Red Database
Version 2.5
Full text search

This document contents description of using full text search in DBMS Red Database 2.5. Document is intended for users knowing with DBMS Red Database, SQL, PSQL and Java.

www.red-soft.biz

REDSOFT

Table of contents

REDSOFT

Glossary

Daemon - the process working in a background mode without direct interaction with the user.

Index - the object created by Lucene system based on analysis of the document contents. Index is necessary for the search realization.

Indexation – update index.

Metadata - "data about data". Set of system objects (tables, triggers etc.).

Morphological search - search with the morphology control (all possible forms of a word).

Fuzzy Searches - allows to find the demanded information if spelling errors in the document or in query are present.

Full text search (FTS) - search document in a database of texts on the basis of of these documents contents, and also set of methods of optimization of this process.

Collation - The logical ordering of character or wide-character strings according to defined precedence rules. These rules identify a collation sequence between the collating elements, and such additional rules that can be used to order strings consisting of multiple collating elements.

Search term - the term which is search on full coincidence. Lucene supports simple and phrases. Simple terms consist of one word, phrases consist of several terms surrounded by double quotes.

Search phrase - the initial information for search by using search system. The search query is consists of a set of terms and operators.

The MIME-format - a document format proceeding from which rules of extraction of data from the document are defined at indexation.

Lucene - freely extended library for high-speed full text search.

RDB$DB_KEY - it is a "record number". It can be used as the unique identifier of record, as well as primary key. However rdb$db_key can vary during the work. Physically it represents table number, number of page and displacement on record (and not on the concrete version, and in general on a package of versions of this record if they are).

Introduction

«Red Database» 2.5 provides full text search in a databases. Full text search is based on is free-extended library Lucene. This library contains indexing and search functions, access to these functions is realized through API Lucene.

General features of FTS is:

• fuzzy search;

• morphological search;

• search on several objects in a database (search in several columns and tables).

Search can be carried out on text fields (char, varchar), and also on binary and text BLOB-fields. Thus binary type BLOB-fields can contain documents of following types:

- *.pdf (Acrobat);
- *.doc (MS Word);
- *.xls (MS Excel);
- *.ppt (MS PowerPoint);

- *.rtf;
- *.htm, *.html;
- *.odt (Open Office Writer).

«Red Database» 2.5 full text search system using Lucene written in Java, therefore for use this feature it is necessary to establish JDK version 1.6 or higher. Adjustment of parameters of interaction of a «Red Database» server and Java virtual machine is carried out by configuration file jvm.conf which is located in the root directory of a Red Database server installation.

In this file symbol «#» is used for a designation of comments. The text following a symbol «#», till the end of a line is the comment. It is necessary to adjust following parameters in a file:

● full path to the virtual machine;

● path to java-libraries.

The full path to the virtual machine is specified in the first uncommented line of file. Path to virtual machine also may be specified as a value of environment variable:

```
C:\Java1_6\jre\bin\client\jvm.dll #full path to jvm
<jvm_from_JAVA_HOME> #environment variable name
```

The path to Java-libraries with all functions of full text search is set by means of parameter -cp (abbr. from class path). By default a server load *.jar files located only in a directory java_lib, relative a root directory of a server. This parameter allows to specify additional *.jar files which will be load additional to *.jar files from a java_lib directory. Names of jar-files are listed in one line, as a divider for Windows systems is used the symbol «;», for Unix systems and a symbol is used the symbol «:» :

```
-cp java_lib/commons-beanutils-
1.6.1.jar;java_lib/commons-collections-2.1.jar
```

For using full text search feature presence of following Java-libraries is necessary:

- fb-lu.jar

- jaybird-esp-2.1.6.jar

- jaybird-full-2.1.6.jar

- lius.jar

- lucene-highlighter-2.1.0.jar

- lucene-analyzers-2.1.0.jar

- lucene-highlighter-2.1.0.jar

- MimeType.jar.

These libraries should be placed in subdirectory java-lib or paths to them should be specified in parameter -cp in a configuration file jvm.conf.

