What You Should Know About the Format of Records Saved to a Save File

Programming - RPG

Written by Juan Macias

Wednesday, 17 April 2013 00:00

0

When a file contains variable-length fields, the data of the fixed-length fields of its deleted records can still be retrieved.

A known technique to retrieve the records deleted from a file consists of saving the file to a save file and extracting the records from the save file. This technique is based on the fact that, in many cases, the records of the file are stored in the save file maintaining their format, apart from a status byte added by the system at the beginning of each record.

However, there are situations where the format of the records in the save file is different from the original format. This happens when one of the fields of the file is variable-length, nullable, or of type date-time (DDS types L, T, and Z). In these cases, we can still retrieve the data of the fixed-length fields of the deleted records and, what's more, the data of some variable-length fields for some of those records.

Some Terms

From now on, when we talk about the length of a field, we will be referring to the number of bytes (not characters) and, for variable-length fields, will not take into account the two initial bytes.

So the *current length* of a variable-length field will refer to the number of bytes taken up currently by the data of the field, its *maximum length* to the maximum number of bytes that its data can take up, and its *allocated length* to the number of bytes that the system allocates for its data.

For example, the maximum length of a single-byte variable-length field of 10 characters (10A VARLEN) is 10 (not 12), and its current length can be from 0 to 10. The maximum length of a graphic variable-length field of 10 characters (10G VARLEN) is 20 (not 22), and its current length can be 0, 2, 4...or 20.

We will say that a variable-length field is of *type 1* if the number of bytes allocated by the system for the field is equal to its maximum length. If it is less than its maximum length, we will say that the field is of *type 2*.

The space allocated by the system for a variable-length field is calculated as shown in Figure 1:

DDS field type	number of characters	DDS keyword	characters allocated	bytes allocated	type of varlen field
A	n <= 20	VARLEN	n	n	1
		VARLEN(m)	n	n	1
	n > 20	VARLEN	0	0	2
		VARLEN(m)	>= m	>= m	1 or 2
G	n <= 10	VARLEN	n	2n	1
		VARLEN(m)	n	2n	1
	n > 10	VARLEN	0	0	2
		VARLEN(m)	>= m	>= 2m	1 or 2

Figure 1: This is the space allocated for variable-length fields.

Note that the system can allocate some space for a variable-length field without the user's request. On the other hand, when the user requests the system to allocate a number of characters for a variable-length field, the number of characters eventually allocated by the system can be greater than the one requested. We will see later how to obtain the length of the space allocated.

Data Records Saved to a Save File

When a physical file is saved to a save file, the records of the physical file are stored in the save file with a format different from the original one. First of all, every record of the physical file is stored in the save file in a logical record A and, in some cases, in a logical record B.

Records A contain the data of the fixed-length fields, the variable-length fields of type 1, and the variable-length fields of type 2 whose current length is less than or equal to their allocated length. Records B contain the data of the variable-length fields of type 2 whose current length is greater than their allocated length.

To simplify the following description, let's suppose that only one of the members of the physical file is saved to the save file.

Records A (corresponding to the records of the saved member) are stored consecutively in positions 1 to 512 of a group of records of the save file. These positions of these records of the save file make up what we will call the area A of the save file. Records B are stored (in general, consecutively) in positions 1 to 512 of another group of records of the save file. We will call these positions the area B of the save file.

Records A and B contained in a save file are shown in Figure 2:

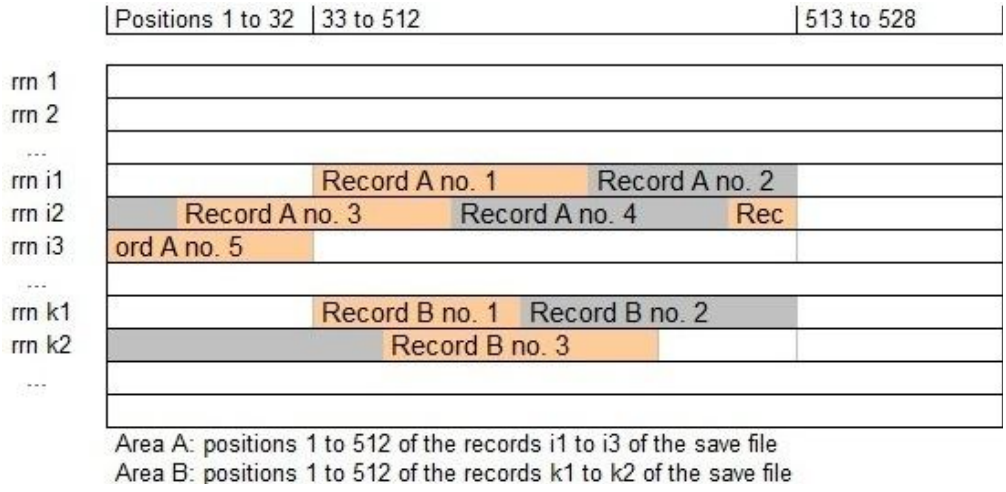

Figure 2: Here's an example of a save file containing a physical file.

To find out where area A begins, we have to look for a record of the save file whose positions 15 to 24 contain the name of the saved member. This is the first record of the save file containing information about that member. Then, look for a record whose position 2 is equal to x03 and whose position 33 is equal to x80. This is the first record of area A and contains the first logical record A from position 33.

To identify the initial record of area B, we have to look for the last record of the save file belonging to area A (we will see

how to in the section "Format of Records A") and then for a record whose position 2 is equal to x13. This is the first record of area B.

Areas A and B are divided into segments 16MB long (2^24 bytes). As each record of the save file contains 512 bytes, a segment is stored in 32K (32,768) records of the save file. The first 32 bytes of a segment are reserved for the segment itself.

A record A can be stored in two consecutive segments. In this case, the part of record A stored in the second segment will begin in position 33 of the segment (as its first 32 bytes are reserved).

A record B cannot be stored in two consecutive segments. If a record B does not fit into the remaining space of a segment of area B, the whole of it will be stored in the following segment from position 33.

Length of the Space Allocated for Variable-Length Fields

The length (in bytes) allocated by the system for each variable-length field of the physical file, as well as whether they are of type 1 or 2, is stored in the save file between the first record containing information about the saved member and the first record of area A (see section "Data Records Saved to a Save File").

We have to look for a record of the save file whose positions 47 to 56 contain the name of the member and then for a record whose positions 1 to 2 contain, in binary format, the number of fields of the file. An array with one item per field of the file is stored in this record from positions 33 to 512 (and, if necessary, in the following records from positions 1 to 512). Each item of this array is 32 bytes long. Its format is shown in Figure 3:

Offset	Bytes	Type	Description
0	9		
9	1		Variable-length indicator. < x80 for fixed-length fields >= x80 for variable-length fields
10	2		
12	2	binary	Number of variable-length field of type 2 in the physical file. 0 for fixed-length fields and for variable-length fields of type 1.
14	2	binary	Number of bytes allocated by the system for a variable-length field. Can be 0. 0 for fixed-length fields.
16	16		

Figure 3: This is the format of an item containing variable-length info for a field.

For an example, see section "Example of Records A and B for a File with Variable-Length Fields."

Format of Records A

RecordA = Status [VarLenInfo] [NullInfo] FieldA1 ... FieldAn [VarLenFiller]

where *[ ]* shows that an element can be present or not and *n* is the number of fields of the physical file.

Status

1 byte. Values x80 or xA0 show a valid record, xC0 or xE0 a deleted record, and x00 or x01 the end of the area A.

VarLenInfo = RecordBSegment RecordBOffset RecordBLen

Exists if any field of the physical file is variable-length. In this case, it is 8 bytes long.

RecordBSegment

2 bytes, binary format.
Number of segment of area B where record B lies. It is 0 when there is no record B associated.

RecordBOffset

4 bytes, binary format.
Offset of record B inside the segment. This offset ranges from x0000 0000 to x00FF FFFF (that is, byte 1 is always x00).

The absolute offset of record B in area B can be calculated as follows:
(*RecordBSegment* - 1) * 2^24 + *RecordBOffset*.

RecordBLen

2 bytes, binary format.
Length (in bytes) of the record B. Can be 0.

NullInfo

Exists if any field of the physical file allows the value null. In this case, 1 bit is used for every field of the file to store whether it is null (1) or not (0). Therefore, the number of bytes needed is *n* / 8 rounded up, where *n* is the number of fields of the file.

