STORAGE OPTIMIZATION OF EDUCATIONAL SYSTEM DATA

Catalin BOJA[1]
PhD Candidate, University Assistant, Economic Informatics Department
Academy of Economic Studies, Bucharest, Romania

E-mail: CatalinBoja@ie.ase.ro

Abstract: *There are described methods used to minimize data files dimension. There are defined indicators for measuring size of files and databases. The storage optimization process is based on selecting from a multitude of data storage models the one that satisfies the propose problem objective, maximization or minimization of the optimum criterion that is mapped on the size of used disk memory. The paper describes different solutions that are implemented to minimize input/output file size for a software application that manages educational system data.*

Key words: file, database, optimization, data, educational system.

1. File and database dimension

It is considered a finite set of N elements, El_1, El_2, …, El_N. It is defined an optimum criterion and it is the El_i element that maximize/minimize the function associated to the optimum criterion. This defines the optimization problem in the informatics field, the framework being applied to any software quality characteristic that is included in the optimization process.

There is considered a collectivity C composed from the elements, c_1, c_2, …, c_N, where N represents the total number of elements. Each element c_i it is described using M characteristics, A_1, A_2, …, A_M.

For each of the software characteristics A_j there are used values or attributes to describe measured levels of c_i elements. The values or attributes are described using arrays of characters or strings. As a result, the s_{ij} characters string describes the levels of the A_j characteristic for the c_i element.

The s_{ij} string is characterized by the L_{ij} length which is represented by a number of symbols.

The problem of storing data into files or conventional databases suppose using homogenous data structures for each of the collectivity articles. To characterize the required memory space there are defined a series of indicators that will measure this dimension and that will provide a quantitative approach of the problem.

In order to determine the memory space reserved by a software application for its data, there are accomplished the next steps:

- there are recorded into a table with n lines and m columns the descriptions of C collectivity elements;
- for each characteristic it is selected the element that has the maximum length;

$$L^j_{\max} = \max_{1 \le i \le N} \{ s_{ij} \}$$

- it is constructed the structure used to describe the characteristic elements; its form is

```
struct struc
{
 type comp1;
 type comp2;
 ..............
 type compi;
 ..............
 type compM
}
```

and it is defined the indicator LG(type $comp_i$) = L^i_{max}.

- it is obtained the database with fixed length articles, BDF.

In the database, for each element of the C collectivity it recorded an article with the dimension equal with $L_{art} = \sum_{j=1}^{M} L^j_{\max}$.

For the students collectivity STUD, described in table 1, there are measured fields length, maximum dimensions and based on that it is determined the article size.

Table 1. Description of students colectivity

No.	Name	First name	Height	Gender	City	Age	Date of Birth	School
1	Anghelache$_{(10)}$	Ion$_{(3)}$	132$_{(3)}$	Male$_{(4)}$	Bucharest$_{(9)}$	12$_{(2)}$	24/11/93$_{(8)}$	173$_{(3)}$
2	Bujor$_{(5)}$	Elena$_{(5)}$	126$_{(3)}$	Female$_{(6)}$	Iasi$_{(4)}$	12$_{(2)}$	12/07/93$_{(8)}$	10$_{(2)}$
3	Biteanu$_{(7)}$	Cristian$_{(8)}$	125$_{(3)}$	Male$_{(4)}$	Ploiesti$_{(8)}$	10$_{(2)}$	14/04/95$_{(8)}$	154$_{(3)}$
4	Cretu$_{(5)}$	Ion$_{(3)}$	132$_{(3)}$	Male$_{(4)}$	Bucharest$_{(9)}$	12$_{(2)}$	06/05/93$_{(8)}$	3$_{(1)}$
5	Cretu $_{(5)}$	Roxana$_{(6)}$	137$_{(3)}$	Female$_{(6)}$	Bucharest$_{(9)}$	14$_{(2)}$	27/05/91$_{(8)}$	189$_{(3)}$
6	Danciulescu$_{(11)}$	Mihai$_{(5)}$	137$_{(3)}$	Male$_{(4)}$	Ploiesti$_{(8)}$	14$_{(2)}$	16/07/91$_{(8)}$	56$_{(2)}$
7	Danciulescu$_{(11)}$	Ion$_{(3)}$	135$_{(3)}$	Male$_{(4)}$	Bucharest$_{(9)}$	14$_{(2)}$	19/07/91$_{(8)}$	133$_{(3)}$
8	Ene$_{(3)}$	Catalin$_{(7)}$	126$_{(3)}$	Male$_{(4)}$	Ploiesti$_{(8)}$	10$_{(2)}$	05/03/95$_{(8)}$	43$_{(2)}$
9	Ionescu$_{(7)}$	Irina$_{(5)}$	131$_{(3)}$	Female$_{(6)}$	Bucharest$_{(9)}$	11$_{(2)}$	22/06/94$_{(8)}$	17$_{(2)}$
10	Ionescu$_{(7)}$	Catalin$_{(7)}$	128$_{(3)}$	Male$_{(4)}$	Iasi$_{(4)}$	12$_{(2)}$	11/02/93$_{(8)}$	23$_{(2)}$
SUM	**71**	**52**	**30**	**46**	**69**	**20**	**80**	**23**
TOTAL = 391								

