



**INTERNATIONAL STANDARD ISO/IEC 13249-2:2000**  
**TECHNICAL CORRIGENDUM 1**

Published 2003-08-15

INTERNATIONAL ORGANIZATION FOR STANDARDIZATION • МЕЖДУНАРОДНАЯ ОРГАНИЗАЦИЯ ПО СТАНДАРТИЗАЦИИ • ORGANISATION INTERNATIONALE DE NORMALISATION  
INTERNATIONAL ELECTROTECHNICAL COMMISSION • МЕЖДУНАРОДНАЯ ЭЛЕКТРОТЕХНИЧЕСКАЯ КОМИССИЯ • COMMISSION ÉLECTROTECHNIQUE INTERNATIONALE

# **Information technology — Database languages — SQL multimedia and application packages —**

## **Part 2: Full-Text**

### **TECHNICAL CORRIGENDUM 1**

*Technologies de l'information — Langages de bases de données — Multimédia SQL et paquetages  
d'application*

*Partie 2: Texte complet*

*RECTIFICATIF TECHNIQUE 1*

Technical Corrigendum 1 to ISO/IEC 13249-2:2000 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 32, *Data management and interchange*.

#### **Statement of purpose of rationale:**

A statement indicating the rationale for each change to ISO/IEC 13249-2:2000(E) is included. This is to inform the users of that standard as to the reason why it was judged necessary to change the original wording. In many cases the reason is editorial or to clarify the wording; in some cases it is to correct an error or an omission in the original wording.

#### **Notes on numbering:**

Where this Corrigendum introduces new Definitional Rules and Descriptions, the new rules have been numbered as follows:

Rules inserted between, for example, Rules 7) and 8) are numbered 7.1), 7.2), etc. [or 7) a.1), 7) a.2), etc.]. Those inserted before Rule 1) are numbered 0.1), 0.2), etc.

Where this Corrigendum introduces new subclauses, the new subclauses have been numbered as follows:

Subclauses inserted between, for example, subclause 4.3.2 and 4.3.3 are numbered 4.3.2a, 4.3.2b, etc.

Those inserted before, for example, 4.3.1 are numbered 4.3.0, 4.3.0a, etc.

## Contents

Page

|                                                                     |    |
|---------------------------------------------------------------------|----|
| Global changes .....                                                | 4  |
| 4.3 Text scoring facilities .....                                   | 4  |
| 4.4.2 Treatment of stop words .....                                 | 4  |
| 4.5 Word normalization .....                                        | 5  |
| 4.6 Types and routines provided by this part of ISO/IEC 13249 ..... | 5  |
| 4.6.1 Types and routines intended for public use .....              | 5  |
| 5.1.1 FullText Type .....                                           | 6  |
| 5.1.3 Rank Methods .....                                            | 7  |
| 5.1.4 Tokenize Method .....                                         | 8  |
| 5.1.5 TokenizePosition Method .....                                 | 8  |
| 5.1.7 TokenizeAndStem Method .....                                  | 8  |
| 5.1.8 TokenizePositionAndStem Method .....                          | 8  |
| 5.1.9 FullText Methods .....                                        | 9  |
| 5.3.1 FT_Pattern Type .....                                         | 9  |
| 6.1.1 FT_Any Type .....                                             | 10 |
| 6.1.3 FT_Any Method .....                                           | 10 |
| 6.4.1 FT_TextLiteral Type .....                                     | 10 |
| 6.4.2 Contains Method .....                                         | 11 |
| 6.4.5 Tokenize Method .....                                         | 12 |
| 6.4.7 FT_TextLiteral Methods .....                                  | 12 |
| 6.5.1 FT_StemmedWord Type .....                                     | 12 |
| 6.5.2 Contains Method .....                                         | 13 |
| 6.5.4 TokenizeAndStem Method .....                                  | 14 |
| 6.5.5 FT_StemmedWord Methods .....                                  | 14 |
| 6.6.1 FT_Phrase Type .....                                          | 15 |
| 6.6.2 Contains Method .....                                         | 15 |
| 6.6.5 TokenizePosition Method .....                                 | 17 |
| 6.6.6 FT_Phrase Methods .....                                       | 17 |
| 6.7.1 FT_StemmedPhrase Type .....                                   | 18 |
| 6.7.2 Contains Method .....                                         | 18 |
| 6.7.4 TokenizePositionAndStem Method .....                          | 20 |
| 6.7.5 FT_StemmedPhrase Methods .....                                | 20 |
| 6.8.1 FT_Proxi Type .....                                           | 21 |
| 6.8.4 FT_Proxi Method .....                                         | 21 |
| 6.9.1 FT_Soundex Type .....                                         | 21 |
| 6.9.4 FT_Soundex Method .....                                       | 22 |
| 6.9.5 GetSoundsSimilar Function .....                               | 22 |
| 6.10.1 FT_BroaderTerm Type .....                                    | 22 |
| 6.10.4 FT_BroaderTerm Method .....                                  | 22 |
| 6.10.5 GetBroaderTerms Function .....                               | 23 |
| 6.11.1 FT_NarrowerTerm Type .....                                   | 24 |
| 6.11.4 FT_NarrowerTerm Method .....                                 | 24 |
| 6.11.5 GetNarrowerTerms Function .....                              | 25 |
| 6.12.1 FT_Synonym Type .....                                        | 26 |
| 6.12.4 FT_Synonym Method .....                                      | 26 |
| 6.13.1 FT_PREFERREDTerm Type .....                                  | 27 |
| 6.13.4 FT_PREFERREDTerm Method .....                                | 27 |
| 6.13.5 GetPreferredTerms Function .....                             | 27 |
| 6.14.1 FT_RelatedTerm Type .....                                    | 28 |
| 6.14.4 FT_RelatedTerm Method .....                                  | 28 |
| 6.15.1 FT_TopTerm Type .....                                        | 29 |