1 Special tables of full text search

For using full text search feature a database should be all necessary special objects. For automatic creation of these objects it is necessary to specify path to an initialization script as a value of parameter InitScript in a configuration file of a server. Then this script will be carried out at creation of each new database. By default the name of a initialization script is init.sql, it is located in the root directory of a server. For example:

```
InitScript = init.sql
```

There are four special tables used in full text search:

Table 1.1 - Full text search special tables

Table	Description
FTS$INDICES	Index meta data
FTS$INDEX_SEGMENTS	Meta data of fields entering into an index
FTS$POOL	RDB$DB_KEY value for changed and still not reindexed fields
FTS$LUCENE_FILE_SYSTEM	Emulation of file system for an index data storage

Metadata of indexes are stored in special table FTS$INDICES. The description of FTS$INDICES special table is resulted below in table 1.2:

Table 1.2 - The special table FTS$INDICES structure

Field name	Field type	Description
FTS$INDEX_NAME	CHAR(31) CHARACTER SET UNICODE_FSS	Index name
FTS$STORE	BLOB SUB_TYPE 1 SEGMENT SIZE 80 CHARACTER SET UNICODE_FSS	The description where the index is stored. Can be NULL, in this case the index will be stored in table FTS$LUCENE_FILE_SYSTEM. Value by default NULL. If a preset value "file" indexes will be stored in file system
FTS$DESCRIPTION	BLOB SUB_TYPE 1 SEGMENT SIZE 80 CHARACTER SET UNICODE_FSS	Comment on index
FTS$INDEX_STATUS	CHAR(1)	The index status. This field can accept following values: ' I ' inactive - an index is inactive; ' N ' new - the index is created, full reindexing is required; ' U ' needs meta data update - change of meta data, triggers etc. is required; ' D ' drop - the index is noted to removal; ' C ' complete - all changes for an index in meta data are applied and it is indexed

Data about structure (segments) of indexes – meta data of the fields entering into an index are stored in the table $INDEX_SEGMENTS. The description of FTS$INDEX_SEGMENT special table is resulted below in table 1.3:

Table 1.3 - The special table FTS$INDEX_SEGMENTS structure

Field name	Field type	Description
FTS$INDEX_NAME	CHAR(31) CHARACTER SET UNICODE_FSS	Index name
FTS$RELATION_NAME	CHAR(31) CHARACTER SET UNICODE_FSS	Indexed table

Field name	Field type	Description
FTS$FIELD_NAME	CHAR(31) CHARACTER SET UNICODE_FSS	Indexed field
FTS$TRIGGER_NAME	CHAR(31) CHARACTER SET UNICODE_FSS	Name of the trigger which will insert data into table FTS$POOL, after change of data (see below)
FTS$ANALIZER	CHAR(255) CHARACTER SET UNICODE_FSS	Analyzer name
FTS$MIME_TYPE	CHAR(127) CHARACTER SET UNICODE_FSS	Name of a MIME format of the document stored in the indexed field. It is used if documents of one type are stored in the indexed field[1]
FTS$MIME_FIELD_NAME	CHAR(31) CHARACTER SET UNICODE_FSS	Field name in which MIME format of a document is stored. It is used if documents of different type are stored in the indexed field

Table FTS$POOL contains values RDB$DB_KEY for changed, but still not indexed records. The structure of FTS$POOL table is resulted below in table 1.4.

Table 1.4 - The special table FTS$POOL structure

Field name	Field type	Description
FTS$DB_KEY	CHAR(8) CHARACTER SET OCTETS	RDB$DB_KEY of record which has been added, changed or removed

Table FTS$LUCENE_FILE_SYSTEM is necessary for an index data storage. Index data loads from an external files into the FTS$LUCENE_FILE_SYSTEM table into FTS$FILE_BODY field. The structure of FTS$LUCENE_FILE_SYSTEM table is resulted below in table 1.5.

Table 1.5 - The special table FTS$ LUCENE_FILE_SYSTEM structure

Field name	Field type	Description
FTS$INDEX_NAME	CHAR(31) CHARACTER SET UNICODE_FSS	Index name
FTS$FILE_NAME	VARCHAR(255) CHARACTER SET UNICODE_FSS	External file name
FTS$LAST_MODIFY_TIME	TIMESTAMP	Time of last change of a file
FTS$FILE_BODY	BLOB SUB_TYPE 0 SEGMENT SIZE 1024	File contents

---

1 Avaliable values of field FTS$MIME_TYPE are resulted in the table 2.3.

REDSOFT

2 Stored procedures of full text search

Stored procedures using in full text search are listed below:

- FTS$CREATE_INDEX;

- FTS$ADD_FIELD_TO_INDEX;

- FTS$APPLY_METADATA_CHANGES;

- FTS$REINDEX;

- FTS$FULL_REINDEX;

- FTS$STARTDAEMON;

- FTS$SEARCH;

- FTS$DROP_FIELD_FROM_INDEX;

- FTS$DROP_INDEX.

