Database
Management
System LINTER®

Version 5.9

Full Text Search Functions

Relational Expert Systems

RELEX®

Table of Contents

Introduction 4

Basic Concepts 5

Purpose and Requirements 6

Purpose 6

Requirements 6

Filters 7

Filter Management 8
- Creating an Internal Filter 8
- Creating an External Filter 8
- Delete a Filter 9
- Alter an External Filter 9

Table Filters 10
- Using a Filter on a Table 10
- Modifying a Column Filter 10
- Deleting a Column Filter 10

File Filters 11
- Setting a File Extension Filter 11
- Cancel a File Extension Filter 12

Indexing 13

Create a Full Text Index 13

Alter a Full Text Index 14

Update a Full Text Index 15

Delete a Full Text Index 15

Search 16

Full Text Index Template 17

Retrieve Text Selection 17

The EXTFILE Data Type 19

Embedded Full Text Index Function 20

External File Name (FILENAME) 20

Full text index creation time(INDEXTIME) 20

Filter Number (FILTER) 21

Extern File Data Updating(FILETIME) 21

Extern File Size (FILESIZE) 21
Extern File Name by Default (DEFAULT) 22

Introduction

This document explains the syntax and semantics of the Linter Search Engine (SE) module that is used to search very large blocks of text.

Basic Concepts

The concept of a full text index implies retrieval of text, based on a word or phrase from the complete text; all the text in one or more documents or all of the text fields in a database.

As a rule, every text document has its on internal structure: paragraphs, headings, signature indents, tables etc. With the help of a text editor, this inner structure can be made fairly complex - we can use various fonts, make lists, do formatting etc. Also different text editors store data in different formats; e.g., .doc, .html, .rtf, .txt. Some documents, such as those in html format, apart from using means of visual presentation of information, have their inner structure marked out, e.g., headings, body text, key, words.

Thus, a requirement for a full text search is recognition of a text's structure and the ability to decode various document formats using converters or filters.

Purpose and Requirements

Purpose

Linter's full text index tools are designed for use in projects requiring high performance and full text indexes through large data warehouses, e.g., Web servers. Full text index tools make it possible to simplify the data storage scheme in an application and minimize the number of required tables.

The full text index system provides:

- a wide variety of word search options including search by beginning, by end by sub-string, by whole words, using wild cards etc.);
- search by misspelled words, a fuzzy search. The system recognizes three main types of mistakes: transposition, omission, substitution of a letter.

Requirements

To perform full text indexes, the following requirements must be met:

- Use Linter version 5.8 or later;
- The $$$FILTER and $$$EXTENSION system tables must exist in the database. To create and upload these tables, the files search.sql and default.sql are executed. These files are delivered as part of the Linter distribution kit and can be found in the DICT sub-directory;
- External filters, see below, are required for processing documents with formats other than those handled by the internal filters.

Filters

Fast document retrieval in a full text index system is possible only hen the words in the document are indexed. Before indexing, texts should be adjusted to a certain standard. E.g., the following text fragments must be identical from the point of view of the search query:

DBMS Linter

DBMS Linter

Since documents in various formats (TXT, DOC, RTF, PDF, HTML, and less widely used formats) may be used, it would not be efficient to process all formats using a single full text index system. A logical solution to this problem is to create a system of filters to extract text from data stored in each specific format. The input of the filter is a stream of data. The output is pure ASCI I text. E.g., using an HTML converter, we can extract only the text from an HTML document.

A filter in Linter is a dynamic library extracting the content (as a text stream) and the properties (author, size date created etc.) from a document. The required filters are installed during DBMS customization or by using special SQL operators.

Linter has a set of embedded filters for the most common file formats.

The filters are attachable modules (libraries). Therefore, any database user can create his own filter for a specific file format, and embed it in the DBMS. See Internal Filters below.

The filters, called internal filters, are embedded in the Linter kernel (Table 1).