|         |                                                      |    |
|---------|------------------------------------------------------|----|
| 6.15.4  | FT_TopTerm Method .....                              | 29 |
| 6.15.5  | GetTopTerms Function .....                           | 29 |
| 6.16.1  | FT_IsAbout Type .....                                | 30 |
| 6.16.4  | FT_IsAbout Method.....                               | 31 |
| 6.17.1  | FT_Context Type .....                                | 31 |
| 6.17.4  | FT_Context Method .....                              | 31 |
| 6.18.1  | FT_ParExpr Type.....                                 | 31 |
| 6.18.4  | FT_ParExpr Method .....                              | 32 |
| 6.19.1  | FT_Term Type.....                                    | 32 |
| 6.19.4  | FT_Term Method .....                                 | 32 |
| 6.20.1  | FT_Expr Type .....                                   | 32 |
| 6.20.4  | FT_Expr Method .....                                 | 33 |
| 6.21.1  | FT_PhraseList Type .....                             | 33 |
| 6.21.4  | FT_PhraseList Method.....                            | 33 |
| 10.2    | FT_FEATURES base table.....                          | 33 |
| 11      | Status Codes .....                                   | 35 |
| 12.1    | Requirements for conformance.....                    | 35 |
| Annex A | (informative) Implementation-defined elements .....  | 35 |
| A.1     | Implementation-defined Meta-variables .....          | 37 |
| Annex B | (informative) Implementation-dependent elements..... | 38 |

## Global changes

1. *Rationale: Shorten Information Schema name to be 18 characters in length.*

Globally replace "FT\_INFORMATION\_SCHEMA" with "FT\_INFORMTN\_SCHEMA".

### 4.2.4.2 Expansion facility patterns

1. *Rationale: Soundex facility cannot be used with <phrase>.*

Replace the bulleted list with:

- 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.

## 4.3 Text scoring facilities

1. *Rationale: The term rank is not consistent with its function.*

Replace Subclause 4.3 with:

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 *Score* method for the *FullText* type. This method takes any pattern that can also be used for text identification as in the following example:

```
firstSample.Score(' ( "Standard", "International", "method" ) ')
```

The *Score* 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 score value is implementation-defined.

### 4.4.2 Treatment of stop words

1. *Rationale: Fix distance between two tokens.*

Delete third bullet item that begins with "It is implementation-defined whether the distance separating two words *W1* and *W2*".