The FTS$CREATE_INDEX procedure is used for index creating:

Table 2.1 - Input parameters of procedure FTS$CREATE_INDEX

Parameter name	Parameter type	Description
FTS$INDEX_NAME	CHAR(31) CHARACTER SET UNICODE_FSS	Index name
FTS$STORE	BLOB SUB_TYPE 1 SEGMENT SIZE 80 CHARACTER SET UNICODE_FSS	The description where the index is stored. Can be NULL then the index will be stored in table FTS $ LUCENE_FILE_SYSTEM. Value by default is NULL. If a value is "file" indexes will be stored in the file system
FTS$DESCRIPTION	BLOB SUB_TYPE 1 SEGMENT SIZE 80 CHARACTER SET UNICODE_FSS	Comments on index. Default value is NULL.

Procedure FTS$ADD_FILED_TO_INDEX adds an indexed field into an index. Input parameters of procedure FTS$ADD_FILED_TO_INDEX are resulted below in the table 2.2.

Table 2.2 - Input parameters of procedure FTS$ADD_FILED_TO_INDEX

Parameter name	Parameter type	Description
FTS$INDEX_NAME	CHAR(31) CHARACTER SET UNICODE_FSS	Index name
FTS$RELATION_NAME	CHAR(31) CHARACTER SET UNICODE_FSS	Indexed table
FTS$FIELD_NAME	CHAR(31) CHARACTER SET UNICODE_FSS	Индексируемое поле.
FTS$ANALIZER	CHAR(255) CHARACTER SET UNICODE_FSS	Analyzer name. Can accept value NULL value, in this case analyzer Standard (English) analyzer will be using. Default value is NULL.[2]
FTS$MIME_TYPE	CHAR(127) CHARACTER SET UNICODE_FSS	The MIME type specifies, for what type of document will creates index. Can accept value NULL. Accessible MIME-types of documents are resulted in

---

2 Available values of names of analyzers are resulted in the table 2.4.

Parameter name	Parameter type	Description
		table 2.3
FTS$MIME_FIELD_NAME	CHAR(31) CHARACTER SET UNICODE_FSS	Name of a field which will define MIME format of documents. Can accept value NULL.

Possible combinations of input parameters are listed below:

- If value of parameters FTS$MIME_TYPE and FTS$MIME_FIELD_NAME is NULL the index will creates not for documents.

- If value of parameter FTS$MIME_TYPE is not NULL, and value of parameter FTS$MIME_FIELD_NAME is NULL the index will creates for documents. In this case, value of parameter FTS$MIME_TYPE, defines one MIME format for all documents stored in the added field.

- If value of parameter FTS$MIME_FIELD_NAME is not NULL, and parameter FTS$MIME_TYPE is NULL the index will creates for documents. In this case MIME format of the document is set for each record by value stored in the additional field which name is specified as value of parameter FTS$MIME_FIELD_NAME.

Table 2.3 - Accessible MIME-types of documents

Document type	Mime type
PDF	application/pdf
Microsoft Excel	application/vnd.ms-excel
Microsoft Word	application/msword
Microsoft PowerPoint	application/vnd.ms-powerpoint
RTF	application/rtf
Open Office Writer (ODT)	application/vnd.oasis.opendocument.text
HTML	text/html

Value of input parametre FTS$ANALIZER defines, what type of the analyzer will be used at indexation of an added field:

Table 2.4 - Correspondence names of analyzers and languages

Analyzer name	Language
English	English
Standard	English
Russian	Russian
German	German
French	French
Czech	Czech
Brazilian	Brazilian
Chinese	Chinese
Dutch	Dutch
Greek	Greek
CJK	Chinese, Japanese, and Korean

Attention! The correctness of results of search directly depends on type of the chosen analyzer.

For applying changes of meta data it is necessary to execute FTS$APPLAY_METADATA_CHANGES procedure. Input and output parameters at procedure are absent. It is necessary to execute given procedure after change of structure of an index (addition or removal of fields in an index).

Special procedure FTS$REINDEX makes full (on all records, with replacement before an existing index) reindexation of the specified index. Procedure has one entrance parameter - an index name (the Table. 2.5).