In the parentheses there is described the dimension of data values as number of characters.

Table 2. Fields length for students collectivity.

	Name	First name	Height	Gender	City	Age	Date of Birth	School
$L^j_{\max}$	11	8	3	6	9	2	8	3
L_{art} =	50							

The database length, L_{BD}(STUD), is determined with the relation $L_{BD} = N * L_{art}$. For the students database the size is L_{BD}(STUD) = 500 bytes.

It is observed that some of the article fields are smaller than what is defined as maximum length. As presented next in this paper, this fact will increase the database size and will highlight an inefficient data storage solution from the memory space viewpoint.

The memory use, or efficiency, degree is determined by the relation:

$$G_U = \frac{\sum_{i=1}^{N}\sum_{j=1}^{M}\lg(s_{ij})}{L_{BD}}$$

The memory non-use, or inefficiency, degree is determined with the relations

$$G_{NU} = 1 - Gu \text{ or } G_{NU} = \frac{\sum_{i=1}^{N}\sum_{j=1}^{M}\left(L_{\max}^{j} - \lg(s_{ij})\right)}{L_{BD}}$$

For the considered example, the STUD database, the indicators value are G_U = 391/500 = 0,78 and G_{NU} = 0,22. These values are used as a comparative base to evaluate the efficiency of proposed solutions from the view point of reserved memory space.

2. Optimization of used memory size

For the ***first variant***, it is considered a separator character, a marker used to indicate the end of a array of characters, as "\0" in C/C++ or other programming languages. This symbol it is noted with α . The s_{ij} string to which is appended this string end marker becomes s'_{ij}.

$$s_{ij} \,||\, \alpha = s'_{ij}\,;\ \lg(s'_{ij}) = \lg(s_{ij}) + 1$$

There are concatenated the s'_{ij} strings, that are used to described the collectivity elements, c_i with i = 1..N. The length indicator that measures the dimension of the database element has the relation.

$$L_{art}^{i} = \sum_{j=1}^{M}\lg(s_{ij}) + M$$

The N elements database dimension, is in this case equal with $L_{BD} = \sum_{i=1}^{N} L_{art}^{i}$

The memory use efficiency degree of this database format is:

$$G_u = \frac{L_{BD} - N*M}{L_{BD}} \text{ or } G_{NU} = 1 - G_u = \frac{N*M}{L_{BD}}$$

In the case of the students STUD database, applying this solution will conduct to the data form:

Anghelache$_{(10)}$#Ion$_{(3)}$#132$_{(3)}$#Male$_{(4)}$#Bucharest$_{(9)\#}$12$_{(2)\#}$24/11/93$_{(8)}$#173$_{(3)}$#Bujor$_{(5)}$ #Elena$_{(5)}$#126$_{(3)}$#Female$_{(6)\#}$Iasi$_{(4)}$#12$_{(2)}$#12/07/93$_{(8)}$#10$_{(2)}$...

For this data storage variant, the database dimension is given by the total number of article characters to which is added the number of bytes reserved for the string end

markers. The length of the first article of table 1 is $L_{art}^{1} = 42 + 7 = 39$, where 42 represents the number of characters contained in the article.