FieldAj = *FixedField* or *VarField* for j = 1 to n

VarLenFiller

Exists if any field of the physical file is variable-length and the length (in bytes) of record A is not a multiple of 16.
It contains as many bytes x00 as necessary for the length of the record A to be a multiple of 16.

FixedField

Contains the data of a fixed-length field. This data has the same format as the physical file except for the types shown in Figure 4:

Type	Bytes in file	Bytes in record A	Format in record A
Date	6, 8 or 10	4	binary, contains a Julian day number
Time	8	3	packed-decimal without sign
Timestamp	26	10	4 bytes for date, 3 for time and 3 for microseconds

Figure 4: These are the types with different formats in a physical file and a record A.

Figure 5 shows how some date-time values are stored in a record A:

Type	Value in ISO format	Value in record A
Date	0001-01-01	x001A 4452
	1900-01-01	x0024 D9AD
	2000-01-01	x0025 6859
	2100-01-01	x0025 F706
	9999-12-31	x0051 FE2C
Time	11:12:13	x1112 13
Timestamp	1900-01-01 11:12:13.212223	x0024 D9AD 1112 1321 2223

Figure 5: These are date, time, and timestamp values in record A.

As a field of type date is stored in record A as a number of days, we can obtain its equivalent in *LONGJUL format by taking any of the dates of Figure 5 as a reference and taking into account the leap years.

VarField = [VarFieldFiller] VarFieldLen [VarFieldAllocSpace]

VarFieldFiller

1 byte x00. Exists if the offset of *VarField* in record A is odd. This way, the offset of *VarFieldLen* will always be even.

VarFieldLen

2 bytes, binary format.

Contains the current length (in bytes) of this variable-length field.

Possible values: from 0 to 32740 (x7FE4)—that is, up to 32740 single-byte characters or 16370 graphic characters—and -32768 (x8000). The value x8000 shows that the data of this variable-length field is not stored in record A but in its associated record B.

VarFieldAllocSpace

Space allocated (if any) in record A for the data of this variable-length field. Its first *VarFieldLen* bytes contain the data of the variable-length field when *VarFieldLen* is different from x8000.

Can exist even when *VarFieldLen* is equal to x8000.

See section "Length of the Space Allocated for Variable-Length Fields" to learn how to obtain the length of *VarFieldAllocSpace*.

Properties of Records A

All of the records A have the same length.

If the physical file contains a variable-length field, the length (in bytes) of record A is a multiple of 16.

The offset in record A of the length of a variable-length field—that is, the offset of *VarFieldLen*—is always even.

The record number 1 of the physical file member is stored in record A number 2, the record number 2 in record A number 3, and so on. The record A number 1 does not correspond to any record of the physical file member.

A record A can be stored in two consecutive segments.

Format of Records B

RecordB = RRN1 AccumLenB1 ... AccumLenBm [DataB1] ... [DataBm] +
RRN2 EndOfRecord

where *[ ]* shows that an element can be present or not and *m* is the number of variable-length fields of type 2 of the physical file. See section "Length of the Space Allocated for Variable-Length Fields" to learn how to obtain *m*.

RRN1

4 bytes, binary format.

Relative record number of the record of the physical file member whose data is stored in this record B. Can be 0.

AccumLenBk for k = 1 to m

2 bytes, binary format.

Length (in bytes) occupied by the data of the *k* first fields of the record B—that is, the sum of the bytes occupied by *DataB1*, *DataB2*...and *DataBk*.

DataBk for k = 1 to m

Contains the data (if any) for the *k*th variable-length field of type 2 of the physical file. This data occupies *AccumLenBk - AccumLenB(k - 1)* bytes, being *AccumLenB0* equal to 0.

RRN2

4 bytes, binary format.
Relative record number of the record of the physical file member whose data is or was stored in this record B.

EndOfRecord

1 byte with value x73.

Properties of Records B

Records B can have different lengths.

The offset and length of record B are stored in its corresponding record A (in *VarLenInfo*).

When the offset of record B is different from 0, its length can still be 0 (meaning that record B contains no data right now).

A record B cannot be stored in two consecutive segments.

