER_LENGTH(FT_FEATUREVALUE) <=
               CAST ((SELECT FT_FEATUREVALUE FROM FT_Features
                  WHERE FT_FEATURENAME = 'FT_MaxLanguageLength')
               AS INTEGER))
         WHEN 'FT_ThesNameLength'
            THEN (0 < CAST(FT_FEATUREVALUE AS INTEGER))
         WHEN 'FT_ThesTermLength'
            THEN (0 < CAST(FT_FEATUREVALUE AS INTEGER))
         WHEN 'FT_MinRankValue'
            THEN (0.0 <= CAST(FT_FEATUREVALUE AS DOUBLE PRECISION))
         WHEN 'FT_MaxRankValue'
            THEN (0.0 <= CAST(FT_FEATUREVALUE AS DOUBLE PRECISION)) AND
               CAST ((SELECT FT_FEATUREVALUE FROM FT_Features
                  WHERE FT_FEATURENAME = 'FT_MinRankValue')
               AS DOUBLE PRECISION ) <=
               CAST ((SELECT FT_FEATUREVALUE FROM FT_Features
                  WHERE FT_FEATURENAME = 'FT_MaxRankValue')
               AS DOUBLE PRECISION )
         WHEN 'FT_ProximityCharacters'
            THEN (FT_FEATUREVALUE IN ('YES', 'NO'))
         WHEN 'FT_ProximityWords'
            THEN (FT_FEATUREVALUE IN ('YES', 'NO'))
         WHEN 'FT_ProximitySentences'
            THEN (FT_FEATUREVALUE IN ('YES', 'NO'))
         WHEN 'FT_ProximityParagraphs'
            THEN (FT_FEATUREVALUE IN ('YES', 'NO'))
         WHEN 'FT_ContextSentence'
            THEN (FT_FEATUREVALUE IN ('YES', 'NO'))
         WHEN 'FT_ContextParagraph'
            THEN (FT_FEATUREVALUE IN ('YES', 'NO'))

         ELSE FALSE
      END
      )
   )
```

Description

1) If the value of *FT_FEATURENAME* is ’FT_MaxTextLength’, then the value of

   `CAST(FT_FEATUREVALUE AS INTEGER)`

   is the supported maximum length (in characters) of the *Contents* attribute of a *FullText* value.

2) If the value of *FT_FEATURENAME* is ’FT_MaxLanguageLength’ , then the value of

   `CAST(FT_FEATUREVALUE AS INTEGER)`

   is the supported maximum length (in characters) of a language value.

3) If the value of *FT_FEATURENAME* is ’FT_DefaultLanguage’ , then the value of *FT_FEATUREVALUE* is the default langauge and the the value of

   `CHARACTER_LENGTH(FT_FEATUREVALUE)`

   is less than or equal to *FT_MaxLanguageLength*.

4) If the value of *FT_FEATURENAME* is ’FT_ThesNameLength’, then the value of

   `CAST(FT_FEATUREVALUE AS INTEGER)`

   is the maximum supported length (in characters) of a thesaurus name.

5) If the value of *FT_FEATURENAME* is ’FT_ThesTermLength’, then the value of

   `CAST(FT_FEATUREVALUE AS INTEGER)`

   is the maximum supported length (in characters) of a thesaurus term.

6) If the value of *FT_FEATURENAME* is ’FT_MinRankValue’, then the value of

   `CAST(FT_FEATUREVALUE AS DOUBLE PRECISION)`

   is the minimum value that can be returned by the rank method.

7) If the value of *FT_FEATURENAME* is ’FT_MaxRankValue’, then the value of

   `CAST(FT_FEATUREVALUE AS DOUBLE PRECISION)`

   is the maximum value that can be returned by the rank method.

8) If the value of *FT_FEATURENAME* is ’FT_ProximityCharacters’, then the values of *FT_FEATUREVALUE* have the following meanings:

   ’YES’ Distances in terms of the document unit CHARACTERS supported in <Proximity expansion>.

   ’NODistances in terms of the document unit CHARACTERS not supported in <Proximity expansion>.

9) If the value of *FT_FEATURENAME* is ’FT_ProximityWords’, then the values of*FT_FEATUREVALUE* have the following meanings:

’YES’ Distances in terms of the document unit WORDS supported in <Proximity expansion>.

’NO’ Distances in terms of the document unit WORDS not supported in <Proximity expansion>.