Table 2.5 - Input parameter of FTS$REINDEX procedure

Paraseter name	Parameter type	Description

FTS$INDEX_NAME	CHAR(31) CHARACTER SET UNICODE_FSS	Index name

To execute full reindexation for all indexes in a DB, it is necessary to execute procedure FTS$FULL_REINDEX procedure. This procedure has no input and output parameters.

That at change of indexed data sets reindexation executes automatically possible to start reindexation daemon. To start daemon it is necessary to execute procedure FTS$STARTDAEMON. Daemon carries out continuous (everyone 0,1) monitoring of table FTS$POOL and reindexes changed records then records leave from FTS$POOL. Procedure FTS$STARTDAEMON has neither input no output parameters.

For extraction of data from an index procedure FTS$SEARCH is used. Input parameters of procedure FTS$SEARCH are resulted in table 2.6. Required input parameters are an index name on which search will be carried out, and a search query.

Table 2.6 - Input parameters of FTS$SEARCH procedure

Parameter type	Parameter type	Description
FTS$INDEX_NAME	CHAR(31) CHARACTER SET UNICODE_FSS	Index name
FTS$RELATION_NAME	CHAR(31) CHARACTER SET UNICODE_FSS	Name of the indexed table. Can accept value NULL then search will go under all tables entering into an index
FTS$FILTER	VARCHAR(4000) CHARACTER SET UNICODE_FSS	The filter, on which search will be carried out (See http://lucene.apache.org/java/2_3_1/queryparsersyntax.html).
FRAGMENT_SIZE	INTEGER	The parameter setting displayed quantity of symbols of a fragment of the search result. The value 50 is used by default.

Procedure FTS$SEARCH has following target parameters: value RDB$DB_KEY for the found records, value of relevance of search result, a table name in which data was found. The description of output parameters is resulted in the 2.7.

Table 2.7 - Output parameters of FTS$SEARCH procedure

Parameter name	Parameter type	Description
RELATION	VARCHAR(512)	Table, witch founded data is stores
ROW_ID	CHAR(8) CHARACTER SET OCTETS	RDB$DB_KEY value for search result
SCOPE	DOUBLE PRESITION	Relevance of search result
HIGHLIGHT	VARCHAR(512)	The fragment of the search result. The found line consists in <B> </B> tags

Procedure FTS$DROP_FILED_FROM_INDEX is used for removal of an indexed field from an index. Input parameters are resulted in the table 2.8

Table 2.8 - Input parameters of FTS$DROP_FILED_FROM_INDEX procedure

Parameter name	Parameter type	Description
FTS$INDEX_NAME	CHAR(31) CHARACTER SET UNICODE_FSS	Index name
FTS$RELATION_NAME	CHAR(31) CHARACTER SET UNICODE_FSS	Indexed table
FTS$FIELD_NAME	CHAR(31) CHARACTER SET UNICODE_FSS	Indexed field

For removal of an index from full text search system procedure FTS$DROP_INDEX is used. The description of input parameter of procedure is resulted in the table 2.9.

Table 2.9 - Input parameters of FTS$DROP_FILED_FROM_INDEX procedure

Parameter name	Parameter type	Description
FTS$INDEX_NAME	CHARCHARACTER SET UNICODE_FSS	Index name

3 Example of full text search use

Generally it is possible to present rules of use of system of text-through search as follows:

1. Index creation;

2. Addition/removal of fields in an index;

3. Full text search system meta data update;

4. Reindexation;

5. Search;

6. Removal of index (if it is necessary).

3.1 Index creation

At first it is necessary to create an index. For this purpose it is necessary to execute FTS$CREATE_INDEX procedure. Obligatory input parameter is only the index name. The second input parameter is used to define where the index should be stored. Can be NULL then the index will be stored in table FTS $ LUCENE_FILE_SYSTEM. Value by default is NULL. If a value is "file" indexes will be stored in the file system.
Example of creation of an index:

```
EXECUTE PROCEDURE FTS$CREATE_INDEX ('SIMPLE_INDEX',
'file', 'Simple index for FTS')
```

At create index record in table FTS$INDICES will be added. Value of FTS$INDEX_STATUS field is equal ' N ', that means, that the index is just created, demands full reindexation.