Deleted Records

For a record deleted from a physical file, all the data in its corresponding record A remains unchanged except *Status* —which is set to xC0 or xE0—and *RecordBSegment*, *RecordBOffset,* and *RecordBLen*—which are set to 0—so the data stored in *NullInfo*, *FieldA1*...and *FieldAn* can be retrieved. On the other hand, and unfortunately, we have no direct reference to the location of its record B.

Example of Records A and B for a File with Variable-Length Fields

Physical file FILE1:

```
A          R RFILE1
A            FLD1           3A
A            FLD2          30A         VARLEN
A            FLD3          30A         VARLEN(10)
A            FLD4          20G         VARLEN(5) CCSID(13488)
A* UCS-2: Space=x0020 1=x0031 2=x0032 3=x0033
A            FLD5           3A         ALWNULL

CRTPF FILE(QTEMP/FILE1) MBR(MEMBER1)

insert into QTEMP/FILE1 values
('AAA', ''     , 'FOURTEEN BYTES',
   vargraphic('12345' , 5, 13488), 'END'),
('BBB', 'VARLEN', 'FOURTEEN BYTES',
   vargraphic('123456', 6, 13488), 'END'),
('CCC', ''     , 'OVER FOURTEEN BYTES',
   vargraphic('12345', 5, 13488), null )
```

Save file:

```
SAVOBJ OBJ(FILE1) LIB(QTEMP) OBJTYPE(*FILE)
       DEV(*SAVF) SAVF(QTEMP/SAVF1)
       FILEMBR((FILE1 (MEMBER1))) UPDHST(*NO)
```

The format of records A and B are shown in Figures 6 and 7, respectively:

Offset	Bytes	Field	Comment
0	1	*Status*	
1	8	*VarLenInfo*	
9	1	*NullInfo*	
10	3	*FieldA1*	
		FieldA2	
13	1	*VarFieldFiller*	the offset of *FieldA2* is odd
14	2	*VarFieldLength*	
		FieldA3	
16	2	*VarFieldLength*	
18	14	*VarFieldAllocSpace*	length requested 10, assigned 14 (in bytes)
		FieldA4	
32	2	*VarFieldLength*	
34	10	*VarFieldAllocSpace*	length requested 10, assigned 10 (in bytes)
44	3	*FieldA5*	
47	1	*VarLenFiller*	length of this record must be a multiple of 16

Figure 6: This is the format of records A for FILE1.

Offset	Bytes	Field	Comment
0	4	*RRN1*	
4	2	*AccumLenB1*	
6	2	*AccumLenB2*	
8	2	*AccumLenB3*	
10		*DataB1*	data of *FieldA2* (varlen field of type 2 no. 1)
		DataB2	data of *FieldA3* (varlen field of type 2 no. 2)
		DataB3	data of *FieldA4* (varlen field of type 2 no. 3)
	4	*RRN2*	
	1	*EndOfRecord*	

Figure 7: This is the format of records B for FILE1.

Records of the save file:

First record containing information on MEMBER1:

```
                                                    RRN  From  To
*...+....1....+....2....+....3....+....4....+....5   33     1  50
    FILE1     MEMBER1              °   ø
FFFFCCDCF44444DCDCCDF4444444444440900070000000000
FFFF693510000045425910000000000000B000000000000000
```

Record with "MEMBER1" in position 47:

```
                                                    RRN  From  To
*...+....1....+....2....+....3....+....4....+....5   41     1  50
     m   o-¯L           Ø       a  °FILE1     MEMB
00000903296BD00000000008200F0008009CCDCF44444DCDC
01083404D60C30000100000000F00010B069351000004542

....+....6                                             51  60
ER1
CDF4444444
5910000000
```