10) If the value of *FT_FEATURENAME* is ’FT_ContextSentence’, then the values of*FT_FEATUREVALUE* have the following meanings:

’YES’ Distances in terms of the document unit SENTENCE supported in <context condition>.

’NO’ Distances in terms of the document unit SENTENCE not supported in <context condition>.

11) If the value of *FT_FEATURENAME* is ’FT_ProximitySentences’, then the values of *FT_FEATUREVALUE* have the following meanings:

’YES’ Distances in terms of the document unit SENTENCES supported in <Proximity expansion>.

’NO’ Distances in terms of the document unit SENTENCES not supported in <Proximity expansion>.

12) If the value of *FT_FEATURENAME* is ’FT_ContextParagraph’, then the values of *FT_FEATUREVALUE* have the following meanings:

’YES’ Distances in terms of the document unit PARAGRAPH supported in <context condition>.

’NO’ Distances in terms of the document unit PARAGRAPH not supported in <context condition>.

13) If the value of *FT_FEATURENAME* is ’FT_ProximityParagraphs’, then the values of *FT_FEATUREVALUE* have the following meanings:

’YES’ Distances in terms of the document unit PARAGRAPHS supported in <Proximity expansion>.

’NO’ Distances in terms of the document unit PARAGRAPHS not supported in <Proximity expansion>.

10.3 FT_SCHEMATA base table

Function

Identify the schemata which include the descriptors of a complete set of types, methods, and functions that are necessary to support the functionality of this part of ISO/IEC 13249.

Definition

```
CREATE TABLE FT_SCHEMATA
   (
   CATALOG_NAME         INFORMATION_SCHEMA.SQL_IDENTIFIER,
   SCHEMA_NAME          INFORMATION_SCHEMA.SQL_IDENTIFIER,

   CONSTRAINT FT_SCHEMATA_PRIMARY_KEY
      PRIMARY KEY (CATALOG_NAME, SCHEMA_NAME)
   )
```

Description

1) All the values of CATALOG_NAME are the name of the catalog in which the schemata identified by SCHEMA_NAME are included.

2) The values of SCHEMA_NAME are the unqualified schema names of the schemata in the catalog which include the descriptors of a complete set of types, methods, and functions that are necessary to support the functionality of this part of ISO/IEC 13249.

3) *INFORMATION_SCHEMA.SQL_IDENTIFIER* is defined in part 2 of ISO/IEC 9075.

11 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.

For a successful completion code but with a warning, the first two characters of the SQLSTATE are equal to the SQLSTATE condition code class value for *warning* (defined in Subclause 22.1, "SQLSTATE" in Part 2 of ISO/IEC 9075) and third character of the SQLSTATE is ’H’.

Table 1 – SQLSTATE class and subclass values

Category	Condition	Class	Subcondition	Subclass
X	SQL/MM Full-Text	XX	invalid search expression	F01
X	SQL/MM Full-Text	XX	invalid language specification	F02

XX If the routine is implemented as an SQL-invoked routine, then the first two characters of the SQLSTATE are equal to the SQLSTATE condition code class for *SQL routine exception* (defined in Subclause 22.1, "SQLSTATE" in Part 2 of ISO/IEC 9075.

If the routine is implemented as an external routine, then the first two characters of the SQLSTATE are equal to the SQLSTATE condition code class for *external routine* exception (defined in Subclause 22.1, "SQLSTATE" in Part 2 of ISO/IEC 9075.

12 Conformance

12.1 Requirements for conformance

A conforming implementation shall support the following public user-defined types and routines:

1) the *FullText* user-defined type as defined in Subclause 5.1, "FullText Type and Routines" with the following:

    a) the attribute *Language* and its associated methods,
    b) the method *Contains(FT_Pattern)*,
    c) the method *Contains(CHARACTER VARYING)*,
    d) the method *Rank(FT_Pattern)*,
    e) the method *Rank(CHARACTER VARYING)*,
    f) the method *FullText(CHARACTER VARYING)*,
    g) the method *FullText(CHARACTER VARYING, CHARACTER VARYING)*,
    h) the cast function *FullText_to_Character(FullText)*.

2) the *FT_Pattern* user-defined type as defined in Subclause 5.3, "FT_Pattern Type and Routines".

A conforming implementation shall also support the views in the *FT_INFORMATION_SCHEMA* as defined in Clause 9, "SQL/MM Full-Text Information Schema".