Table 1 – Linter's SE Internal Filters

Filter Name	Input File Type
ASCTEXT2TEXT	ASCII
ASCXML2TEXT	HTML, XML, ASCII
UNITEXT2TEXT	UNICODE
UNIXML2TEXT	HTML, XML, UNICODE
DOCRTF2TEXT	RTF, PDF, DOC

The information about all filters available (internal, as well as attached by the user) is stored in the $$$FILTER system table that has the following structure:

```
CREATE  TABLE  $$$FILTER
(
$$$ID INTEGER,            /* filter number     */
$$$NAME CHAR(18),         /* filter name  */
$$$KEY INTEGER,           /* external filter control sum */
$$$MODULE CHAR(128),      /* lib. name, for external filters  */
$$$DESC CHAR(256)         /* comment */
);
```

As previously mentioned, this table is created by executing the search.sql file.

Filter Management

Creating an Internal Filter

Function

Attaching an internal filter.

Although Linter has a set of embedded internal filters, not all of them need be used. This set only represents the potential of the system. To make a filter available for use, the information about it must be placed into the $$$FILTER system table.

Specification

```
<internal filter creation>::=
    CREATE INTERNAL FILTER<filter name>
    DESCRIPTION<description text>;
<filter name>::=identifier
<description>::=string literal
```

Semantic Rules

1) <filter name> must match the name of an internal filter; see Table 1.
2) The $$$FILTER table does not allow filter duplication i.e., repeatedly adding the same internal filter is ignored.
3) The <description> string literal is limited to 256 characters. The value of the string goes in the comment field of the $$$FILTER table.

The filter is available for use immediately upon creation.

Examples

```
CREATE INTERNAL FILTER "ASCTEXT2TEXT";
CREATE INTERNAL FILTER "ASCXML2TEXT" DESCRIPTION 'version 1.0';
```

Creating an External Filter

Version 5.8 of Linter does not support this function.

Function

Attach a user-developed filter to Linter.

Specification

```
<external filter creation>::=
    CREATE [EXTERNAL] FILTER<filter name>=<number>
    MODULE <file specification> DESCRIPTION<description text>;
<filter name>::=identifier
<number>::= integer
<file specification>::= string literal
<description>::=string literal
```

Semantic Rules

1) If <filter name> duplicates the name of any existing external or internal filter (see Table 1, it will not be recognized.
2) <number> must be unique among all filter numbers in the $$$FILTER table.
3) <file specification> names the filter file (library).

4) The <description> string literal is limited to 256 characters. The value of the string goes in the comment field of the $$$FILTER table.

The filter is available for use immediately upon creation.

Example

```
CREATE FILTER "Annotation"=17
MODULE 'f:\phrase\filter\annotation.dll'
DESCRIPTION 'for book annotations';
```

Delete a Filter

Function

Delete a previously installed internal or external filter.

Specification

```
<filter deletion>::= DROP FILTER<filter name>
<filter name>::=identifier
```

Semantic Rules

<filter name> must exist in the $$$FILTER table.

Basic Rules

1) The named filter is deleted from the $$$FILTER system table and is no longer available.
2) Information about an internal filter remains in the database.
3) An external filter file is not deleted physically and can be made available again with the CREATE INTERNAL | [EXTERNAL] FILTER command.

Example

```
DROP FILTER "ASCTEXT2TEXT";
```

Alter an External Filter

Version 5.8 of Linter does not support this function.

Function

Replaces the file (library) of a previously installed external filter.

Specification

```
<filter modification>::= ALTER FILTER<filter name> MODULE<file specif ication>
<filter name>::=identifier
<file specification>::=string literal
```

Semantic Rules

<filter name> must exist in the $$$FILTER table.

Basic Rules

1) For the named filter, the old file name is replaced with the new <filter specification> in the $$$FILTER table.
2) The old file is not deleted physically and can be made available again with the CREATE INTERNAL | [EXTERNAL] FILTER command.

Example

```
ALTER FILTER “Annotation”
MODULE ‘f:\phrase\filter\annotation01.dll’;
```

Table Filters

Using a Filter on a Table

Function

Designate a filter for a column in a table.

Specification

The following element has been added to the column properties specification in the CREATE TABLE function of the SQL language used I n Linter:

CREATE TABLE <table name>
(…<column name> <type> DEFAULT FILTER <filter name> …);

Semantic Rules

<filter name> must exist in the $$$FILTER table.

Basic Rules

The installed filter is used if not blocked by other commands concerning the use of filters.