Record with the number of fields in positions 1 and 2:

```
   < beginning of item   > end of item
```

```
   V Variable-length indicator   ww allocated length
   nn number of variable-length field of type 2

                                                        RRN  From  To
*...+....1....+....2....+....3....+....4....+....5   43     1  50
              Ë  §
00000000000000700B0000000000000000000000000000000
05030A1000090003035000000000000403000A0000000000
                                 <        V  nnww

....+....6....+....7....+....8....+....9....+....0         51 100
                       Ø
00000000000000000100000800000000000000000000000001
0000000000000040E000E000001000000000000000000040E
             ><        V  nnww               ><

....+....1....+....2....+....3....+....4....+....5        101 150
     Ø                              µ
000108000000000000000000000000200020A000000000000
00000000020E000000000000000006080000000030A000000
     V  nnww               ><        V  nnww

....+....6....+....7....+....8....+....9....+....0        151 200
                                          Ø
00020000000000002000000000000000000000080000000
0000000000403000C2000000000000000000000000000000
          ><       V  nnww               >
```

First (and only) record of the area A:

```
   Aj___ FieldAj   S Status   v VarLenInfo   n NullInfo
   f filler   ww length of variable-length field

                                                        RRN  From  To
*...+....1....+....2....+....3....+....4....+....5   49     1  50
     © u o-¯L                  Ø
00000B0A296BD0000000000000030000800000000044400000
03083404D60C3000010000000000100000000000800000000
                                        A1_A2_A3
                                Svvvvvvvvn   fwwww

....+....6....+....7....+....8....+....9....+....0         51 100
                                Ø         AAA     FO
00000000000000000000000000044408000000000CCC00000CD
00000000000000000000000000000000000000001110000E66
_____________A4_________A5_           A1_A2_A3__
             ww             fSvvvvvvvvn   fwwww

....+....1....+....2....+....3....+....4....+....5        101 150
URTEEN BYTES   ? ? ? ? ?END Ø         BBB Ø   FOUR
EDECCD4CEECE000303030303CDC08000002020CCC08000CDED
4935550283520A010203040555400010000010222000E6649
____________A4_________A5_           A1_A2_A3____
            ww             fSvvvvvvvvn   fwwww

....+....6....+....7....+....8....+....9....+....0        151 200
```

```
TEEN BYTESØ           END Ø         CCC   Ø
ECCD4CEECE800000000000CDC08000004020CCC00080000000
3555028352000000000000554000100010283330000000000
__________A4__________A5_           A1_A2_A3______
          ww             fSvvvvvvvvn   fwwww

....+....1....+....2....+....3....+....4....+....5       201 250
          ? ? ? ? ?
00000000000303030303444000000000000000000000000000
000000000A0102030405000000000000000000000000000000
________A4__________A5_
        ww             fS
```

First (and only) record of the area B:

```
   Bk___ DataBk   nnnn RRN   wk AccumLenk
   E EndOfRecord
                                                  RRN  From  To
*...+....1....+....2....+....3....+....4....+....5  57     1  50
     © u o-¯L                              VARLEN ?
01000B0A296BD0000000000000000000000000001ECDDCD03
03083404D60C3000010000000100000000020606025193550 1
                                          B1____B3
                               nnnnw1w2w3

....+....6....+....7....+....8....+....9....+....0        51 100
 ? ? ? ? ?    Ë          OVER FOURTEEN BYTES    Ë
030303030300007000000010 1DECD4CDEDECCD4CEECE000070
020304050600023000300030365590664935550283520003 30
__________              B2__________________
          nnnnEnnnnw1w2w3                  nnnnE
```

Appendix

Figure 8 contains the maximum values for database files that we have been implicitly dealing with:

Number of bytes in a record	32,766	= 2^15 - 2
Number of fields in a record format	8,000	< 2^13
Number of records contained in a file member	4,294,967,294	= 2^32 - 2
Number of bytes in a file member	1,869,162,846,624	< 2^41

Figure 8: These are the maximum values for database files (from V5R3 to V7R1).

Juan Macias

About the Author:

Juan Macias has a Degree in Mathematics (1988) and a Master in Data Management from the Universidad Complutense de Madrid (1989). He's a developer of RPG and COBOL applications mainly for finance and insurance companies. He can be contacted at correojrm@yahoo.es.

Read More > >

< Prev Next >

Last Updated on Saturday, 27 July 2013 13:48

What You Should Know About the Format of Records Saved to a Save File, Part 2

Programming - RPG

Written by Juan Macias

Wednesday, 21 August 2013 00:00

0

Do you need to retrieve data from variable-length fields of deleted records?