All other user-defined types and routines defined in this part of ISO/IEC 13249 but not listed above are used only to define the semantics of the public user-defined types and routines. A conforming implementation need not support these additional user-defined types and routines for public use.

12.2 Claims of conformance

Claims of conformance to this part of ISO/IEC 13249 shall state:

1) The definitions for all elements and actions that this part of ISO/IEC 13049 specifies as implementation-defined.

Annex A
(informative)
Implementation-defined elements

This Annex references those features that are identified in the body of this part of ISO/IEC 13249 as implementation-defined.

The term implementation-defined is used to identify characteristics that may differ between implementations, but that shall be defined for each particular implementation.

1) Subclause 5.1.2, "Contains Methods":

    a) If *pattern* contains a pattern that meets one of the following conditions, then it is implementation-defined whether an exception condition is raised: *SQL/MM Full-Text - invalid search expression*:

        i) A pattern of the form <word> or <stemmed word> specifies a stop word.

        ii) A pattern of the form <phrase> or <stemmed phrase> contains only stop words, or contains leading or trailing stop words.

        iii) A pattern of the form <text literal list> contains only stop words.

2) Subclause 5.1.3, "Rank Methods":

    a) The value returned by the *Rank* method is an implementation-dependent *DOUBLE PRECISION* value constrained by implementation-defined minimum and maximum values.

3) Subclause 5.1.4, "Tokenize Method":

    a) Further details of the relationship between the input and output of the *Tokenize* method are implementation-defined.

4) Subclause 5.1.5, "TokenizePosition Method":

    a) *corrVal* is zero for the distance units ’WORDS’, ’SENTENCES’ and ’PARAGRAPHS’; its value is implementation-defined for distance unit ’CHARACTERS’.

    b) It is implementation-defined whether no stop words of *SELF.Contents*, all stop words of *SELF.Contents,* or all stop words of *SELF.Contents* other than leading and trailing stop words are effectively included in the result of *SELF.TokenizePosition(FullText_Token).* If stop words are included, then it is implementation-defined how they are effectively represented, provided their representation is such that the result of comparing any two stop words is true.

    c) Further details of the relationship between the input and output of the *TokenizePosition* method are implementation-defined.

5) Subclause 5.1.6, "Segmentize Method":

    a) Further details of the relationship between the input and output of the *Segmentize* method are implementation-defined.

6) Subclause 5.1.7, "TokenizeAndStem Method":

    a) Further details of the relationship between the input and output of the *TokenizeAndStem* method are implementation-defined.

7) Subclause 5.1.8, "TokenizePositionAndStem Method":

    a) It is implementation-defined whether no stop words of *SELF.Contents*, all stop words of *SELF.Contents*, or all stop words of *SELF.Contents* other than leading and trailing stop words are effectively included in the result of *SELF.TokenizePositionAndStem()*. If stop words are included, then it is implementation-defined how they are effectively represented, provided their representation is such that the result of comparing any two stop words is true.

    b) Further details of the relationship between the input and output of the *TokenizePositionAndStem* method are implementation-defined.

8) Subclause 5.3.1, "FT_Pattern Type":

    a) 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.

    b) It is implementation-defined whether a specific <word separator> character is needed between two consecutive <phrasepart representation>s.

    c) The details of the <language specification>, as well as the default language is implementation-defined.

    d) The document units supported are implementation-defined.

    e) The characters <thesaurus name character> that can be used to construct thesaurus names are implementation-defined.

9) Subclause 6.1.2, "Contains Method":

    a) It is implementation-defined whether an exception condition is raised: *SQL/MM Full-Text - invalid search expression*.

10) Subclause 6.4.2, "Contains Method":

    a) Let *TL* be the result of the invocation of *text.Tokenize()* and *TLE* be elements of *TL*, normalized in an implementation-defined way, and with leading and trailing blanks removed. Let *T* be *SELF.LitPart*, normalized in an implementation-defined way and with leading and trailing blanks removed.

11) Subclause 6.4.5, "Tokenize Method":

    a) *Tokenize()* normalizes *SELF.LitPart* in an implementation-defined way.

12) Subclause 6.5.2, "Contains Method":

   a) Let *TL* be the result of the invocation of *text.TokenizeAndStem()*. Let *TLE* be elements of *TL*, normalized and reduced to stems in an implementation-defined way, and with leading and trailing blanks removed. Let *T* be *SELF.LitPart*, normalized and reduced to stems in an implementation-defined way, and with leading and trailing blanks removed.

