Working with Documents in Databases

Marian DÂRDALĂ, Cristian IONIŢĂ
Academy of Economic Studies, Bucharest, România
dardala@ase.ro, Cristian.Ionita@softmentor.ro

Using on a larger and larger scale the electronic documents within organizations and public institutions requires their storage and unitary exploitation by the means of databases. The purpose of this article is to present the way of loading, exploitation and visualization of documents in a database, taking as example the SGBD MSSQL Server. On the other hand, the modules for loading the documents in the database and for their visualization will be presented through code sequences written in C#. The interoperability between averages will be carried out by the means of ADO.NET technology of database access.
Keywords: *interoperability, documents, database, full text search.*

1 Introduction

The need to store unconventional data in databases made the producers of SGBD-s diversify the types of data associated to the fields of a table. Thus, it appeared the *image* type for manipulating the image type fields, and the generic BLOB type (Binary Large Object) for storing the data that are not structured as length and representation format. By the means of BLOBs was solved the problem of storing the documents in databases, and for filtering them, on the basis of their content (words, expressions, dictionary) were added new operators to the SELECT phrase. Taking into account the diversity of the types of documents, for the access to the document content, the search engine uses filters.

2. Defining the tables and preparing the database for working with documents

In a database, a table that contains fields should have at least three fields:
- the primary key of the table;
- the type of documents;
- the document itself.

The field that plays the key role identifies in a unique way a tuple from the table - that is a document. The type of this field was defined by user according to the identification data of the documents.
The type of document is a literal that memorizes the format of the document stored in that tuple, as *.txt*, *.pdf* etc. This field is very important because the access to the content of the document will be done through different procedures according to the type of document. Thus, for indexing the documents in *pdf* format, as well as for the access to the content of a *pdf* document, the Adobe PDF IFilter should be inserted in the system. For obtaining all the formats recognized by the system of access to the documents, can be used the command (stored procedure) sp_help_fulltext_system_components with the filter parameter:

```
sp_help_fulltext_system_components filter
```

The result will appear under the form of a table like in Figure 1.
It can be noticed that *Adobe PDF IFilter* application installs in the system a library with dynamic binding called *AcroIF.dll* that contains the routines of access to the content of a *pdf* document.
After the database was created, to activate the full-text search component on that database, the next command should be executed:

```
sp_fulltext_database enable
```

The field that memorizes the document itself is *varbinary (MAX)* type and corresponds to *BLOB* type generic.
The creation of the *documente* table with the above described structure is done with the command:

```
create table documente ( cheie varchar(50) primary
key, tipdoc varchar(8), doc varbinary(MAX) )
```

To be able to carry out search operations on documents, there will be created a catalogue containing the search index on the content basis. Thus, it will be created a folder (*cat_doc*) in which the catalogue will be stored. Moreover, there can be chosen also searches insensitive to accents, that is considering the *a* character equal with *á* character or with *â* character:

```
CREATE FULLTEXT CATALOG [cat_doc] IN PATH
N'd:\temp' WITH ACCENT_SENSITIVITY = OFF
```

After creating the catalogue, the search index for searches in documents will be constructed, index associated to the index of the primary key of the *documente* table (PK__documente__7C8480AE); the index is stored in the catalogue *cat_doc* and is updated automatically when there are changes of content.

```
CREATE FULLTEXT INDEX ON [dbo].[documente]
KEY INDEX [PK__documente__7C8480AE] ON
[cat_doc] WITH CHANGE_TRACKING AUTO
```

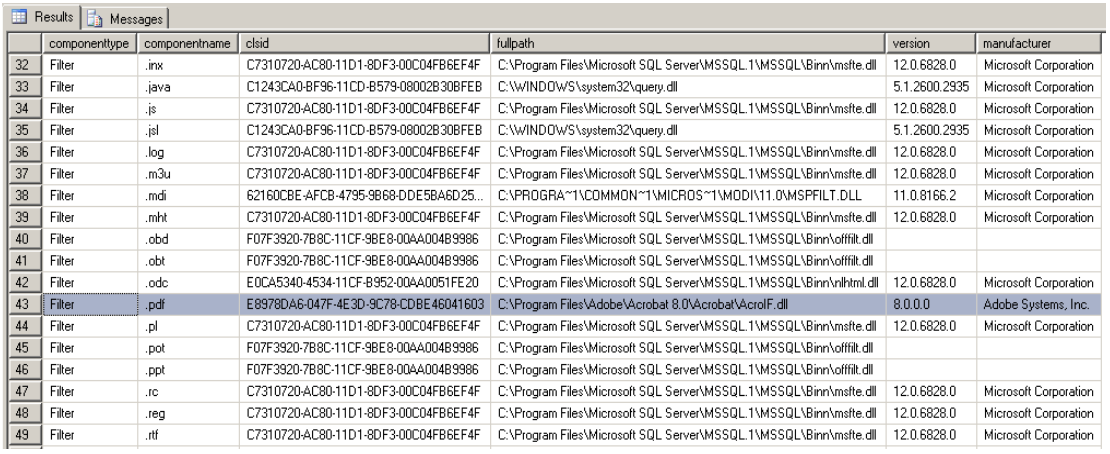
Through the command:

```
ALTER FULLTEXT INDEX ON [dbo].[documente]
ADD ([doc] TYPE COLUMN [tipdoc] LANGUAGE
[English])
```