Values of previous defined indicators are L_{BD} = 391 + 70 = 461 bytes, G_{NU} = 70/461 = 0,15 and G_U = 0,85.

To optimize means to find the modality used to construct a database that has a dimension smaller than other databases of same collectivity, but based on a different data storage technique.

For this solution, there must be taken into account the particular situations that will conduct to worse results. These cases are described by the existence of a data set in which every article size is equal with the maximum dimension. If $lg(s_{ij}) = L_{\max}^{j}$ cu i=1,2, …, N and j=1,2, …, M it results that $lg(s'_{ij}) = L_{\max}^{j} + 1$ and the database BD' has a dimension equal with $L_{BD'} = L_{BD} + M * N$.

For a database with ten articles that have eight fields and $L_{\max}^{j}$ = 50, applying these solution will generate a $L_{BD^{over}}$ = 500 + 70 = 570 bytes database. The overuse degree G_D is given by the relation $G_D = \frac{L_{BD^{over}}}{L_{BD}} - 1$. For the analyzed situation, the indicator value is G_D = 0,14. Based on this result, it is concluded that in this particular case, the storage variant will generate a database with a 14% increased memory size.

The second variant uses data conversions and compressions that will reduce the database length. Numerical values represented in the database by characters arrays are converted, representing them in binary integer or floating format. For example, the values that describe the student height, will necessitate one byte if there are saved in numerical format as unsigned integers.

For the table 1 data, the internal binary format to be associated to fields values is determined based on the variable maximum value and on the fundamental data types defined by the programming language used to develop the software application. Choosing C/C++ as programming medium, the numerical fields of the *stud* structure will require the memory space described in table 3.

Table 3. Memory space reserved by article numerical fields

Field:	Height	Age	Date of Birth	School
Dimension:	1 byte	1 byte	3 bytes	1 byte
C/C++ used data type	unsigned int	unsigned int	structure of 3 unsigned int	unsigned int

By storing numerical data, using binary format, it is obtaining a minimization in memory size. Base don that, it results an article which contains:

- end mark fields as field1, field2, field4 and field5;
- fields with standard imposed by conversion length as field3, field6, field7 and field8.

The length of the compressed database BD'' is

$$L_{BD''} = \sum_{i=1}^{N} L_{comp}^{i} + k * N + N$$

where k represents the number of fields that have end separator. For the others N-k fields, through compression/conversion there have been obtained constant lengths. Also, it will be used a marker to indicate the end of an article. For the table 1 example, the first article will be saved in the form

$Anghelache_{(10)}\#Ion_{(3)}\#Male_{(4)}\#Bucharest_{(9)}\#132_{(1)}12_{(1)}24/11/93_{(3)}173_{(1)}\#$

and it has the dimension equal with $L^1_{art} = 32+\#_{(5)} = 37$ bytes.

In the end, it is obtained the total length of 298 bytes for all 10 records and the database dimension is $L_{BD''} = 298 + 4*10 + 10 = 348$, because k = 4 fields have string markers.

For this data storage variant, the degree of space use efficiency has the value

$$G_u = \frac{L_{BD''} - k * M * N - N}{L_{BD''}} = 0{,}85$$

where *M* represents the size of the marker, in this case equal with one.

The solution proposed in previous variants is improved by ***the third variant*** by defining a method that will not use end markers. The working context and the implementation of the solution impose a series of restrictive conditions that will the base of used data model.

It is considered the structure *art* that combines into a single article all the data needed to process the entity. Its format is:

art { tip_1 $camp_1$; tip_2 $camp_2$; …; tip_s $camp_s$;}

In order to store data and minimize reserved memory space, it is implemented a method to arrange the fields in a way in which two adjacent fields $camp_i$ and $camp_{i+1}$ does not have same type, $tip_i \neq tip_{i+1}$ cu *i = 1..s-1*. The situation allows the elimination of filed end markers because the cross from one data type to another one is announced by the different internal format.

For this approach, the size of a database that contains *nart* articles of this type, is determined by the indicator $L_{BD} = \sum_{i=1}^{nart}\sum_{j=1}^{s} s_{ij}$, in which s_{ij} represents the length of the *j* field from the *i* article.