3.2 Removal of an index

For index removal FTS$DROP_INDEX procedure is used. It has one obligatory input parameter - an index name .
Example of removal of an index:

```
EXECUTE PROCEDURE FTS$DROP_INDEX ('SIMPLE_INDEX');
```

At index removal also leave:

- The record from FTS$INDICES table related with an index;

- Records form FTS$INDEX_SEGMENTS table related with an index;

- Records from FTS$LUCENE_FILE_SYSTEM table related with an index;

- Temporary files relate with an index.

3.3 Addition fields to an index

After the index is created, it is possible to add fields from database tables in it. For addition field in an index FTS$ADD_FIELD_TO_INDEX procedure is used. It has six input parameters. Procedure FTS$ADD_FIELD_TO_INDEX has four obligatory input parameters: index name, table name, field name, analyzer name. To define analyz-

er name parameter FTS$ANALIZER is used. It can accept value NULL value, in this case analyzer Standard (English) analyzer will be using. Default value is NULL. Accessible types of analyzer are resulted in table 2.4.

Last two parameters of procedure FTS$ADD_FIELD_TO_INDEX procedure also are used to specify, for what type of document will creates index. To define one MIME format for all documents stored in the added field parameter FTS$MIME_TYPE is used. If it is necessary to define MIME format of the document for each record parameter FTS$MIME_FIELD_NAME. Accessible MIME-types of documents are resulted in table 2.3

Example of addition of a field in an index (the index is created not for documents):

```
EXECUTE PROCEDURE FTS$ADD_FIELD_TO_INDEX
('SIMPLE_INDEX', 'TABLE1', 'FIELD1',NULL ,NULL, NULL)
```

Example of addition of a field in an index (the index is created for documents of one type)::

```
EXECUTE PROCEDURE FTS$ADD_FIELD_TO_INDEX
('SIMPLE_INDEX', 'TABLE1', 'FIELD1', 'English',
'application/pdf', NULL)
```

After addition field in an index in table FTS$INDEX_SEGMENTS there should be a corresponding record.

3.4 MIME type of a document

If the index is created for documents it is required to specify MIME type of a document at addition of a field in an index.

«Red Database» full text search system it is possible to use one of seven MIME-formats. Correspondence of MIME-types of documents to the values can specified in the field FTS$MIME_TYPE, is resulted in table 2.3.

3.5 Removal fields from an index

To remove fields from an index in FTS$DROP_FIELD_FROM_INDEX procedure is used. Procedure has three obligatory input parameters: an index name from which the field removes; a name of the table which contains this field; a name of a deleted field. The example of removal of a field from an index is resulted below:

```
EXECUTE PROCEDURE FTS$DROP_FIELD_FROM_INDEX
('SIMPLE_INDEX', 'TABLE1', 'FIELD1')
```

After the field was removed from index relation record from FTS$INDEX_SEGMENTS table should be removed.

3.6 Full text search system meta data update

After change of index structure, in particular after removal of a field from it , it is necessary to update the meta data of full text search system. Procedure FTS$APPLY_METADATA_CHANGES is intended for this purpose. Procedure FTS$APPLY_METADATA_CHANGES has no neither input, nor output parameters. The example or use is resulted below:

```
EXECUTE PROCEDURE FTS$APPLY_METADATA_CHANGES
```

Procedure FTS$APPLY_METADATA_CHANGES removes not used triggers which are created at addition of fields in an index.

3.7 Reindexation

To update index data it is necessary to execute reindexation. Reindexation can be executed a call of one of three procedures:

- FTS$FULL_REINDEX

- FTS$REINDEX

- FTS$STARTDAEMON

Procedure FTS$FULL_REINDEX carries out full reindexation for all indexes in full text search system. Procedure FTS$FULL_REINDEX has neither input no output parameters. An example of use:

```
EXECUTE PROCEDURE FTS$FULL_REINDEX
```

To execute reindexation of one index FTS$REINDEX procedure is used. It has one obligatory input parameter - index name, for example:

```
EXECUTE PROCEDURE FTS$REINDEX ('SIMPLE_INDEX')
```

Procedure FTS$STARTDAEMON starts reindexation daemon. Daemon carries out monitoring of table FTS$POOL each 0,1 seconds. Procedure FTS$STARTDAEMON has neither input no output parameters. To start daemon it is necessary to execute procedure FTS$STARTDAEMON, for example:

```
EXECUTE PROCEDURE FTS$STARTDAEMON
```

3.8 Search

For extraction of data from an index FTS$SEARCH procedure is used. Procedure has following input parameters: index name, table name and search query. Obligatory input parameters are index name on which search will be carried out, and a search query. The second input parameter is name of the indexed table can accept NULL value , in this case search is carried out under all tables entering into an index. If the name of the indexed table is specified search will be carried out only under this table.