It is updated the index for searches in documents, of the *documente* table, by adding the field according to which will be done the indexing for searches based on content (*doc*) and on attribute (*tipdoc*) that indicates the format of the document stored in that tuple.

After establishing the options regarding the configuration of the index for searches in documents, it is activated through the command:

```
ALTER FULLTEXT INDEX ON [dbo].[documente]
ENABLE
```


Results | Messages

	componenttype	componentname	clsid	fullpath	version	manufacturer
32	Filter	.inx	C7310720-AC80-11D1-8DF3-00C04FB6EF4F	C:\Program Files\Microsoft SQL Server\MSSQL.1\MSSQL\Binn\msfte.dll	12.0.6828.0	Microsoft Corporation
33	Filter	.java	C1243CA0-BF96-11CD-B579-08002B30BFEB	C:\WINDOWS\system32\query.dll	5.1.2600.2935	Microsoft Corporation
34	Filter	.js	C7310720-AC80-11D1-8DF3-00C04FB6EF4F	C:\Program Files\Microsoft SQL Server\MSSQL.1\MSSQL\Binn\msfte.dll	12.0.6828.0	Microsoft Corporation
35	Filter	.jsl	C1243CA0-BF96-11CD-B579-08002B30BFEB	C:\WINDOWS\system32\query.dll	5.1.2600.2935	Microsoft Corporation
36	Filter	.log	C7310720-AC80-11D1-8DF3-00C04FB6EF4F	C:\Program Files\Microsoft SQL Server\MSSQL.1\MSSQL\Binn\msfte.dll	12.0.6828.0	Microsoft Corporation
37	Filter	.m3u	C7310720-AC80-11D1-8DF3-00C04FB6EF4F	C:\Program Files\Microsoft SQL Server\MSSQL.1\MSSQL\Binn\msfte.dll	12.0.6828.0	Microsoft Corporation
38	Filter	.mdi	62160CBE-AFCB-4795-9B68-DDE5BA6D25...	C:\PROGRA~1\COMMON~1\MICROS~1\MODI\11.0\MSPFILT.DLL	11.0.8166.2	Microsoft Corporation
39	Filter	.mht	C7310720-AC80-11D1-8DF3-00C04FB6EF4F	C:\Program Files\Microsoft SQL Server\MSSQL.1\MSSQL\Binn\msfte.dll	12.0.6828.0	Microsoft Corporation
40	Filter	.obd	F07F3920-7B8C-11CF-9BE8-00AA004B9986	C:\Program Files\Microsoft SQL Server\MSSQL.1\MSSQL\Binn\offfilt.dll		
41	Filter	.obt	F07F3920-7B8C-11CF-9BE8-00AA004B9986	C:\Program Files\Microsoft SQL Server\MSSQL.1\MSSQL\Binn\offfilt.dll		
42	Filter	.odc	E0CA5340-4534-11CF-B952-00AA0051FE20	C:\Program Files\Microsoft SQL Server\MSSQL.1\MSSQL\Binn\nlhtml.dll	12.0.6828.0	Microsoft Corporation
43	Filter	.pdf	E8978DA6-047F-4E3D-9C78-CDBE46041603	C:\Program Files\Adobe\Acrobat 8.0\Acrobat\AcroIF.dll	8.0.0.0	Adobe Systems, Inc.
44	Filter	.pl	C7310720-AC80-11D1-8DF3-00C04FB6EF4F	C:\Program Files\Microsoft SQL Server\MSSQL.1\MSSQL\Binn\msfte.dll	12.0.6828.0	Microsoft Corporation
45	Filter	.pot	F07F3920-7B8C-11CF-9BE8-00AA004B9986	C:\Program Files\Microsoft SQL Server\MSSQL.1\MSSQL\Binn\offfilt.dll		
46	Filter	.ppt	F07F3920-7B8C-11CF-9BE8-00AA004B9986	C:\Program Files\Microsoft SQL Server\MSSQL.1\MSSQL\Binn\offfilt.dll		
47	Filter	.rc	C7310720-AC80-11D1-8DF3-00C04FB6EF4F	C:\Program Files\Microsoft SQL Server\MSSQL.1\MSSQL\Binn\msfte.dll	12.0.6828.0	Microsoft Corporation
48	Filter	.reg	C7310720-AC80-11D1-8DF3-00C04FB6EF4F	C:\Program Files\Microsoft SQL Server\MSSQL.1\MSSQL\Binn\msfte.dll	12.0.6828.0	Microsoft Corporation
49	Filter	.rtf	C7310720-AC80-11D1-8DF3-00C04FB6EF4F	C:\Program Files\Microsoft SQL Server\MSSQL.1\MSSQL\Binn\msfte.dll	12.0.6828.0	Microsoft Corporation