It is considered the data model implemented by the software application that manages the database described in table 1. The difference between recorded data types allows the use of current data storage variant, obtaining the article:

stud { Name; Height; First_name; Age; Gender; Date_of_Birth; City; School;}

Implementing this method, the first article of the database has its dimension reduced to $L^1_{comp} = 10 + 1 + 3 + 1 + 4 + 3 + 9 + 1 = 32$.

It is observed that the fields dimension it is not modified from the previous solution and it is obtained the total length of 298 bytes for all ten articles. The memory space reserved for the entire database is $L_{BD''} = 298 + 10 = 308$ bytes. The reduced size is the result of using only the article end markers.

For this data storage version, the indicator used to measure the efficiency of space utilization has the value $G_u = 0,96$, most of the bytes representing data used in the processing activity.

In ***fourth variant***, it is considered a vocabulary V_i that contains the set of distinct values of the C_i collectivity elements.

The V_i set is described by the elements $V_i = \{v_{i1}, v_{i2}, \ldots, v_{ih}\}$, where v_{i1}, v_{i2} are words from the V_i vocabulary and $lg(v_{ji})$ describes the length of the v_{ij} word.

Any array of characters s_{ij} that represents the value of the *j* field of *i* article exists in the collectivity vocabulary, V_i.

The supposition based on which is implemented this solution requires a the presence of a large number of data and a limited number vocabulary. The greater repeating degree of values means an increase efficiency of the method.

Each vocabulary word occupies a fixed position. The new form of the article will contains the value position in vocabulary, replacing the characters array by a number.

The steps required for a proper application of the method are:

- it is defined the vocabulary $V_1, V_2, \ldots, V_M$ for the all M characteristics used to describe collectivity elements;
- the vocabularies are stored in a particular database BDV that the length equal with

$$Lg(BDV) = lg(V_1) + lg(V_2) + \ldots + lg(V_M) = \sum_{k=1}^{M} lg(V_k)$$

- it is developed the collectivity database, BDC, using values positions from the vocabulary

$$Lg(BDC) = \sum_{i=1}^{N}\sum_{j=1}^{M} lg(Poz_{ij})$$

where Poz_{ij} is the field that represents the value vocabulary position for the c_i element and V_i vocabulary.

If it is defined that all the positions are represented by a field with length equal with L'_{voc}, then the collectivity database length is

$$Lg(BDC) = M*N*L'_{voc}.$$

For the table 1 example it is defined the common vocabulary
VV = { Anghelache (1), Bujor (2), Biteanu (3), Cretu (4), Danciulescu (5), Ene (6), Ionescu (7), Ion (8), Elena (9), Cristian (10), Roxana (11), Mihai (12), Catalin (13), Irina(14), Male(15), Female(16), Bucharest(17), Iasi(18), Ploiesti(19)}

In parentheses are defined the values positions in the VV vocabulary. If it is considered the maximum length $L_{max} = 11$ for all the VV vocabulary values then Lg(BDV) = 19*11 = 209 bytes.

The positions required one byte, $L'_{voc} = 1$, so the size of the database article if given by the relation

$$L^i_{art} = \sum_{l=1}^{nc} lg(s'_{il}) + k * L'_{voc}$$

where

nc – number of article fields that have the initial format; if these fields have variable length then it is used an end marker to separate them;

s'_{il} – the string value with the end marker;

k – number of fields that are replaced by their position in the vocabulary;

L_{poz} – the length of the position field.

The size of the compressed database is determined by the indicator

$$L(BDC) = \sum_{i=1}^{N} L_{art}^{i} + L(BDV)$$

For the considered example it is obtained:

$L(BDC) = (6+4*1)_{art1} + (6+4*1)_{art2} + \ldots + (6+4*1)_{art10} + 209 = 100 + 209 = 309$ bytes.

This solution is more improved by minimizing the vocabulary dimension, because its efficiency is directly dependent by the maximum size of vocabulary values and also by their medium size. Because of the elements length variation, the implementation of a fixed size structure will results in a waste of memory space. The use of elements end markers will reduce the reserved space.

Implementing the '#' marker it is defined a vocabulary with the size equal with L(BDV) = 118 + 19 *1 = 137 bytes. In this case the database lengths becomes L(BDC) = 100 + 137 = 237 bytes.

