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

**Part 2:
Full-Text**

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

Partie 2: Texte complet

Reference number
ISO/IEC 13249-2:2000(E)



PDF disclaimer

This PDF file may contain embedded typefaces. In accordance with Adobe's licensing policy, this file may be printed or viewed but shall not be edited unless the typefaces which are embedded are licensed to and installed on the computer performing the editing. In downloading this file, parties accept therein the responsibility of not infringing Adobe's licensing policy. The ISO Central Secretariat accepts no liability in this area.

Adobe is a trademark of Adobe Systems Incorporated.

Details of the software products used to create this PDF file can be found in the General Info relative to the file; the PDF-creation parameters were optimized for printing. Every care has been taken to ensure that the file is suitable for use by ISO member bodies. In the unlikely event that a problem relating to it is found, please inform the Central Secretariat at the address given below.

© ISO/IEC 2000

All rights reserved. Unless otherwise specified, no part of this publication may be reproduced or utilized in any form or by any means, electronic or mechanical, including photocopying and microfilm, without permission in writing from either ISO at the address below or ISO's member body in the country of the requester.

ISO copyright office
Case postale 56 • CH-1211 Geneva 20
Tel. + 41 22 749 01 11
Fax + 41 22 749 09 47
E-mail copyright@iso.ch
Web www.iso.ch

Printed in Switzerland

Contents	Page
FOREWORD	ix
INTRODUCTION	x
1 SCOPE	1
2 NORMATIVE REFERENCES	3
2.1 International standards	3
2.2 Publicly available standards	3
3 DEFINITIONS, NOTATIONS, AND CONVENTIONS	5
3.1 Definitions	5
3.1.1 Definitions provided in Part 1	5
3.1.2 Definitions provided in Part 2	5
3.1.3 Definitions taken from ISO/IEC 9075	5
3.1.4 Definitions taken from ANSI/NISO Z39.19	6
3.2 Notations	6
3.3 Conventions	6
4 CONCEPTS	7
4.1 Text model	7
4.2 Text identification facilities	7
4.2.1 Single word patterns (patterns of the form <word>)	8
4.2.2 Single phrase patterns (patterns of the form <phrase>)	8
4.2.3 Patterns representing sets of single words	9
4.2.4 Patterns formed by sets of single phrases	11
4.2.5 Patterns specifying context conditions	13
4.2.6 Patterns involving Boolean operators	14
4.2.7 Identification of <i>FullText</i> values which are pertinent to a given text	15
4.3 Text ranking facilities	15
4.4 Language aspects	16
4.4.1 Multilingual texts and patterns	16
4.4.2 Treatment of stop words	17
4.5 Word normalization	17
4.6 Types and routines provided by this part of ISO/IEC 13249	18
4.6.1 Types and routines intended for public use	18

4.6.2 Types and routines for definition	18
4.6.3 Technique for defining the semantics of Category 1 Contains methods	18

5 FULL-TEXT TYPES 21

5.1 FullText Type and Routines	21
5.1.1 FullText Type	21
5.1.2 Contains Methods	24
5.1.3 Rank Methods	26
5.1.4 Tokenize Method	28
5.1.5 TokenizePosition Method	29
5.1.6 Segmentize Method	31
5.1.7 TokenizeAndStem Method	32
5.1.8 TokenizePositionAndStem Method	33
5.1.9 FullText Methods	35
5.1.10 FullText_to_Character Function	36
5.1.11 StrctPattern_to_FT_Pattern Function	37
5.2 FT_TokenPosition Type and Routines	38
5.2.1 FT_TokenPosition Type	38
5.3 FT_Pattern Type and Routines	39
5.3.1 FT_Pattern Type	39
5.3.2 FT_Pattern Key Words	55