## 4.5 Word normalization

1. *Rationale: The term rank is not consistent with its function.*

Replace Subclause 4.5 with:

When evaluating *Score* 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 scored by the *Score* 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

1. *Rationale: The term rank is not consistent with its function.*

Replace Item 2 with:

2. definition oriented types and routines that are used to capture the semantics of the Category 1 types and routines, except for *Score* methods (see Subclause 5.3.1, "FT\_Pattern Type").

### 4.6.1 Types and routines intended for public use

1. *Rationale: The term rank is not consistent with its function.*

Replace Subclause 4.6.1 with:

The following types and routines are intended for public use:

- *FullText* type with
  - methods *Language*,
  - methods *Contains*,
  - methods *Score*,
  - methods *FullText*,
  - function *FullText\_to\_Character* to cast a *FullText* value into a character string,
- *FT\_Pattern* type.

### 5.1.1 FullText Type

#### 1. Rationale: Correct constructor method definitions.

Replace the <original method specification>s that begins with "METHOD FullText" in the Definition with:

```

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

CONSTRUCTOR 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

```

#### 2. Rationale: Support SQL-data dependent implementations of Contains() and Rank() methods.

Replace the <original method specification>s that begins with "METHOD Contains" in the Definition with:

```

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

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

```

#### 3. Rationale: The term rank is not consistent with its function and support SQL-data dependent implementations of Contains() and Rank() methods.

Replace the <original method specification>s that begins with "METHOD Rank" in the Definition with:

```

METHOD Score
  (pattern FT_Pattern)
  RETURNS DOUBLE PRECISION
  LANGUAGE SQL
  FT_ScoreDeterminism
  READS SQL DATA
  CALLED ON NULL INPUT,

METHOD Score
  (pattern CHARACTER VARYING (FT_MaxPatternLength))
  RETURNS DOUBLE PRECISION
  LANGUAGE SQL
  FT_ScoreDeterminism
  READS SQL DATA
  CALLED ON NULL INPUT,

```

4. *Rationale: The term rank is not consistent with its function.*

Add the following Definitional Rule:

5.1) *FT\_ScoreDeterminism* is either NOT DETERMINISTIC or DETERMINISTIC.

5. *Rationale: The term rank is not consistent with its function.*

Replace Description 1) d) with:

d) a method *Score(FT\_Pattern)*,

6. *Rationale: The term rank is not consistent with its function.*

Replace Description 1) e) with:

e) a method *Score(CHARACTER VARYING)*,

### 5.1.3 Rank Methods

1. *Rationale: The term rank is not consistent with its function.*

Replace Subclause 5.1.3 with:

#### Purpose

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

#### Definition

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

CREATE METHOD Score
  (pattern CHARACTER VARYING(FT_MaxPatternLength))
  RETURNS DOUBLE PRECISION
  FOR FullText
  RETURN SELF.Score(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 *Score(FT\_Pattern)* takes the following input parameters:
  - a) an *FT\_Pattern* value *pattern*.
- 2) The method *Score(CHARACTER VARYING)* takes the following input parameters:
  - a) a *CHARACTER VARYING* value *pattern*.
- 3) The result of the invocation *Score(CHARACTER VARYING)* or *Score(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 exception – 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 *Score(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

1. *Rationale: Provide missing text regarding tokenization of stop words.*

Add the following Description:

- 3.1) 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.Tokenize()*. 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.

#### 5.1.5 TokenizePosition Method

1. *Rationale: Lowest position is not necessarily 1 (one).*

Replace Description 9) with:

- 9) Let *TLE* be the element of *SELF.TokenizePosition('WORDS')* with the lowest *Position* value. If leading stop words are included in the result of *SELF.TokenizePosition('WORDS')* then the value of *TLE.Position* shall be 1 (one).

#### 5.1.7 TokenizeAndStem Method

1. *Rationale: Provide missing text regarding tokenization of stop words.*

Add the following Description:

- 3.1) 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.TokenizeAndStem()*. 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.

#### 5.1.8 TokenizePositionAndStem Method

1. *Rationale: Lowest position is not necessarily 1 (one).*

Replace Description 7) with:

- 7) Let *TLE* be the element of *SELF.TokenizePositionAndStem()* with the lowest *Position* value. If leading stop words are included in the result of *SELF.TokenizePositionAndStem()* then the value of *TLE.Position* shall be 1 (one), otherwise the value of *TLE.Position* shall be one more than the number of leading stop words.