The data of variable-length fields of records deleted from a file can be retrieved provided that the inserts and updates performed in the file after the records were deleted didn't affect the variable part of the file.

In the first part of this article, we saw that, when a physical file is saved to a save file, the fixed and the variable parts of every record—that is, its records A and B—are stored separately in areas A and B of the save file. We also described the format of records A and B and saw that the data stored in area A of the save file could be retrieved for the deleted records.

In the second part of the article, we will see how the system manages the data of the variable part of the records and determine under which conditions this data can be retrieved for the deleted records.

Effect of File Operations on Records B

When a record is deleted from a file, the only field changed in its corresponding record B is *RRN1*, which is set to 0 (see section "Format of Records B" in the first part of the article). In addition, the space taken up by this record B is marked as available so that it can be reused to store data for another record of the file. This space is marked as available by adding an item containing its length, segment number, and segment offset to an array located in an area of the save file that we will call area 14. As the only change that needs to be done in area B when a record is deleted is to set field *RRN1* to 0, the system can delay this change until another access to area B is requested.

When a record of a file is updated, the new values that are to be stored in area B are kept in the current record B if it's long enough to hold them. If the new values don't fit into this record B, the system looks for an available record B (that is, a record B whose length and location are stored in area 14) long enough for the new values. If such a record B is found, the system reuses it to store the new values. If it's not found, a new record B is added at the end of area B. Finally, if the original record B could not be used to store the new values, its field *RRN1* is set to 0

and the space it takes up is marked as available.

For example, let's insert the two following records into physical file FILE2:

```
     A            R RFILE2
     A              FLD1            1A
     A              FLD2
30A                VARLEN
     A              FLD3
30A                VARLEN
     A              FLD4            1A

insert into QTEMP/FILE2 values
   ('1', 'AAA', '12345', '0'),
   ('2', 'BBB', '12345', '0')
```

The two corresponding records B are the following:

```
Area B (save file record 57):
   Bk___ DataBk   nnnn RRN   wk
AccumLenk
   E EndOfRecord
```



```
                                                    RRN  From  To
*...+....1....+....2....+....3....+....4....+....5   57     1  50
     © u 4ze¦                          AAA12345
01000B0A2FA860000000000000000000000000CCCFFFFF00
030834040495A000010000000100000000010308111234500
                                       B1_B2___
                                  nnnnw1w2        nn

....+....6....+....7....+....8....+....9....+....0        51 100
  Ë        BBB12345    Ë
00700000000CCCFFFFF000070000000000000000000000000
01300020308222123450002300000000000000000000000000
          B1_B2___
nnEnnnnw1w2        nnnnE
```

In the following update, the new values for fields FLD2 and FLD3 fit into the current record B:

```
update QTEMP/FILE2
set FLD2 = 'NEWAAA', FLD3 = '7'
where FLD1 = '1'

Area B:
*...+....1....+....2....+....3....+....4....+....5
     © u 4ze¦                          NEWAAA75
01000B0A2FA860000000000000000000000000DCECCCFF00
030834040495A000010000000100000000010607556111 7500
```

```
....+....6....+....7....+....8....+....9....+....0
  Ë        BBB12345    Ë
007000000000CCCFFFFF000070000000000000000000000000
013000203082221234500023000000000000000000000000
```

When a record is deleted, the field *RRN1* of its corresponding record B is set to 0 and the space taken up by this record B is marked as available:

```
delete from QTEMP/FILE2
where FLD1 = '1'

Area B:
*...+....1....+....2....+....3....+....4....+....5
     © u 4ze¦                          NEWAAA75
01000B0A2FA860000000000000000000**0000**0000DCECCCFF00
030834040495A000010000000100000**0000**06075561117500

....+....6....+....7....+....8....+....9....+....0
  Ë        BBB12345    Ë
007000000000CCCFFFFF000070000000000000000000000000
013000203082221234500023000000000000000000000000

Area 14 (save file record 65):
   ll length   ss segment number   oooo segment offset
*...+....1....+....2....+....3....+....4....+....5
     m u 4ze¦
0100090A2FA860000000000000000000**01000002**0000000000
040834040495A000010000000000000**05010000**0000000000
                                 llssoooo
```