Output parameters of procedure are table name in which data are found, RDB$DB_KEY value of the found records, search result relevance, fragment of result of search.

Example of search of a word test in index SIMPLE_INDEX and only under table TABLE1, length of a fragment of result of search it is equal 100 symbols:

```
SELECT * from FTS$SEARCH ('SIMPLE_INDEX', 'TABLE1',
'test', 100);
```

Let's admit, that by search has been found two records, in this case result of search for example resulted above can look like:

ROW_ID	SCOPE	RELATION	HIGHLIGHT
85000000E4010000	0,298344343900681	TABLE1	<B>test</B> text
85000000E5010000	0,298344343900681	TABLE1	<B>test</B> text

ROW_ID stores RDB$DB_KEY value of the found record; SCOPE stores con-

formity of the found record to a search condition; RELATION stores table name in which record has been found; HIGHLIGHT stores the text fragment containing a line that satisfy to search conditions.

Using RDB$DB_KEY value it is possible to select found records from corresponding tables, for example:

```
SELECT B.* from FTS$SEARCH ('SIMPLE_INDEX', 'TABLE1',
'test') as A LEFT JOIN TABLE1 as B on A.FTS$DB_KEY =
B.RDB$DB_KEY
```

The result of executing such query will be:

F_ID	F_VCHAR
3	test text
6	test text

3.9 Search query syntax

Use in terms of symbols "?" And "*" allows to carry out wildcard search. In this case a symbol "?" replaces one any symbol, but symbol "*" replaces any quantity of symbols, for example:

```
te?t test* tes*t
```

Attention! The search query cannot beguns with symbols "?" or "*".

To do a fuzzy search use the tilde, "~", symbol at the end of a single word term. For example to search for a term similar in spelling to "roam" use the fuzzy search.

Lucene allows to change the significance value of terms in a search query. For example, you search for a phrase "Hello world" and want, that the word "world" was more significant. The importance of a term in a search query can be increased, using a symbol "^" after which the significance value is underlined. In a following example the importance of a word "world" is four times more than importance of a word "Hello" which is by default equal to one.

```
"Hello world^4"
```

Boolean operators allow terms to be combined through logic operators. Lucene supports AND, "+", OR, NOT and "-" as Boolean operators.

Attention! Boolean operators should be specified by all capital letters.

3.9.1 OR Operator

The OR operator is the default conjunction operator. It means that if there is no boolean operator between two terms in search query operator OR is used. Thus the search system finds the document if one of specified in a search query term is present in it. An alternative designation of operator OR is symbol "||".

```
"Hello world" "world"
```

Equal to:

```
"Hello world" OR "world"
```

3.9.2 AND operator

AND Operator specifies that booth terms united by the operator should exist at the text. An alternative designation of the operator is symbol "&&"", for example:

```
"Hello" AND "world"
```

3.9.3 "+" operator

The operator "+" specifies that the word following it should exists at the text necessarily. For example, to search records which must contain a word "hello" and can contain a word "world", the search query can look like:

```
+Hello world
```

3.9.4 NOT operator

NOT operator allows to exclude a term following it from search results. Instead of word NOT the symbol can be used "!". For example, to search records which must contain a word "hello" and not contain a word "world", the search phrase can look like:

```
"Hello" NOT "world"
```

> **Attention! NOT operator can not be used with one term only, search with such condition will not return results, for example:**

```
NOT 'world'
```

3.9.5 "-" operator

This operator is similar to NOT operator. Example of use:

```
"Hello" -"world"
```

3.9.6 Boolean operators grouping

The Lucene query parser supports grouping of boolean operators. Let's admit, it is necessary to find either a word "word" or a word "dolly" and the word "hello" is obligatory, for this purpose used such search query:

```
"Hello"&&("world"||"dolly")
```

3.9.7 Escaping special characters

Lucene supports escaping special characters that are part of the query syntax. To escape these character use symbol "\" before the character
The current list special characters are:

```
+ - && || ! ( ) { } [ ] ^ " ~ * ? : \
```

For example to search for (1+1):2 use the query:

```
\ (1 \ +1 \) \: 2
```

More detailed description of syntax is located on Lucene official site: http://lucene.apache.org/java/2_3_1/queryparsersyntax.html.