Example

```
CREATE FILTER TABLE TESTBLOB
(
Id   INTEGER,
Name CHAR(18),
Document BLOB DEFAULT FILTER ASCXML2TEXT
);
```

Modifying a Column Filter

Function

Setting a missing filter for a column or replacing a previously set column filter.

Specification

<filter modification>::=
ALTER TABLE <table name> ALTER COLUMN <column name>
SET DEFAULT FILTER <filter name>;

Semantic Rules

<filter name> must exist in the $$$FILTER table.

Basic Rules

The new filter is assigned to the named column. If a filter was previously assigned it is replaced.

Deleting a Column Filter

Function

Cancel a filter assigned to a column.

Specification

```
<filter cancellation>::=
    ALTER TABLE<table name> ALTER COLUMN<column name>
    DROP DEFAULT FILTER;
```

Semantic Rules

<column name> must refer to the column for which the filter is to be set.

File Filters

For EXTFILE (external files, see section "The EXTFILE Data Type") columns, the choice of filter can be performed by Linter automatically based on the file extension. For this purpose the $$$EXTENSION table is used. It has the following structure:

```
CREATE TABLE  $$$EXTENSION
(
$$$EXT CHAR(18),            /*extension, case sensitive    */
$$$FILTER INTEGER           /*filter ID by default    */
);
```

The table is created using the search.sql file.

Setting a File Extension Filter

Function

Set a filter for a specific file extension.

Specification

```
<filter setting>::= SET DEFAULT FILTER<filter name> FOR<extension>;
<extension>::= string literal
```

Semantic Rules

1) <filter name> must refer to one of the filters present in the $$$FILTER table.
2) The <extension> string is the name of the file extension; e.g., .html.

Basic Rules

SET FILTER is used for EXTFILE columns if the use of another filter is not expressly required.

When default.sql is run the following filters are set for the indicated file extensions:

Table 2 – Filters for Specific File Extensions

Filter Name	File Extension
asctext2text	TXT
asctext2text	Txt
Docrtf2text	DOC
Docrtf2text	Doc
Docrtf2text	RTF
Docrtf2text	Rtf
Docrtf2text	PDF

Filter Name	File Extension
Docrtf2text	Pdf
Ascxml2text	XML
Ascxml2text	Xml
Ascxml2text	HTM
Ascxml2text	Htm
Ascxml2text	HTML
Ascxml2text	Html
Ascxml2text	PHTML
Ascxml2text	Phtml
Ascxml2text	SHTML
Ascxml2text	Shtml

In all internal filters, words may be separated by a space or any character having an ASCII code value lower than that of the space.

Cancel a File Extension Filter

Function

Cancel a filter previously set for a specific file extension.

Specification

```
<cancel filter setting>::= CANCEL DEFAULT FILTER FOR <extension>;
<extension>::= string literal
```

Semantic Rules

The indicated extension must exist in the $$$EXTENSION table.

Basic Rules

The previously set default filter is no longer available.

Indexing

All search engines use indices for data retrieval. Thus, all documents must be indexed prior to searching. Systems that do not use indexing (i.e., that perform searches by scanning the entire text) cannot be utilized for real-time data retrieval involving dozens or hundreds of megabytes of text. The following methods can be used for indexing:

Key word indexing: index entries on each word in a phrase excluding stop-words (prepositions, conjunctions and other frequently occurring syntactic words). E.g., in the phrase "War and Peace", two words ("war" and "peace") will be included in the index. Word order is irrelevant in the process of searching, as every word is retrieved separately and then the search results intersect.

Full text indexing: the content of a field or sub-field is entered into the index as a single whole. Thus, the phrase "war and peace" will constitute one index entry. In the process of retrieval, words must be entered in the correct order.

Key indexing: each word is truncated to equal a specific number of letters.

Permutation indexing: the word order in a phrase is changed, so that any word in the phrase can stand at the beginning.

Linter uses a combination of the first two indexing methods.

Create a Full Text Index

Function

Create a full text index.