In the next case, the new values for the record don't fit into its record B and the only record B available is not long enough for the new values, so a new record B is added at the end of area B:

```
update QTEMP/FILE2
set FLD2 = 'NEWBBB'
where FLD1 = '2'

Area B:
*...+....1....+....2....+....3....+....4....+....5
     © u 4ze¦                          NEWAAA75
01000B0A2FA8600000000000000000000000000DCECCCFF00
030834040495A000010000000100000000000006075561117500

....+....6....+....7....+....8....+....9....+....0
  Ë        BBB12345    Ë        NEWBBB12345    Ë
007**0000**0000CCCFFFFF00007**00000000DCECCCFFFFF00007**00
013**0000**030822212345002**30002060B556222123450002**300

Area 14:
*...+....1....+....2....+....3....+....4....+....5
     m u 4ze¦
0100090A2FA860000000000000000000100000**01000003**00
040834040495A000010000000000000050100**05010005**00
```

Let's delete the record we have just updated:

```
delete from QTEMP/FILE2
where FLD1 = '2'

Area B:
*...+....1....+....2....+....3....+....4....+....5
     © u 4ze¦                          NEWAAA75
01000B0A2FA86000000000000000000000000000DCECCCFF00
030834040495A000010000000100000000000006075561117500

....+....6....+....7....+....8....+....9....+....0
  Ë        BBB12345    Ë        NEWBBB12345    Ë
00700000000CCCFFFFF00007 0000 0000DCECCCFFFFF0000700
013000003082221234500023 0000 060B556222123450002300

Area 14:
*...+....1....+....2....+....3....+....4....+....5
     m u 4ze¦
0100090A2FA86000000000000000000010000020100000301
040834040495A000010000000000000005010000050100050 8

....+....6....+....7....+....8....+....9....+....0
     [
000004 0000000000000000000000000000000000000000000
01000A 0000000000000000000000000000000000000000000
```

When we insert the following record, the system reuses an available record B to store the fields FLD2 and FLD3. Note that the item in area 14 corresponding to this record B is initialized to 0.

```
insert into QTEMP/FILE2 values
('3', 'CCC', '123456', '0')

Area B:
*...+....1....+....2....+....3....+....4....+....5
     © u 4ze¦                          NEWAAA75
01000B0A2FA86000000000000000000000000000DCECCCFF00
030834040495A000010000000100000000000006075561117500

....+....6....+....7....+....8....+....9....+....0
  Ë        BBB12345    Ë        CCC12345645    Ë
00700000000CCCFFFFF00007 00000000CCCFFFFFFF 0000700
013000003082221234500023 0003030933312345645 0003300

Area 14:
*...+....1....+....2....+....3....+....4....+....5
     m u 4ze¦
0100090A2FA86000000000000000000010000020100000300
040834040495A000010000000000000005010000050100050 0

....+....6....+....7....+....8....+....9....+....0

000000 0000000000000000000000000000000000000000000
000000 0000000000000000000000000000000000000000000
```

Area 14

Area 14, as it happens for areas A and B, consists of the positions 1 to 512 of a group of consecutive records of the save file. From its position 33, it stores an array of items 8 bytes long containing the length and location of records B whose space has become available and can be reused by the system. Figure 1 shows the format of these items:

Offset	Bytes	Field	Comment
0	2	*Length*	length (in bytes) of record B
2	2	*SegmentNumber*	number of segment of area B where record B lies
4	4	*SegmentOffset*	offset of record B inside the segment

Figure 1: This is the format of an item of area 14.

When an available record B is reused by the system to store data for a record inserted or updated in the file, its corresponding item in area 14 is initialized to 0.

To identify the record of the save file where area 14 begins, we'll use the fact that positions 7 to 18 of every segment of every area of a given member are the same. Beginning at the record of the save file that follows the first record of area B, we will look for a record of the save file whose position 2 is equal to x14 and whose positions 7 to 18 are equal to positions 7 to 18 of the first segment of area B.

The end of area 14 is determined by the beginning of the following area. This area begins at the following record of the save file whose position 2 is equal to x15 and whose positions 7 to 18 are equal to positions 7 to 18 of the first segment of area B.