Fig.1. Filters associated to the formats of documents

3. Loading documents in the database and their visualization

In order to store the document in a table, it is taken over from a folder and it is formed a data flow that will be inserted in the field that memorizes the document. At the same time, in the *tipdoc* field, the type of document is memorized under the form of a row of characters. The loading of the document in the *documente* table involves adding a new tuple in the relationship, as it can be noticed in the next sequence:

```
// database connection opening
string cstr = @"Data Source=MM-LAPTOP\SQLEXPRESS;Initial Catalog=test_doc;Integrated Security=True";
SqlConnection conn = new SqlConnection(cstr);
try {  conn.Open();   }
catch (Exception ex)
{
        MessageBox.Show(ex.Message);
}

// we read the data from the file and store it into a byte array
FileStream fs = File.OpenRead("d:\\ex.pdf");
byte[] vecb = new byte[fs.Length];
fs.Read(vecb, 0, (int)fs.Length);
```

```
// we insert the new row into the table
string sinsert = "Insert into documente values(@mcheie, @mtip, @mdoc)";
SqlCommand cinsert = new SqlCommand(sinsert, conn);
cinsert.Parameters.Add(new SqlParameter("@mcheie",(string)"100"));
cinsert.Parameters.Add(new SqlParameter("@mtip",(string)".pdf"));
cinsert.Parameters.Add(new SqlParameter("@mdoc",vecb));
cinsert.ExecuteNonQuery();
```

Extracting a document from *documente* table was carried out by storing the data flow in the *doc* field in a temporary folder with *temp* name and the extension given by the value of *tipdoc* attribute. In the example presented there are used *pdf* documents - that is why the folder has the name *temp.pdf*. Visualizing the document was done by using the method *Navigate*, in the control of type *WebBrowser* (*webb*) included in a dialogue window (*viz*).

```
// database connection opening
SqlConnection sc = new SqlConnection(@"Data Source=MM-LAPTOP\SQLEXPRESS;
                                        Initial Catalog=test_doc;Integrated Security=True");
sc.Open();

// we build and execute the SELECT command
string sirq = "Select * from documente where cheie= '100' ";
SqlCommand sqc = new SqlCommand(sirq, sc);
SqlDataReader dr;
bool vb;
dr = sqc.ExecuteReader();
vb=dr.Read();
if(vb==false)
{
        MessageBox.Show("Document inexistent!!! ");
        return;
}

// we read the document into the vecb variable
System.Data.SqlTypes.SqlBytes vecb = dr.GetSqlBytes(2);
sc.Close();

byte[] vbytes = new byte[vecb.Stream.Length];
vecb.Stream.Read(vbytes, 0, (int)vecb.Stream.Length);

// we create the local file
string calefisurl = "file:///" + Application.StartupPath + "/temp.pdf";
string calefis = Application.StartupPath + "/temp.pdf";
FileStream fs = File.Create(@calefis);
fs.Write(vbytes, 0, vbytes.Length);
fs.Close();

// we open the document inside the WebBrowser control
Viz_doc viz = new Viz_doc();
viz.webb.Navigate(@calefisurl);
viz.ShowDialog();
```

In the following program sequence it is presented by the operation of extracting and visualization of a document from the *documente* table in a *Web* application (developed in the ASP.NET technology).

```
// database connection opening
SqlConnection con = new SqlConnection();
con.ConnectionString = @"Data Source=MM-LAPTOP\SQLEXPRESS;
                                        Initial Catalog= test_doc; Integrated Security=True";
SqlCommand command =
         new SqlCommand("SELECT LEN(doc), doc FROM documente WHERE cheie='100' ", con);
con.Open();

using (SqlDataReader reader = command.ExecuteReader(CommandBehavior.SequentialAccess))
{
        if (reader.Read())
```

```
            {
// we clear the buffers used for response
                this.Response.Clear();
// we disable buffering
                this.Response.Buffer = false;
// we add a MIME header to inform the browser about the file type
                this.Response.ContentType = "application/pdf";
// we instruct the browser to save the file locally rather then open it
                Response.AddHeader(@"Content-Disposition", "attachment; filename=tmp.pdf");
                long fileLen = reader.GetInt64(0);
                Response.AddHeader("Content-Length", fileLen.ToString());

                byte[] buffer = new byte[FileChunkSize];
                long count = 0;
                long offset = 0;
// we send the file in small chunks
                while ((count = reader.GetBytes(1, offset, buffer, 0, FileChunkSize)) > 0)
                {
                   this.Response.OutputStream.Write(buffer, 0, (int)count);
                   offset += count;
                }
            }
reader.Close();
```