Specification

```
<create full text index>::=
        CREATE PHRASE [IMMEDIATE | DEFERRED] [<flag> ...]
        INDEX <column name> ON <table name>;
<flag>::= XML | UNICODE
```

Semantic Rules

1) The flag list is <space>, not comma, separated.
2) Possible column data types and their default values are:

Table 3 – Default Flag Values for Various Data Types Basic Rules

Column Type	XML Flag	UNICODE Flag
CHAR	cleared	permanently cleared (cannot be set)
VARCHAR	cleared	permanently cleared (cannot be set)
NCHAR	cleared	invariably set (cannot be reset)
NVARCHAR	cleared	invariably set (cannot be reset)
BLOB	cleared	cleared
EXTFILE	cleared	cleared

Basic Rules

1) IMMEDIATE denotes instant index updating in case a field is rebuilt. DEFERRED denotes index updating only on the command REBUILD PHRASE INDEX.

2) IMMEDIATE is the default modifier for CHAR, VARCHAR, NCHAR and NVARCHAR columns.
3) For EXTFILE columns, DEFERRED is the default modifier.
4) For BLOB columns, only the DEFERRED modifier is permitted.
5) When creating an index the following filter selection rules will apply:
   a. For BLOB columns:
      - the filter whose number in the $$$FI LTER table equals the BLOB data type number (i.e. the filter specially set for this BLOB value) is selected;
      - if such a filter does not exist, the default filter set for this column is selected;
      - if there is no default filter, the filter is determined by the flags selected;
      - if the flag has not been set, the asctext2text filter is used.
   b. For EXTFILE columns:
      - the filter set for the column in the following construction is selected:

```
INSERT INTO table name ( … column name … )
VALUES (… EXTFILE(file name [,filter name] ) | NULL … );
```

or

```
UPDATE INTO table name SET column name= VALUES (… EXTFILE(file
name [,filter name] ) | NULL … );
```

      - if this filter does not exist, the default filter set for this column is selected;
      - if there is no such filter, then the default filter set for the file extension is selected;
      - if this filter has not been set, the filter is determined by the flags;
      - if the flag has not been set, the asctext2text filter is used.
   c. For CHAR and VARCHAR columns:
      - the default filter set for the column;
      - if there is no default filter, the filter is determined by the flag;
      - if the flag has not been set, the asctext2text filter is used.
   d. For NCHAR and NVARCHAR columns:
      - the default filter set for the column;
      - the default filter determined by the flag (i.e. unitext2text).

Alter a Full Text Index

Specification

```
<alter full text index>::=
        ALTER PHRASE INDEX <column name> ON <table name>
        [ IMMEDIATE | DEFERRED ];
```

Basic Rules

1) IMMEDIATE denotes instant index updating in case a field is rebuilt. DEFERRED denotes index updating only at the command REBUILD PHRASE INDEX.

2) IMMEDIATE is the default modifier for CHAR, VARCHAR, NCHAR and NVARCHAR columns.
3) DEFERRED is the default modifier for EXTFILE columns.
4) For BLOB columns, DEFERRED is the only modifier permitted.
5) If DEFERRED has been the previous working mode, switching to the IMMEDIATE mode will cause immediate index updating.

Update a Full Text Index

Function

Update full text indices for BLOB and EXTFILE columns.

Specification

```
<updater full text index >::=
        REBUILD PHRASE INDEX <column name> ON <table name>;
```

Semantic Rules

<column name> must be of the BLOB or EXTFILE type.

Basic Rules

All types of indices (simple, compound, phrase) for CHAR, VARCHAR NCHAR and NVARCHAR columns are updated by Linter automatically on adding, deleting or altering a value. Conversely, indices for BLOB and EXTFILE columns can be updated only by execution of this function (if IMMEDIATE full text index updating mode is not indicated - see sections “Create a Full Text Index” and “Alter a Full Text Index”). This rule is implemented because index updating is a time-consuming process that should be performed when the DBMS kernel workload is minimal.

Delete a Full Text Index

Function

Delete an existing column full text index.

Specification

```
<delete full text index>::=
        DROP PHRASE INDEX <column name> ON <table name>
```

Semantic Rules

<column name> must belong to a column having a full text index.

Search

Function

In most cases, it is necessary to retrieve text containing not only several predetermined words and phrases, but also words similar to each other, morphologically related or having the same beginning or ending.