13) Subclause 6.5.4, "TokenizeAndStem Method":

   a) *TokenizeAndStem()* normalizes and stem-reduces *SELF.LitPart* in an implementation-defined way.

14) Subclause 6.6.2, "Contains Method":

   a) If the first element of *SELF.PhrasePart* or the last element of *SELF.PhrasePart* is a stop word, or all elements of *SELF.PhrasePart* are stop words, then it is implementation-defined whether an exception condition is raised: *SQL/MM Full-Text - invalid search expression.*

   b) Let *TL* be the result of the invocation of *text.TokenizePosition()* and *TLE* be elements of *TL*. Every *TLE* represents some word of *text* in an implementation-defined normalized way, with leading and trailing blanks removed. It is implementation-defined whether no stop word of *text*, all stop words of *text*, or all stop words of *text* except for leading and trailing stop words are represented by some *TLE*. If stop words are included, then it is implementation-defined how they are effectively represented, provided their representation is such that the result of comparing any two stop words is <u>true</u>.

   c) Let *TPL* be the result of *SELF.TokenizePosition ()* and let *TPLE* be the elements of *TPL*. Every *TPLE* represents some word of *SELF.PhrasePart* in an implementation-defined normalized way with leading and trailing blanks removed. It is implementation-defined whether no stop word of *SELF.PhrasePart,* all stop words of *SELF.PhrasePart*, or all stop words of *SELF.PhrasePart* except for leading and trailing stop words are represented by some *TPLE* in an implementation-defined way, provided stop word are dealt with in the same fashion by the *TokenizePosition* methods of the *FullText* and *FT_Phrase* types.

   d) It is implementation-defined whether the distance between two words *W1* and *W2* in a pattern of the form <phase> or <stemmed phrase> is exactly or at most one more than the number of consecutive stop words between *W1* and *W2*. In the latter case, the stop words effectively behave like optional words.

15) Subclause 6.6.5, "TokenizePosition Method":

   a) *TokenizePosition()* normalizes *SELF.PhrasePart* in an implementation-defined way.

16) Subclause 6.7.2, "Contains Method":

   a) If the first element of *SELF.PhrasePart* or the last element of *SELF.PhrasePart* is a stop word, or all elements of *SELF.PhrasePart* are stop words, then it is implementation-defined whether an exception condition is raised: *SQL/MM Full-Text - invalid search expression.*

   b) Let *TL* be the result of the invocation of *text.TokenizePositionAndStem()* and *TLE* be elements of *TL*. Every *TLE* represents some word of *text* reduced to its base reduced form and in an implementation-defined normalized way, with leading and trailing blanks removed. It is implementation-defined whether no stop word of *text*, all stop words of *text*, or all stop words of *text* except for leading and trailing stop words are represented by some *TLE*. If stop words are included, then it is implementation-defined how they are effectively represented, provided their representation is such that the result of comparing any two stop words is <u>true</u>.

   c) Let *TPL* be the result of *SELF.TokenizePositionAndStem()*. Every element *TPLE* of *TPL* represents some word of *SELF.PhrasePart* reduced to its base reduced form and represented in an implementation-defined normalized way, with leading and trailing blanks removed. It is implementation-defined whether no stop word of *SELF.PhrasePart*, all stop words of *SELF.PhrasePart*, or all stop words of *SELF.PhrasePart* except for leading and trailing stop words are represented by some *TPLE* in an implementation-defined way, provided stop word are dealt with in the same fashion by the *TokenizePositionAndStem* methods of the *FullText* and *FT_StemmedPhrase* types.

   d) It is implementation-defined whether the distance between two words *W1* and *W2* in a pattern of the form <phase> or <stemmed phrase> is exactly or at most one more than the number of consecutive stop words between *W1* and *W2*. In the latter case, the stop words effectively behave like optional words.

17) Subclause 6.9.4, "FT_Soundex Method":

   a) 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.

18) Subclause 6.9.5, "GetSoundsSimilar Function":

   a) 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.

19) Subclause 6.10.1, "FT_BroaderTerm Type"

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

20) Subclause 6.10.5, "GetBroaderTerms Function":

   a) It is implementation-defined, whether a check is made to ensure that the language specified in *startingTerm.Language* is compatible with the thesaurus as specified by *thes_name*, and if so, what kind of condition is raised in case of a language incompatibility.