### 5.1.9 FullText Methods

#### 1. Rationale: Correct constructor method definitions.

Replace the Definition with:

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

CREATE CONSTRUCTOR 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
```

### 5.3.1 FT\_Pattern Type

#### 1. Rationale: Proximity facility cannot be used with phrases.

Replace the syntactic element for <token list1> and <token list2> in Description 3) with:

```
<token list1> ::= <token list>
<token list2> ::= <token list>
```

Replace the first paragraph of 5) f) that begins with "If *P* is a <Proximity expansion> *PFI*" with:

- 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>, or <stemmed word> 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> which does not specify the optional key word STEMMED is adorned by this missing optional key word.

#### 2. Rationale: Soundex facility cannot be used with <phrase>.

Replace the first sentence of 5) i) that begins with "If *P* is a <Soundex expansion>" with:

- i) If *P* is a <Soundex expansion> *SFI*, augment *SFI* such that the occurrence of a <word> which does not specify a <language specification> is adorned by a <language specification> denoting the default language.

### 6.1.1 FT\_Any Type

1. *Rationale: Correct constructor method definitions.*

Replace the <original method specification> that begins with "METHOD FT\_Any" in the Definition with:

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

### 6.1.3 FT\_Any Method

1. *Rationale: Correct constructor method definitions.*

Replace the Definition with:

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

### 6.4.1 FT\_TextLiteral Type

1. *Rationale: The RETURNS clause is not consistent with the usage of the method.*

Replace the <original method specification>s that begins with "METHOD FT\_Tokenize" in the Definition with:

```
METHOD Tokenize()
  RETURNS FT_TextLiteral ARRAY[1]
  LANGUAGE SQL
  DETERMINISTIC
  CONTAINS SQL
  CALLED ON NULL INPUT,
```

2. *Rationale: Correct constructor method definitions.*

Replace the <original method specification>s that begins with "METHOD FT\_TextLiteral" in the Definition with:

```
CONSTRUCTOR 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

CONSTRUCTOR 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,
```

## 6.4.2 Contains Method

### 1. Rationale: Align semantics with FT\_Phrase.

Replace the Definition with:

```
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;
    ELSEIF CARDINALITY(SELF.Tokenize()) = 0 THEN
        --
        -- !! See Description
        --
    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()[1].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
```

Replace first paragraph of Description 2) with:

- 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 the result of the invocation of *SELF.Tokenize()* and if the cardinality of *T* is one then let *TE* be the first element of *T*, normalized in an implementation-defined way and with leading and trailing blanks removed. If *SELF.EscapeSpec* is the null value, let *TT* be *TE*; otherwise, let *TT* be *TE* ESCAPE *SELF.EscapeSpec*.

### 2. Rationale: Align semantics with FT\_Phrase and allow stop words in query.

Replace Description 2) a) i)

- i) If the cardinality of *T* is zero then it is implementation-defined whether
  - 1) to let *R* be false, or
  - 2) an exception condition is raised: SQL/MM Full-Text - effectively empty search expression.

### 6.4.5 Tokenize Method

1. *Rationale: The RETURNS clause is not consistent with the usage of the method.*

Replace Definition with:

```
CREATE METHOD Tokenize()
  RETURNS FT_TextLiteral ARRAY[1]
  FOR FT_TextLiteral
  BEGIN
    --
    -- !! See Description
    --
  END
```

2. *Rationale: Wild card characters are preserved and provide missing text regarding tokenization of stop words.*

Replace Description 2) with:

- 2) *Tokenize()* normalizes *SELF.LitPart* in an implementation-defined way. Any wild card characters in *SELF.LitPart* are preserved in the result of *Tokenize()*. In addition, it is implementation-defined 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 *Tokenize()* in the same way.

### 6.4.7 FT\_TextLiteral Methods

1. *Rationale: Correct constructor method definitions.*

Replace the Definition with:

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

### 6.5.1 FT\_StemmedWord Type

1. *Rationale: The RETURNS clause is not consistent with the usage of the method.*

Replace the <original method specification>s that begins with "METHOD FT\_Tokenize" in the Definition with:

```
METHOD TokenizeAndStem()
  RETURNS FT_TextLiteral ARRAY[1]
  LANGUAGE SQL
  DETERMINISTIC
  CONTAINS SQL
  CALLED ON NULL INPUT,
```

2. *Rationale: Correct constructor method definitions.*

Replace the <original method specification>s that begins with "METHOD FT\_StemmedWord" in the Definition with:

```

CONSTRUCTOR 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

CONSTRUCTOR 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

```

## 6.5.2 Contains Method

1. *Rationale: Align semantics with FT\_StemmedPhrase.*

Replace the Definition with:

```

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;
  ELSEIF CARDINALITY(TEMP (BasI) 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()[1].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

```

Replace first paragraph of Description 2) with:

- 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 the result of the invocation of *SELF.TokenizeAndStem()* and if the cardinality of *T* is one then let *TE* be the first element of *T*, 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 *TE*; otherwise, let *TT* be *TE* ESCAPE *SELF.EscapeSpec*.

Insert new Description 2) a) 0.i):

- 0.i) If the cardinality of *T* is zero then it is implementation-defined whether
  - 1) to let *R* be false, or
  - 2) an exception condition is raised: SQL/MM Full-Text - effectively empty search expression.

#### 6.5.4 TokenizeAndStem Method

1. *Rationale: Align semantics with FT\_StemmedPhrase.*

Replace the Definition with:

```

CREATE METHOD TokenizeAndStem()
RETURNS FT_TextLiteral ARRAY[1]
FOR FT_StemmedWord
BEGIN
--
-- !! See Description
--
END

```

2. *Rationale: Provide missing text regarding tokenization of stop words.*

Replace Description 2) with:

- 2) *TokenizeAndStem()* normalizes and stem-reduces *SELF.LitPart* in an implementation-defined way. In addition, it is implementation-defined 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 *TokenizeAndStem()* in the same way.

#### 6.5.5 FT\_StemmedWord Methods

1. *Rationale: Correct constructor method definitions.*

Replace the Definition with:

```

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

```

```

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

```

### 6.6.1 FT\_Phrase Type

#### 1. Rationale: Correct constructor method definitions.

Replace the <original method specification>s that begins with "METHOD FT\_Phrase" in the Definition with:

```

CONSTRUCTOR 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,

CONSTRUCTOR 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

```

### 6.6.2 Contains Method

#### 1. Rationale: Align semantics with FT\_TextLiteral.

Replace Definition with:

```

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
    --
    -- !! See Description
    --
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

```

## 2. Rationale: Clean-up of stop word treatment.

Delete Description 4).

3. *Rationale: Align semantics with FT\_TextLiteral.*

Replace Description 6) c) with:

- c) If the cardinality of *TPL* is zero then it is implementation-defined whether
  - i) to let *R* be false, or
  - ii) an exception condition is raised: SQL/MM Full-Text – effectively empty search expression.
- c.1) If *TPL* represents optional words only, then let *R* be true.

4. *Rationale: Fix distance between two tokens.*

Delete Description 8).

### 6.6.5 TokenizePosition Method

1. *Rationale: Wild card characters are preserved and treatment of stop words is implementation-defined.*

Replace Description 2) with:

- 2) *TokenizePosition()* normalizes *SELF.PhrasePart* in an implementation-defined way. Any wild card characters in *SELF.PhrasePart* are preserved in the result of *TokenizePosition()* and *SELF.EscapeSpec* is used as the <escape representation character>. In addition, it is implementation-defined 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

1. *Rationale: Correct constructor method definitions.*

Replace the Definition with:

```
CREATE CONSTRUCTOR 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 CONSTRUCTOR METHOD FT_Phrase
  (w1 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)

```

### 6.7.1 FT\_StemmedPhrase Type

#### 1. Rationale: Correct constructor method definitions.

Replace the <original method specification>s that begins with "METHOD FT\_StemmedPhrase" in the Definition with:

```

CONSTRUCTOR METHOD FT_StemmedPhrase
  (w1 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,

CONSTRUCTOR METHOD FT_StemmedPhrase
  (w1 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

```

### 6.7.2 Contains Method

#### 1. Rationale: Align semantics with FT\_StemmedWord.

Replace Definition with:

```

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
    --
    -- !! See Description
    --
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

```

## 2. Rationale: Clean-up of stop word treatment.

Delete Description 4).

3. *Rationale: Align semantics with FT\_StemmedWord.*

Replace Description 6) c) with:

- c) If the cardinality of *TPL* is zero then it is implementation-defined whether
  - i) to let *R* be false, or
  - ii) an exception condition is raised: SQL/MM Full-Text – effectively empty search expression.
- c.1) If *TPL* represents optional words only, then let *R* be true.

4. *Rationale: Fix distance between two tokens.*

Delete Description 8).

#### 6.7.4 TokenizePositionAndStem Method

1. *Rationale: Treatment of stop words is implementation-defined.*

Replace Description 2) with:

- 2) *TokenizePositionAndStem()* normalizes and stem-reduces the sequence of words represented by *SELF.PhrasePart* in an implementation-defined way. In addition, it is implementation-defined 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

1. *Rationale: Correct constructor method definitions.*

Replace the Definition with:

```
CREATE CONSTRUCTOR 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.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[EliminatedDQS(wl[i + 1])];
    SET i = i + 1;
  END WHILE L1;
  RETURN SELF;
END
```

```

CREATE CONSTRUCTOR 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)

```

### 6.8.1 FT\_Proxi Type

#### 1. Rationale: Correct constructor method definitions.

Replace the <original method specification> that begins with "METHOD FT\_Proxi" in the Definition with:

```

CONSTRUCTOR 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

```

### 6.8.4 FT\_Proxi Method

#### 1. Rationale: Correct constructor method definitions.

Replace the Definition with:

```

CREATE CONSTRUCTOR 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)

```

### 6.9.1 FT\_Soundex Type

#### 1. Rationale: Correct constructor method definitions.

Replace the <original method specification> that begins with "METHOD FT\_Soundex" in the Definition with:

```

CONSTRUCTOR METHOD FT_Soundex(snd FT_TextLiteral)
RETURNS FT_Soundex
SELF AS RESULT
LANGUAGE SQL
DETERMINISTIC
CONTAINS SQL
CALLED ON NULL INPUT

```

#### 6.9.4 FT\_Soundex Method

1. *Rationale: Correct constructor method definitions.*

Replace the Definition with:

```
CREATE CONSTRUCTOR METHOD FT_Soundex
  (snd FT_TextLiteral)
  RETURNS FT_Soundex
  FOR FT_Soundex
  RETURN SELF.spoken(snd).NOT_tag(True)
```

2. *Rationale: Soundex facility is implementation-dependent.*

Replace Description 2) with:

- 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-dependent.

#### 6.9.5 GetSoundsSimilar Function

1. *Rationale: Soundex facility is implementation-dependent.*

Replace Description 3) with:

- 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-dependent.

#### 6.10.1 FT\_BroaderTerm Type

1. *Rationale: Correct constructor method definitions.*

Replace the <original method specification> that begins with "METHOD FT\_BroaderTerm" in the Definition with:

```
CONSTRUCTOR 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
```

#### 6.10.4 FT\_BroaderTerm Method

1. *Rationale: Correct constructor method definitions.*

Replace the Definition with:

```
CREATE CONSTRUCTOR 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)
```

### 6.10.5 GetBroaderTerms Function

#### 1. Rationale: Correcting the semantics of expansion functions.

Replace the Definition with:

```

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)
    UNION
    (SELECT ARRAY[c1] AS EXPRarr1
     FROM (VALUES(startingTerm)) AS t1(c1),
          (VALUES(thes_name)) AS t2(c2)
     WHERE c1 IS NOT NULL AND c2 IS NOT NULL)
    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

```

Replace Description 3) with:

- 3) *GetBroaderTerms*(*CHARACTER VARYING*, *FT\_WordOrPhrase*, *INTEGER*) returns an empty array if the following is true:
  - a) Either *startingTerm* or *thes\_name* is the null value.

Replace Description 6) with:

- 6) The term *startingTerm* is included in the result.

#### 6.11.1 FT\_NarrowerTerm Type

1. *Rationale: Correct constructor method definitions.*

Replace the <original method specification> that begins with "METHOD FT\_NarrowerTerm" in the Definition with:

```

CONSTRUCTOR 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

```

#### 6.11.4 FT\_NarrowerTerm Method

1. *Rationale: Correct constructor method definitions.*

Replace the Definition with:

```

CREATE CONSTRUCTOR 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)

```

### 6.11.5 GetNarrowerTerms Function

#### 1. Rationale: Correcting the semantics of expansion functions.

Replace the Definition with:

```
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)
    UNION
    (SELECT ARRAY[c1] AS EXPRarr1
     FROM (VALUES(startingTerm)) AS t1(c1),
          (VALUES(thes_name)) AS t2(c2)
     WHERE c1 IS NOT NULL AND c2 IS NOT NULL)
    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

```

Replace Description 3) with:

- 3) *GetNarrowerTerms*(*CHARACTER VARYING*, *FT\_WordOrPhrase*, *INTEGER*) returns an empty array if the following is true:
  - a) Either *startingTerm* or *thes\_name* is the null value.

Replace Description 6) with:

- 6) The term *startingTerm* is included in the result.

### 6.12.1 FT\_Synonym Type

1. *Rationale: Correct constructor method definitions.*

Replace the <original method specification> that begins with "METHOD FT\_Synonym" in the Definition with:

```

CONSTRUCTOR 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

```

### 6.12.4 FT\_Synonym Method

1. *Rationale: Correct constructor method definitions.*

Replace the Definition with:

```

CREATE CONSTRUCTOR 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)

```

### 6.13.1 FT\_PREFERREDTERM Type

#### 1. Rationale: Correct constructor method definitions.

Replace the <original method specification> that begins with "METHOD PreferredTerm" in the Definition with:

```
CONSTRUCTOR 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
```

### 6.13.4 FT\_PREFERREDTERM Method

#### 1. Rationale: Correct constructor method definitions.

Replace the Definition with:

```
CREATE CONSTRUCTOR 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)
```

### 6.13.5 GetPreferredTerms Function

#### 1. Rationale: Correct the semantics.

Replace the Definition with:

```
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 OR
         CARDINALITY(ret) > 0 THEN
        CAST(ARRAY[] AS FT_WordOrPhrase
             ARRAY[FT_MaxArrayLength])
    ELSE
        ARRAY[startingTerm]
    END;
END

```

Replace Description 5) with:

- 5) The term *startingTerm* is included in the result if no corresponding preferred terms are found in TERM\_SYNONYM.

#### 6.14.1 FT\_RelatedTerm Type

1. *Rationale: Correct constructor method definitions.*

Replace the <original method specification> that begins with "METHOD FT\_RelatedTerm" in the Definition with:

```

CONSTRUCTOR 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

```

#### 6.14.4 FT\_RelatedTerm Method

1. *Rationale: Correct constructor method definitions.*

Replace the Definition with:

```

CREATE CONSTRUCTOR 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)

```

### 6.15.1 FT\_TopTerm Type

1. *Rationale: Correct constructor method definitions.*

Replace the <original method specification> that begins with "METHOD FT\_TopTerm" in the Definition with:

```
CONSTRUCTOR 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
```

### 6.15.4 FT\_TopTerm Method

1. *Rationale: Correct constructor method definitions.*

Replace the Definition with:

```
CREATE CONSTRUCTOR 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)
```

### 6.15.5 GetTopTerms Function

1. *Rationale: Correcting the semantics of expansion functions.*

Replace the Definition with:

```
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 ||
CASE
  WHEN startingTerm IS NULL OR
    thes_name IS NULL OR
    CARDINALITY(ret) > 0 THEN
    CAST(ARRAY[] AS FT_WordOrPhrase
      ARRAY[ FT_MaxArrayLength])
  ELSE
    ARRAY[startingTerm]
END;
END

```

Replace Description 4) with:

- 4) The term *startingTerm* is included in the result if no top terms are found in TERM\_HIERARCHY.

### 6.16.1 FT\_IsAbout Type

#### 1. Rationale: Correct constructor method definitions.

Replace the <original method specification> that begins with "METHOD ST\_IsAbout" in the Definition with:

```

CONSTRUCTOR METHOD FT_IsAbout (phr FullText)
  RETURNS FT_IsAbout
  SELF AS RESULT
  LANGUAGE SQL
  DETERMINISTIC
  CONTAINS SQL
  CALLED ON NULL INPUT

```