For this purpose, a full text index predicate has been included in Linter's SQL:

Specification

```
<full text index predicate>::=
        <column name> [ NOT ] CONTAINS [ <modifier> … ]
        <full text index template>;
<modifier>::=
        SENSITIVE | PARTIALLY | AT_BEGIN | AT_END | FUZZY
<full text index template>::= string literal
```

Semantic Rules

1) <column name> must belong to the column on which the full text index is based.
2) The modifiers, absent by default, are enumerated in any order, separated by spaces, and applied to all words in a template.

Basic Rules

The modifiers have the following characteristics:

Table 4 – full text index Modifiers

Modifier	Function	Symbol	Use in Template	Example
SENSITIVE	sets a case sensitive search; by default the search is case-insensitive	#	before the word	#Relex
PARTIALLY	ets a search of documents in which the search template may occur in any place	*	before and after the word	*RELEX*
AT_BEGIN	sets the search of documents in which the search template occurs only at the beginning of words	*	at the beginning	RELEX*
AT_END	sets a search of documents in which the search template occurs only at the end of words	*	at the end	*RELEX
FUZZY	sets fuzzy search	%	before the word	%RELEX

When setting a modifier, all the words are accompanied by the corresponding symbol (see Symbol in Table 4) when the search function is called.

If the modifier is to be applied to only some of the words in a text, it should not be set. Instead, a special symbol should be used for these specific words in the search template.

Full Text Index Template

Function

Create a full text index template.

Specification

```
<full text index template>::= <search expression>
<search expression>::=
       <word>
       | <not equal> <search expression>
       | <search expression> or <search expression>
       | <search expression> [ and ] <search expression>
       | (<search expression> )
       | <word> =<word>
<not equal>::= !
<or>::= |
<and>::= &
```

Semantic Rules

1) *: a single asterisk displays all documents represented in the full text index or search expression.
2) < `word` >: a string literal can contain only the following characters:
    - A-Z, a-z,o-9,and_
    - In the middle of words only: @,-,/,and\
    - The special characters: #, 96, and * (see Table 4).

Basic Rules

1) ! `<search expression>` construction initiates a search for documents not containing the given search expression.
2) `<search expression> | <search expression>` searches for documents containing either or both expressions.
3) `<search expression>` [&] `<search expression>` searches for documents containing both expressions. The & sign may not be an indicator.
4) (`<search expression>`) controls the logical order of analyzing the search expression.
5) `<word>=<word>` sets attributes for searching XML and HTML documents. The first `<word>` sets the attribute name (e.g., date user); the second <word> determines the search value of the attribute.

Retrieve Text Selection

Function

Get text from a document using the embedded SQL function GETTEXT.

Specification

```
GETTEXT(<column name>,<offset>,<length>)
```

Semantic Rules

1) <column name> must belong to a column whose data type allows creation of a full text index. The index itself does not necessarily exist.
2) <offset> is an integer determining the initial position of the required portion of the text. The count starts from l.
3) <length> is the size of the required text portion measured in symbols. It must be a positive integer in the range of l through 4000).

Returned Value

The returned value is the required text portion processed by the phrase text filter and defined by the indicated indent and length. If the text for the given value is blank, or does not fit into the required indent range the results buffer is filled with spaces. The same happens to the remainder of the response if its length is excessive.

As the system returns the text processed by a filter, the punctuation marks may be missing.

The EXTFILE Data Type

This section describes the syntax and internal structure of the EXTFILE data type implemented by Linter. The EXTFILE column specifies the PATH to a text file that is outside of, external to, the database.

To create an EXTFILE column:

```
CREATE TABLE<table name>
    ( ... <column name> EXTFILE [ ROOT'<root directory>' ] ... );
```

The internal storage structure (from which the last field may be deleted) is:

```
#define LMAXPATHSIZE 512
typedef struct
{
LONGINT filterid;
LDATETIME index_time;
char   file_name[L_MAXPATHSIZE];
} FILEDESC;
```

For storage, the file names in DOS/WIN32 systems are transformed by replacing forward slashes, /, with back slashes, /.

If the initial directory in the ROOT clause has been set, all files with relative names (i.e. having no initial'/' symbol in UNIX and no initial device name in DOS/WIN32) are retrieved in relation to this directory. Otherwise they are retrieved in relation to the database directory.

The file name set in the ROOT clause for the EXTFILE type is stored as a default value containing a text line.