3. Selecting optimization method

There are considered optimization methods M_1, M_2, …, M_t to which are associated modules into a software application intended to optimize educational data storage.

A file F represents the entry data for the considered application.

The result of data processing activity consists in obtaining the files E_1, E_2, …, E_t, that are created by correct optimization modules. The relation between modules and methods is one to one. There are determined the indicators $LG(E_1)$, $LG(E_2)$, …, $LG(E_t)$. To optimize storage files in a automate manner is equivalent to implementing in the software application a module that will select $LG_{min} = \min\{ LG(E_1), LG(E_2), \ldots, LG(E_t)\} = LG(E_k)$. Based on that, it results that the M_k storage methods is the most efficient and it is the method that will be implemented in the final version of the product.

The software application is developed in C programming language and it implements storage techniques previous described.

It is defined the data structure needed to store data regarding the high school students database. It is considered the example described in table 4.

Table 4. Students database

No.	Name	First name	PNC	Height	Weight	School	City
1	Alexandrescu	Ionela	2…	145	47	175	Bucharest
2	Bratescu	Catalin	1…	139	50	175	Buftea
3	Constantin	Adrian	1…	145	50	160	Mihailesti
4	Constantin	Mihai	1…	135	47	163	Bucharest
5	Gheorghe	Florin	1…	137	49	179	Bucharest
6	Ionescu	Gabriela	2…	139	44	3	Bucharest
7	Ionescu	Adrian	1…	132	50	175	Bucharest
8	Popescu	Adrian	1…	135	48	173	Otopeni
9	Popescu	Alina	2…	139	41	160	Bucharest
10	Zamfir	Ion	1…	135	50	3	Buftea

The methods used to store table 4 data are:

- a solution with high use degree in real applications and with a low complicity level is given by the definition of a data structure; this is associated with each of the database articles; the file saving operation is made without auxiliary data processing; the data structure used to memorize students data is

```
struct stud
{
        char nume[13];
        char prenume[9];
        char cnp[14];
        unsigned short int inaltime;
        unsigned char greutate;
        unsigned short int scoala;
        char localitate[11];
};
```

the dimension of the *stud* article is 52 bytes; the dimension of the database that has a normal form by saving the articles in the output file is LG(BDF) = 520 bytes; the cod sequence that writes the data in the file is:

```
void salvareDate(FILE *pfisier, stud *listaStud, int dim)
{
        if(pfisier){
                for(int i=0;i<10;i++){
                        fwrite(&listaStud[i],sizeof(stud),1,pfisier);
                }
        }
}
```

- the data are written in the file using the delimiter marker'#' in order to separate the articles fields; this solution is described by the first version of the storage methods; the numerical values are converted into char arrays before writing them into the file; it is obtained the $BD_{separator}$ database and its dimension is $LG(BD_{separator})$ = 504 bytes; the internal routine used to save data with the corresponding format is

```
void transformare1_OUT(stud *listaStud, int dim)
{
        FILE *pfisOUT = fopen("DateTEST.txt","wb");
        fwrite(&dim,sizeof(int),1,pfisOUT);
        for(int k=0;k<dim;k++){
        unsigned int j;
        char *rez;
        char inaltime[3];
        char greutate[2];
        char scoala[3];
        _itoa(listaStud[k].inaltime,inaltime,10);
        _itoa(listaStud[k].greutate,greutate,10);
        _itoa(listaStud[k].scoala,scoala,10);
        int dim_Articol = strlen(inaltime) + strlen(greutate) + strlen(scoala) +
                strlen(listaStud[k].nume) + strlen(listaStud[k].prenume) +
                strlen(listaStud[k].localitate) + strlen(listaStud[k].cnp);
        rez = new char[dim_Articol+7];
        int i=0;
        for(j=0;j<strlen(listaStud[k].nume);j++,i++)
                rez[i]=listaStud[k].nume[j];
```

```
                rez[i]='#';
                i++;
                for(j=0;j<strlen(listaStud[k].prenume);j++,i++)
                        rez[i]=listaStud[k].prenume[j];
                rez[i]='#';
                i++;
                for(j=0;j<strlen(listaStud[k].cnp);j++,i++)
                        rez[i]=listaStud[k].cnp[j];
                rez[i]='#';
                i++;
                for(j=0;j<strlen(inaltime);j++,i++)
                        rez[i]=inaltime[j];
                rez[i]='#';
                i++;
                for(j=0;j<strlen(greutate);j++,i++)
                        rez[i]=greutate[j];
                rez[i]='#';
                i++;
                for(j=0;j<strlen(scoala);j++,i++)
                        rez[i]=scoala[j];
                rez[i]='#';
                i++;
                for(j=0;j<strlen(listaStud[k].localitate);j++,i++)
                        rez[i]=listaStud[k].localitate[j];
                rez[i]='#';
                i++;
                fwrite(rez,sizeof(char),dim_Articol+7,pfisOUT);
                delete rez;
                }
                fclose(pfisOUT);
}
```