21) Subclause 6.11.1, "FT_NarrowerTerm Type":

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

22) Subclause 6.11.5, "GetNarrowerTerms Function":

   a) It is implementation-defined, whether a check is made to ensure that the language specified in *startingTerm.Language* is compatible with the thesaurus as specified by *thes_name*, and if so, what kind of condition is raised in case of a language incompatibility.

23) Subclause 6.12.1, "FT_Synonym Type":

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

24) Subclause 6.12.5, "GetSynonymTerms Function":

   a) It is implementation-defined, whether a check is made to ensure that the language specified in *startingTerm.Language* is compatible with the thesaurus as specified by *thes_name*, and if so, what kind of condition is raised in case of a language incompatibility.

25) Subclause 6.13.1, "FT_PreferredTerm Type":

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

26) Subclause 6.13.2, "GetPreferredTerms Function":

    a) It is implementation-defined, whether a check is made to ensure that the language specified in *startingTerm.Language* is compatible with the thesaurus as specified by *thes_name*, and if so, what kind of condition is raised in case of a language incompatibility.

27) Subclause 6.14.1, "FT_RelatedTerm Type":

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

28) Subclause 6.14.2, "GetRelatedTerms Function":

    a) It is implementation-defined, whether a check is made to ensure that the language specified in *startingTerm.Language* is compatible with the thesaurus as specified by *thes_name*, and if so, what kind of condition is raised in case of a language incompatibility.

29) Subclause 6.15.1, "FT_TopTerm Type":

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

30) Subclause 6.15.2, "GetTopTerms Function":

    a) It is implementation-defined, whether a check is made to ensure that the language specified in *startingTerm.Language* is compatible with the thesaurus as specified by *thes_name*, and if so, what kind of condition is raised in case of a language incompatibility.

31) Subclause 6.16.2, "Contains Method":

    a) *Contains(FullText)* tests whether a given *FullTex*t item is pertinent to the *FT_WordOrPhrase* item of a given *FT_IsAbout* value. The result is subject to implementation-defined criteria of pertinence.

32) Subclause 7.1, "FullText_Token Type":

    a) The function *p* returns <u>*true*</u> if and only if the character string *VALUE* is a valid token. It is implementation-defined whether a character string is a valid token.

33) Subclause 8.3, "TERM_DICTIONARY base table":

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

34) Subclause 8.4, "TERM_HIERARCHY base table":

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

35) Subclause 8.5, "TERM_SYNONYM base table":

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

36) Subclause 8.6, "RELATED_TERM base table":

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

A.1 Implementation-defined Meta-variables

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

2) *FT_MaxLanguageLength* is the implementation-defined maximum length for the character representation of a language specification.

3) *FT_DefaultLanguage* is an implementation-defined character string literal which denotes the implementation-defined default language. The length of *FT_DefaultLanguage* does not exceed *FT_MaxLanguageLength*.

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

Annex B
(informative)
Implementation-dependent elements

This Annex references those places where this part of ISO/IEC 13249 states explicitly that the actions of a conforming implementation are implementation-dependent.

The term implementation-dependent is used to identify characteristics that may differ between implementations, but that are not necessarily specified for any particular implementation.

1) Subclause 5.1.3, "Rank Methods":

   a) The value returned by the *Rank* method is an implementation-dependent *DOUBLE PRECISION* value constrained by implementation-defined minimum and maximum values.

2) Subclause 6.6.5, "TokenizePosition Method":

   a) *TokenizePosition()* normalizes *SELF.PhrasePart* in an implementation-defined way. In addition, it is implementation-dependent whether stop words are effectively included in the result, and if so, how they are represented. However, a conforming implementation must treat stop words in this method and in the *FullText* method *TokenizePosition(FullText_Token)* in the same way.

3) Subclause 6.7.4, "TokenizePositionAndStem Method":

   a) *TokenizePositionAndStem()* normalizes and stem-reduces the sequence of words represented by *SELF.PhrasePart* in an implementation-dependent way. In addition, it is implementation-dependent whether stop words are effectively included in the result, and if so, how they are represented.

4) Subclause 6.9.9, "GetSoundsSimilar Function":

   a) *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. The mechanism for generating this array, taking into account the language as specified in *spoken.Language*, is implementation-dependent.

B.1 Implementation-dependent Meta-variables

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

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

3) *FT_MaxTokenLength* is the implementation-dependent maximum length for the character representation of a *FullText_Token* value.

Index