When using the EXTFILE function, the following SQL clauses are used in Linter:

Set Filter

```
EXTFILE ('<file name>' [ ,<filter name ] );
```

Insert Data

```
NSERT INTO <table name> ( ... <column name> ... )
    VALUES ( ... EXTFILE ( '<file name>' [ , <filter name> ] ) | NULL ... );
```

Alter Data

```
UPDATE INTO <table name>
  SET <column name> = EXTFILE ( '<file name>' [ , <filter name> ] )
  | NULL ... ;
```

Set New Root Directory for an EXTFILE Column

```
ALTER TABLE <table name>
  ALTER COLUMN <column name>  SET ROOT '<directory name>' ;
```

Cancel Root Directory for an EXTFILE Column

```
ALTER TABLE <table name>
  ALTER COLUMN <column name>  DROP ROOT;
```

Embedded Full Text Index Function

External File Name (FILENAME)

Function

Gen the name of the file complaint with the EXTFILE data type.

Specification

`FILENAME(column name);`

Basic Rules

`<column name>` should belong to a column with the EXTFILE data type.

Return Value

1) Character line with a fixed length of 512 bytes;
2) Char(512) containing a file name. The directory name is not included in the file name if set as ROOT. If the directory name is expressly indicated in the record it is included in the return value;
3) The separator (colon) in the directory specification is replaced with the standard symbol' |';
4) If the argument is NULL the result is NULL.

Example

```
DROP TABLE EXT;
CREATE TABLE EXT(ID INT, EXT1 EXTFILE);
INSERT INTO EXT VALUES(1, EXTFILE(‘C:\AUTOEXEC.BAT));
INSERT INTO EXT VALUES (2, EXTFILE(‘C:\CONFIG.SYS’));
INSERT INTO EXT VALUES (3, EXTFILE(‘D:\TEST1TXT));
INSERT INTO EXT VALUES (4, EXTFILE(‘C:\TESTTEST2TXT));
UPDATE EXT
SET EXT1 = EXTFILE(‘C:\AUTOEXEC.BAT,ASCTEXT2TEXT)
WHERE ID =>2;
SELECT ID, CAST FILENAME(EXT1) AS CHAR 20 FROM EXT;
|1 |C|/AUTOEXEC.BAT
|2|C|/AUTOEXEC.BAT
|3|D|/TEST1TXT
|4|D|/TESTTEST2TXT
```

Full text index creation time(INDEXTIME)

Function

Get the date and time of the last updating of the full text index.

Specification

`INDEXTIME(<column name>);`

Basic rules

`<column name>` should be of CHAR, VARCHAR, NCHAR, NVARCHAR, BLOB OR EXTFILE data type.

Returned Value

1) Value of DATE data type contained the date and time (Greenwich Time) of the last updating of the full text index of the <column name> column;
2) NULL value in case of:

- Full text index for <column name> is not created;
- Value of the <column name> is NULL.

Filter Number (FILTER)

Function

Get the filter number.

Specification

```
FILTER(<column name>);
```

Basic Rules

<column name> should be of CHAR, VARCHAR, NCHAR, NVARCHAR, BLOB OR EXTFILE data type.

Returned Value

1) Integer value – filter number for <column name> column.
2) 0 - in case of:
   - if filter is not specified;
   - if filter is identified automatically by external file name extension;
   - if default filter is used.
3) NULL value if value of column is NULL.

Extern File Data Updating(FILETIME)

Function

Get the date and time of the last updating of the external file.

Specification

```
FILETIME(<column name>);
```

Basic Rules

<column name> should be of EXTFILE data type.

Return Value

1) DATE data type value contained date and time (Greenwich Time) of the last updating of the external file;
2) NULL – if value of the column is NULL.

Extern File Size (FILESIZE)

Function

Get the size of the external file.

Specification

```
FILESIZE(<column name>);
```

Basic Rules

`<column name>` should be of EXTFILE data type.

Returned Value

1) BIGINT value which contains size of the file in bytes;
2) NULL value if:
   - External file doesn't exist;
   - Value of the column is NULL.

Extern File Name by Default (DEFAULT)

Function

Get external file default name.

Specification

```
DEFAULT(<column name>);
```

Basic Rules

`<column name>` should be of EXTFILE data type.

Returned Value

The return value is the file specification.