It can be noticed that the document is loaded at the client progressively, on data blocks; a data block was dimensioned at a number of *FileChunkSize* bites:

```
        const int FileChunkSize = 1024 * 128;
```

4. Searching the documents on the basis of their content

The search component on the basis of content is a new characteristic added to SGBDs, allowing thus the efficient search in large documents. For searches by using conventional data fields, there can be used, for making up the conditions, equality, relational and logical operators. For searches based on patterns, the *like* operator is used. The main difference between using the *like* operator, respectively the full-text search facility, consists of the fact that the *like* operator searches sequentially, while using the full-text search facility, the search is based on index, so the operation is much faster.

The search operators that enriched the *select* phrase are: *contains*, *containstable*, *freetext* and *freetexttable*. The operators *contains* versus *freetext,* respectively *containstable* versus *freetexttable,* are used the same way, differing only the search operation which is larger in the case of using the *contains* operators. The reason is that in case of using the *freetext* clause there are taken into account also derivates of the searched word. The search carried out on the basis of these operators can be done based on simple words, on expressions, on radicals of words, using logical operators for defining the search criteria, on the proximity of words in the text, and on the dictionary specified through *language* clause.

There will be presented examples of using the searches based on contents, using different facilities and operators:

- search based on simple words:

```
select cheie from documente where freetext(doc,
                    'asfalt' )
```

there will be displayed the keys of those tuples that contain documents in which *asphalt* word is found;

- search based on expressions:

```
select cheie from documente where contains(doc, '
                "panta de scurgere" ')
```

- search based on defining a logic search expression, that is displaying the keys of the tuples that contain documents in which *cale* or *ferata* words are found.

```
select cheie from documente where contains(doc, '
                "cale" or "ferata" ')
```

- search based on using the * character with role of substitution of a group of characters:

```
select cheie from documente where contains(doc, '
                "transp*" ')
```

there will be displayed the keys of documents in which there can be found words that begin with *transp* and continue with other characters, such as: *transport*, *transplant* or *transparent*.

- by using the *containstable* clause in the *select* phrase, there can be obtained the result of filtering under the form of a relation containing the key and the rank (*rank*) for evaluating the representativity of the search operation:

```
select * from containstable(documente, doc, ' "cale" and "ferata" ') order by rank desc
```

The result will be ordered decreasingly according to the rank. In the next example the *near* operator is used for searching those documents that contain the *beton* word near *armat* word:

```
select * from containstable(documente, doc, ' "beton" near "armat" ') order by rank desc
```

The rank of satisfying the search condition depends upon the frequency of finding the condition in the document, upon the proximity of the words involved in the search etc.

There is also the possibility to search in the text of a document various forms of a word. To exemplify, the following select phrase searches in the table documents that contain inflectional forms of *metal* word, that is *metalic*, *metalifer* etc., depending on the language indicated by the *language* clause or by the set language, which is considered current. The inflectional form of a word depends on the dictionary of the used language.

```
SELECT cheie FROM documente WHERE CONTAINS(doc, ' FORMSOF (INFLECTIONAL, "metal") ', language 'Romanian-Romanian');
```

To visualize the languages that the current instant recognize, there can be consulted the system table *sys.fulltext_languages*, with the command:

```
select * from sys.fulltext_languages
```

5. Conclusions

Using the BLOB generic type to store unconventional data does not involve only working with documents in a *MSSqlServer* database. The same thing happens when we want to store images, sounds, video and animations in a database, from the point of view of their storage and visualization. Yet, as we mentioned in the article, the specific part of working with documents refers to the possibility of searches based on the contents of documents, by constructing the search index on the basis of content.

References

1. Dârdală, M., Reveiu, A., *Controlling Media Type in Multimedia Databases*, Proceedings of the Romanian Symposium on Computer Science, Universitatea *Alexandru Ioan Cuza*, Iaşi, 2006;
2. MacDonald, M., Szpuszta, M., *Pro ASP.NET 2.0 in C# 2005*, Editura Apress, 2005, USA;
3. Sack, J., *SQL Server 2000 Fast Answers for DBAs and Developers*, Editura Apress, 2005, USA;
4. Smeureanu, I., Dârdală, M., Reveiu, A., *Visual C#.NET,* Editura CISON, Bucureşti, 2004;
5. *** * *,** *Microsoft Developer Network Library,* Microsoft Press, 2005;
6. *** * *,** Microsoft SQL Server 2005 Implementation and Maintenance, Microsoft Press, USA, 2006;