- data are written in the output file using the character marker '#' to separate string values of the *stud* article; numerical data are stored using their binary internal format; this solution represents the implementation of the second storage version; the obtained database, $BD_{numeric}$, has the dimension $LG(BD_{numeric}) = 448$ bytes; the subprogram used to write the data is

```
void transformare2_OUT(stud *listaStud, int dim)
{
        FILE *pfisOUT = fopen("DateTEST2.txt","wb");
        fwrite(&dim,sizeof(int),1,pfisOUT);

        for(int k=0;k<dim;k++)
        {
        unsigned int j;
        char *rez;
        int dim_Articol = strlen(listaStud[k].nume) + strlen(listaStud[k].prenume)
                + strlen(listaStud[k].localitate) + strlen(listaStud[k].cnp);
        rez = new char[dim_Articol+4];

        int i=0;
        for(j=0;j<strlen(listaStud[k].nume);j++,i++)
                rez[i]=listaStud[k].nume[j];
        rez[i]='#';
        i++;
        for(j=0;j<strlen(listaStud[k].prenume);j++,i++)
                rez[i]=listaStud[k].prenume[j];
        rez[i]='#';
```

```
        i++;
        for(j=0;j<strlen(listaStud[k].cnp);j++,i++)
                rez[i]=listaStud[k].cnp[j];
        rez[i]='#';
        i++;
                for(j=0;j<strlen(listaStud[k].localitate);j++,i++)
                rez[i]=listaStud[k].localitate[j];
        rez[i]='#';
        i++;

        fwrite(rez,sizeof(char),dim_Articol+4,pfisOUT);

        fwrite(&listaStud[k].inaltime,sizeof(unsigned short int),1,pfisOUT);
        fwrite(&listaStud[k].greutate,sizeof(unsigned char),1,pfisOUT);
        fwrite(&listaStud[k].scoala,sizeof(unsigned short int),1,pfisOUT);

        delete rez;

        }
        fclose(pfisOUT);
}
```

- data are stored without using separator markers between article fields because the structure of the *stud* article allows the relocation of a numeric field between two string fields; despite the low disk space of the resulting output file, the solution given by the third variant must be modified in practice in order to allow the placement of the marker '#' after each numeric value; this will reduce the effort to write code sequences used to identify inside the file the limit between a string value and a numeric one; the resulting database $BD_{combinat}$, formed without using the marker has the dimension $LG(BD_{combinat}) = 418$ bytes, and the data saving routine is

```
void transformare3_OUT(stud *listaStud, int dim)
{
        FILE *pfisOUT = fopen("DateTEST3.txt","wb");
        fwrite(&dim,sizeof(int),1,pfisOUT);
        char StudentEnd = '#';
        for(int k=0;k<dim;k++)
        {
        fwrite(&listaStud[k].nume,strlen(listaStud[k].nume),1,pfisOUT);
        fwrite(&listaStud[k].inaltime,sizeof(unsigned short int),1,pfisOUT);
        fwrite(&listaStud[k].prenume,strlen(listaStud[k].prenume),1,pfisOUT);
        fwrite(&listaStud[k].greutate,sizeof(unsigned char),1,pfisOUT);
        fwrite(&listaStud[k].cnp,strlen(listaStud[k].cnp),1,pfisOUT);
        fwrite(&listaStud[k].scoala,sizeof(unsigned short int),1,pfisOUT);
        fwrite(&listaStud[k].localitate,strlen(listaStud[k].localitate),1,pfisOUT);
        fwrite(&StudentEnd,sizeof(char),1,pfisOUT);
        }
        fclose(pfisOUT);
}
```

for this solution it is not taken into discussion the reverse operation, used to read data from file;