6 STRUCTURED SEARCH PATTERN TYPES 57

6.1 FT_Any Type and Routines	58
6.1.1 FT_Any Type	58
6.1.2 Contains Method	59
6.1.3 FT_Any Method	61
6.2 FT_Primary Type and Routines	62
6.2.1 FT_Primary Type	62
6.2.2 Contains Method	63
6.2.3 StrctPattern_to_FT_Pattern Method	64
6.3 FT_WordOrPhrase Type and Routines	65
6.3.1 FT_WordOrPhrase Type	65
6.3.2 Contains Method	66
6.3.3 StrctPattern_to_FT_Pattern Method	67
6.3.4 getWordArray Method	68
6.4 FT_TextLiteral Type and Routines	69
6.4.1 FT_TextLiteral Type	69
6.4.2 Contains Method	71
6.4.3 StrctPattern_to_FT_Pattern Method	73
6.4.4 matches Method	74
6.4.5 Tokenize Method	75
6.4.6 getWordArray Method	76
6.4.7 FT_TextLiteral Methods	77
6.4.8 EliminateDQS Function	78
6.4.9 InsertDQS Function	79

6.5 FT_StemmedWord Type and Routines	80
6.5.1 FT_StemmedWord Type	80
6.5.2 Contains Method	82
6.5.3 StrctPattern_to_FT_Pattern Method	84
6.5.4 TokenizeAndStem Method	85
6.5.5 FT_StemmedWord Methods	86
6.6 FT_Phrase Type and Routines	87
6.6.1 FT_Phrase Type	87
6.6.2 Contains Method	89
6.6.3 StrctPattern_to_FT_Pattern Method	93
6.6.4 getWordArray Method	94
6.6.5 TokenizePosition Method	95
6.6.6 FT_Phrase Methods	96
6.6.7 matches Function	98
6.6.8 prune Function	100
6.7 FT_StemmedPhrase Type and Routines	101
6.7.1 FT_StemmedPhrase Type	101
6.7.2 Contains Method	103
6.7.3 StrctPattern_to_FT_Pattern Method	107
6.7.4 TokenizePositionAndStem Method	109
6.7.5 FT_StemmedPhrase Methods	110
6.8 FT_Proxi Type and Routines	112
6.8.1 FT_Proxi Type	112
6.8.2 Contains Method	113
6.8.3 StrctPattern_to_FT_Pattern Method	116
6.8.4 FT_Proxi Method	117
6.9 FT_Soundex Type and Routines	118
6.9.1 FT_Soundex Type	118
6.9.2 Contains Method	119
6.9.3 StrctPattern_to_FT_Pattern Method	120
6.9.4 FT_Soundex Method	121
6.9.5 GetSoundsSimilar Function	122
6.10 FT_BroaderTerm Type and Routines	123
6.10.1 FT_BroaderTerm Type	123
6.10.2 Contains Method	125
6.10.3 StrctPattern_to_FT_Pattern Method	126
6.10.4 FT_BroaderTerm Method	127
6.10.5 GetBroaderTerms Function	128
6.11 FT_NarrowerTerm Type and Routines	131
6.11.1 FT_NarrowerTerm Type	131
6.11.2 Contains Method	133
6.11.3 StrctPattern_to_FT_Pattern Method	134
6.11.4 FT_NarrowerTerm Method	135
6.11.5 GetNarrowerTerms Function	136
6.12 FT_Synonym Type and Routines	139
6.12.1 FT_Synonym Type	139
6.12.2 Contains Method	141
6.12.3 StrctPattern_to_FT_Pattern Method	142