Area 14 seems to have a maximum size, at least in many cases. Currently, this maximum size is 16MB—that is, one segment—so area 14 can store up to (16MB – 32 bytes) / 8 = 2,097,148 items. If this number of items is reached, the system will ignore the requests to store new items in area 14 until there is some free space—that is, until some of the available records B are reused by the system and their corresponding items in area 14 are initialized to 0.

Retrieving Data from Records B

The record B that an item of area 14 points to can be found in the record of the save file and at the position that is shown in Figure 2:

RRN of the record of the save file = RRN of the first record of the save file in Area B + (*SegmentNumber* – 1) * 32,768 + integer_part(*SegmentOffset* / 512)
Position in the record of the save file = remainder(*SegmentOffset* / 512) + 1

Figure 2: These are the record of the save file and the position where record B begins.

Field *RRN2* of record B allows us to identify the corresponding record of the physical file and the corresponding record A in area A. Field *Status* of this record A will tell us whether the record of the physical file was deleted or not.

As we saw in section "Effect of File Operations on Records B", there can be several records B containing data for the same record of the physical file, one of them corresponding possibly to the insert of the record in the file and the others to some of the updates of the record. Taking into account how the system manages records B, we can get to the conclusion that the record B containing the latest data is the longest one.

In order to retrieve the data of the fields stored in area B for a record R that has been deleted from a file, we need to be certain that the latest record B—that is, the record B containing the latest data of record R—is present in area B. If some other record has been inserted or updated in the file after record R was deleted, and this insert or update required a record B, the system may have reused the latest record B of record R.

If the system did reuse the latest record B and there are another records B corresponding to record R, it would be incorrect to consider one of these records B as the latest one. For example, when record no. 3 was inserted into

FILE2 in section "Effect of File Operations on Records B", the system reused the record B containing the latest data of record no. 2 ("NEWBBB", "12345") to store the data for record no. 3 ("CCC", "123456"). The data from the other record B corresponding to record no. 2 ("BBB", "12345") is not the data record no. 2 had when it was deleted.

The only case where we can know that the latest record B has not been reused by the system is when its field *RRN1* is different from 0. As we saw in section "Effect of File Operations on Records B", this happens when a record has been deleted from the file and the system has delayed setting to 0 the field *RRN1* of the corresponding record B.

As a consequence, for each insert or update requiring a record B that has been performed in a file after the records were deleted, there may be a deleted record whose data in area B cannot be retrieved or can be retrieved but is not the latest one.

Another limitation to the retrieval of data from area B for a deleted record comes from the fact that area 14 has a maximum size. If an item cannot be stored in area 14 because this area has reached its maximum size, the location of the record B contained in that item is lost.

Finally, we must take into account that the fields whose data can be retrieved from the latest record B are only those whose corresponding length in record A is equal to x8000. A length of a field different from x8000 implies that the latest data for that field was not stored in the corresponding record B (see *VarFieldLen* in section "Format of Records A" in the first part of the article).

GETDLTRCD Command

The Get Deleted Records (GETDLTRCD) command is an implementation of the concepts seen in this article. It can be downloaded from the MC Press web site in English and Spanish. I would be grateful for your feedback on the command, especially if some error is found.

Juan Macias

About the Author:

Juan Macias has a Degree in Mathematics (1988) and a Master in Data Management from the Universidad Complutense de Madrid (1989). He's a developer of RPG and COBOL applications mainly for finance and insurance companies. He can be contacted at correojrm@yahoo.es.

Read More >>

Articles by this Author:

- What You Should Know About the Format of Records Saved to a Save File

View all articles by this author

Next >

Last Updated on Wednesday, 21 August 2013 04:19

An update

In a previous article ("Format of records saved to a save file, Part 2") I included the command GETDLTRCD (Get Deleted Records) for version V5R4. This command allowed, in certain cases, to retrieve variable-length fields from deleted records.

In version 7.3, the internal management of variable-lengh fields has changed, and the traces that allowed to locate the variable-length data of deleted records have been partially removed. As a consequence, the command GETDLTRCD cannot retrieve variable-length fields in 7.3 systems.

Anyway, this command can still be used to retrieve fixed-length fields from deleted records, even when the file contains variable-length, nullable or date-time fields.

You can download this command for versions 7.1 and 7.3—in English and Spanish—from mcpressonline.com.