- data are saved into the file using a symbol vocabulary that contains the distinct string values of article fields; in order to minimize the vocabulary dimension, its elements are separated by the '#' marker; inside the database, these values are replaced by their vocabulary position; the new data structured for the *stud* article is in this case

```
struct pozvocabular
{
        unsigned char poznume;
        unsigned char pozprenume;
        unsigned char pozcnp;
        unsigned short int inaltime;
        unsigned char greutate;
        unsigned short int scoala;
        unsigned char pozloc;
};
```

the new database dimension is $BD_{vocabular}$ and it is obtained by summing the vocabulary dimension and the values zone length, $LG(BD_{vocabular}) = 299 + 124 = 423$ bytes; because the example dataset has reduced size, it is not highlighted this solution efficiency; the code sequence used to convert the database from the normal form to the current one is

```
void transformare4_OUT(stud *listaStud, int dim)
{
        vocabular *Vocabular = NULL;
        vocabular *VocabularEnd = NULL;
        int flag=0;

        FILE *pfisOUT = fopen("DateTEST4.txt","wb");
        fwrite(&dim,sizeof(int),1,pfisOUT);

        // se construieste vocabularul
        pozvocabular elemCurent;
        int elemDictionar = 0;

        for(int k=0;k<dim;k++)
        {
                elemCurent.greutate=listaStud[k].greutate;
                elemCurent.inaltime=listaStud[k].inaltime;
                elemCurent.scoala=listaStud[k].scoala;
                flag = IsInVocabular(listaStud[k].nume,Vocabular);
                if(flag==-1)
                {
                AddVocabular(listaStud[k].nume,Vocabular, VocabularEnd);
                elemCurent.poznume=elemDictionar;
                elemDictionar++;
                }
                else
                        elemCurent.poznume=flag;

                flag = IsInVocabular(listaStud[k].prenume,Vocabular);
                if(flag==-1)
                {
                AddVocabular(listaStud[k].prenume,Vocabular, VocabularEnd);
                elemCurent.pozprenume=elemDictionar;
                elemDictionar++;
                }
                else
                        elemCurent.pozprenume=flag;

                flag = IsInVocabular(listaStud[k].cnp,Vocabular);
                if(flag==-1)
                {
                        AddVocabular(listaStud[k].cnp,Vocabular, VocabularEnd);
```

```
                        elemCurent.pozcnp=elemDictionar;
                        elemDictionar++;
                }
                else
                        elemCurent.pozcnp=flag;

                flag = IsInVocabular(listaStud[k].localitate,Vocabular);
                if(flag==-1)
                {
                AddVocabular(listaStud[k].localitate,Vocabular, VocabularEnd);
                elemCurent.pozloc=elemDictionar;
                elemDictionar++;
                }
                else
                        elemCurent.pozloc=flag;

                fwrite(&elemCurent,sizeof(pozvocabular),1,pfisOUT);
        }
        fclose(pfisOUT);

        pfisOUT = fopen("DateTEST4Vocabular.txt","wb");
        fwrite(&elemDictionar,sizeof(int),1,pfisOUT);

        char caracterVocab = '#';

        if(Vocabular!=NULL)
                for(vocabular *temp = Vocabular;temp!=NULL;temp=temp->next)
                {
                        fwrite(temp->element,strlen(temp->element),1,pfisOUT);
                        fwrite(&caracterVocab,sizeof(char),1,pfisOUT);
                }
        fclose(pfisOUT);
}
```

For each of the described routines there has been recorded a set of parameters, which are described in table 5. The developing environment of current software application is Microsoft Visual Studio 6.0, without using compiler specific optimization options. For measuring the processing effort of implemented solutions it has been used the Visual Studio environment profiler.