6.12.4 FT_Synonym Method	143
6.12.5 GetSynonymTerms Function	144
6.13 FT_PREFERREDTERM Type and Routines	146
6.13.1 FT_PREFERREDTERM Type	146
6.13.2 Contains Method	148
6.13.3 StrctPattern_to_FT_Pattern Method	149
6.13.4 FT_PREFERREDTERM Method	150
6.13.5 GetPreferredTerms Function	151
6.14 FT_RelatedTerm Type and Routines	153
6.14.1 FT_RelatedTerm Type	153
6.14.2 Contains Method	154
6.14.3 StrctPattern_to_FT_Pattern Method	155
6.14.4 FT_RelatedTerm Method	156
6.14.5 GetRelatedTerms Function	157
6.15 FT_TopTerm Type and Routines	159
6.15.1 FT_TopTerm Type	159
6.15.2 Contains Method	160
6.15.3 StrctPattern_to_FT_Pattern Method	161
6.15.4 FT_TopTerm Method	162
6.15.5 GetTopTerms Function	163
6.16 FT_IsAbout Type and Routines	165
6.16.1 FT_IsAbout Type	165
6.16.2 Contains Method	166
6.16.3 StrctPattern_to_FT_Pattern Method	167
6.16.4 FT_IsAbout Method	168
6.17 FT_Context Type and Routines	169
6.17.1 FT_Context Type	169
6.17.2 Contains Method	170
6.17.3 StrctPattern_to_FT_Pattern Method	173
6.17.4 FT_Context Method	175
6.18 FT_ParExpr Type and Routines	176
6.18.1 FT_ParExpr Type	176
6.18.2 Contains Method	177
6.18.3 StrctPattern_to_FT_Pattern Method	178
6.18.4 FT_ParExpr Method	179
6.19 FT_Term Type and Routines	180
6.19.1 FT_Term Type	180
6.19.2 Contains Method	181
6.19.3 StrctPattern_to_FT_Pattern Method	182
6.19.4 FT_Term Method	183
6.20 FT_Expr Type and Routines	184
6.20.1 FT_Expr Type	184
6.20.2 Contains Method	185
6.20.3 StrctPattern_to_FT_Pattern Method	186
6.20.4 FT_Expr Method	187
6.21 FT_PhraseList Type and Routines	188

6.21.1 FT_PhraseList Type	188
6.21.2 Contains Method	189
6.21.3 StrctPattern_to_FT_Pattern Method	191
6.21.4 FT_PhraseList Method	192
7 FULLTEXT_TOKEN TYPE	193
7.1 FullText_Token Type	193
8 SQL/MM FULL-TEXT THESAURUS SCHEMA	195
8.1 Introduction	195
8.2 FT_THESAURUS Schema	196
8.3 TERM_DICTIONARY base table	197
8.4 TERM_HIERARCHY base table	198
8.5 TERM_SYNONYM base table	199
8.6 TERM_RELATED base table	200
9 SQL/MM FULL-TEXT INFORMATION SCHEMA	201
9.1 Introduction	201
9.2 FT_FEATURES view	202
9.3 FT_Schemata view	202
10 SQL/MM FULL-TEXT DEFINITION SCHEMA	203
10.1 Introduction	203
10.2 FT_FEATURES base table	204
10.3 FT_SCHEMATA base table	207
11 STATUS CODES	209
12 CONFORMANCE	211
12.1 Requirements for conformance	211
12.2 Claims of conformance	211
ANNEX A	213

ANNEX B	219
INDEX	220

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 SC 32, *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.

(blank page)

2 Normative references

The following normative documents contain provisions which, 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 ISO and IEC 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 13249-1:2000, *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*.

(blank page)

3 Definitions, notations, and conventions

3.1 Definitions

For the purposes 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 in 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. This condition can be rephrased as:

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 *PPR_i* simply contained in <stemmed phrase>. Let *N_i* be 1 (one) if *PPR_i* represents an optional word. Otherwise, let *N_i* be the number of <phrasepart representation>s *WP_{ij}* that share the base reduced form of *PPR_i*. Let *SPP* be such that *SPP* contains *N₁ * ... * N_N* <phrase> patterns.

For a given <phrasepart representation> *i* there are only occurrences of *WP_{ij}* and every *WP_{ij}* 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 <phrase> 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 stop 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)
```

(blank page)

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 true.

B) 0 (zero), if

s_pattern.Contains(SELF)

is false.

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 true.
- 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 CHARACTER 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>

```