Table 5. Parameters recorded for different data storage methods

Output database	Dimension (bytes)	Vocabulary (bytes)	Database (bytes)	Save (mseconds)	Load (mseconds)
BDF	520	-	520	0.032	0.039
$BD_{separator}$	504	-	504	0.618	0.124
$BD_{numeric}$	448	-	448	0.713	0.121
$BD_{combinat}$	418	-	418	0.582	-
$BD_{vocabular}$	124	299	423	1.235	0.233

From the table 5 values it is observed that the processing effort increase depending on the minimization degree of stored data dimension. Despite that $BD_{vocabular}$ has a bigger dimension than the $BD_{combinat}$ one, in real cases, with a great number of data, the last solution will conduct to better results.

Conclusions

Real world collectivities have defined descriptions accordingly to needed objectives. From this point of view it is important to define article structures that are enough flexible so that changes in objectives will not affect them in a radical manner.

In the analysis phase, for each database and file there are developed new storage solutions, the designers' vision having an important impact on that. Taking into discussion and promoting new solutions there are defined the premises for further development of the creative spirit into the direction of adapting all optimization instruments, techniques, methods and algorithms for particular cases. The objective is to obtain numerous different solution for storing data in order to analyze them and to select the one that gives the best results.

For each problem there are defined specific performance criteria and the procedures used to measure optimization effects, providing in this manner the base for variants comparability.

As there is accumulated more experience regarding data storage optimization there will be obtained homogenous databases that have efficient storing techniques.

Based on practical experience there are defined optimal storage procedure, specifying which storage method give best results for a database that has well defined characteristics.

Bibliography

1. Catalin BOJA **Software Multicriterial Optimization**, The Proceedings of the Seventh International Conference of Informatics in Economy, Academy of Economic Studies, Bucharest, Romania, Inforec Printing House, p. 1068 – 1074, May 2005
2. Graeme C. Simsion and Graham C. **Witt Data Modeling Essentials, Third Edition**, Morgan Kaufmann Publishers, 2005
3. IEEE Standards Collection Software Engineering, **Std. 1061-1992 IEEE standard for software quality metrics methodology**, Published by The Institute of Electrical and Electronics Engineers, New York, 1994
4. ISO/IEC 9126 International Standard - Information Technology **Software product evaluation - Quality characteristics and guidelines for their use**, Geneve, Switzerland, 1991
5. Ion IVAN, Catalin BOJA **Statistical methods in software analysis**, ASE Printing House, 2004 (in Romanian)
6. Ion IVAN, Catalin BOJA **Empirical Software Optimization**, Economic Informatics Magazine, vol. IX, nr. 2/2005, Inforec Printing House, Bucharest, 2005 (in Romanian)
7. Ion IVAN, Catalin BOJA **Global Software Optimization**, p. 205 – 214, "**Information Systems & Operations Management – ISOM**" no. 3 Workshop, April 20 – 21, 2005 Romanian American University Master of Economic Informatics, Academy of Economic Studies Master BRIE, Bucharest, ProUniversalis Printing House, 2005
8. Ion IVAN, Catalin BOJA **Optimizarea bicriteriala a produselor program**, Economic Informatics Magazine, vol. 10, nr. 1/2006, p. 17 – 24
9. Ion IVAN, Gh. NOSCA, O PARLOG, S. TCACIUC **Data quality**, Inforec Printing House, Bucharest, 1999 (in Romanian)

10. Gh. Nosca, I. Ivan, A. Parlog **Data Quality Assurance**, Quality Assurance Review, vol. 2, no. 8, 1998, p. 8-14
11. Laurentiu TEODORESCU, Ion IVAN **Managementul Calitatii Software**, Inforec Printing House, Bucharest, 2001

---

[1] Catalin Boja is Assistant Lecturer at the Economic Informatics Department at the Academy of Economic Studies in Bucharest, Romania. In June 2004 he has graduated the Faculty of Cybernetics, Statistics and Economic Informatics at the Academy of Economic Studies in Bucharest. In March 2006 he has graduated the Informatics Project Management Master program organized by the Academy of Economic Studies of Bucharest. He is a team member in various undergoing university research projects where he applied most of his project management knowledge. Also he has received a type D IPMA certification in project management from Romanian Project Management Association which is partner of the IPMA organization.
He is the author of more than 40 journal articles and scientific presentations at conferences. His work focuses on the analysis of data structures, assembler and high level programming languages. He is currently conducting a PhD study on software optimization and on improvement of software applications performance.