Working with null-capable fields in RPG

Barbara Morris
IBM

Agenda

- **What is a null-capable field?**
- Working with null-capable fields in RPG
- Working with null-capable fields in embedded SQL
- Trigger programs

Null-capable fields

A null-capable field has

- Its value, 3.2, 'Jack Sprat' etc.
- An associated value that says whether it is null or not, called a "null indicator"

If the null indicator is "on", then the value in the field is meaningless.

For example, if a customer does not have any orders, the due-date for the orders is meaningless, so it may be useful to define the "duedate" field as null-capable, to avoid trying to use the date's value when it has no meaning.

Defining a field as null-capable in a file

When you create a table with SQL, fields are null-capable by default

```
CREATE TABLE MYLIB/TESTNULL
   (NUM_ORDERS DECIMAL (7, 0) NOT NULL WITH DEFAULT,
    DUE_DATE DATE)
```

- The NUM_ORDERS field is defined with "NOT NULL WITH DEFAULT", so it is not null capable
- The DUE_DATE field does not have "NOT NULL", so it is null-capable

In DDS, you use the ALWNULL keyword

```
A      R REC
A        NUM_ORDERS      7P 0
A        DUE_DATE         L          DATFMT(*ISO) ALWNULL
```

The "null-byte map"

The I/O buffer for a file has a separate section called the "null-byte map" which has an indicator for each field in the file indicating whether it is null or not.

(If the field is not null-capable, the null-byte-map indicator for that field is always '0'.)

The "null-byte map"

Imagine a file with three fields

- NAME: not null-capable
- DUEDATE: null-capable
- PRVBAL: null-capable

Here are the I/O buffer and null-byte map for a sample record

```
Buffer: Jack Sprat     0001-01-010041.75

Null-byte map: 010
```

Null-capable field "DUEDATE" has the null-value. Its value of 0001-01-01 is meaningless.

Displaying null-valued fields

STRSQL output

```
....+....1....+....2....+....3....+.
NAME                 DUEDATE     PRVBAL
Jack Sprat           -             41.75
Mary Contrary        17/06/01            -
```

DSPPFM output shows the default value. Why?

```
*...+....1....+....2....+....3.
Jack Sprat      0001-01-01004175
Mary Contrary   2017-06-01000000
<---- NAME --------><- DUEDATE -><PRVBAL>
```

Displaying null-valued fields

The "field" part of a null-capable field always has a value, even if the null-indicator is on.

DSPPFM shows the values of the "field" part of a field. It does not have any way to show the null-indicators.

- For the record 1, DUEDATE is null. The field value is '0001-01-01'.
- For the record 2, PRVBAL is null. The field value is 0000.00.

```
*...+....1....+....2....+....3.
Jack Sprat      0001-01-01004175
Mary Contrary   2017-06-01000000
<---- NAME --------><- DUEDATE -><PRVBAL>
```

Agenda

- What is a null-capable field?
- **Working with null-capable fields in RPG**
- Working with null-capable fields in embedded SQL
- Trigger programs

ALWNULL(*USRCTL) keyword

To work with null-capable fields in your RPG module, you need to compile with ALWNULL(*USRCTL)

Either

- As a command parameter
- As an H-spec keyword

I recommend using the H-spec keyword, to ensure the module is always compiled correctly

The null-byte indicator in RPG

The associated null-indicator for null-capable fields is an internal variable maintained by the RPG compiler.

- Prior to 7.3, this was always the case.
- Starting in 7.3 this is the default (more on this later)

You refer to the null-indicator using the %NULLIND built-in function

```
dueDate = curDate + %days(30);
%nullind(dueDate) = *off;       // set to "not null"

read custfile;
if %nullind(duedate);           // check if null
```

Reading a record with null-capable fields

When you read a record containing null-capable fields

- The buffer values get moved into the program fields
- The null-byte map values get moved into the associated null-indicators of the null-capable fields

```
Buffer: Jack Sprat      0001-01-010041.75
Null-byte map: 010
```

```
ctl-opt
alwnull(*usrctl);

dcl-f custfile;
read custrec;
```

```
> EVAL name
  NAME = 'Jack Sprat    '
> EVAL duedate
  DUEDATE = '0001-01-01'
> EVAL _QRNU_NULL_DUEDATE
  _QRNU_NULL_DUEDATE = '1'
> EVAL PRVBAL
  DUEDATE = 41.75
> EVAL _QRNU_NULL_PRVBAL
  _QRNU_NULL_PRVBAL = '0'
```

Writing a record with null-capable fields

When you write or update a record containing null-capable fields

- The program field values get moved into the buffer
- The associated null-indicators of the null-capable fields get moved into the null-byte map. The null-byte map for the non-null-capable fields is set to '0'.

```
dueDate = curDate + %days(30);
%nullind(dueDate) = *off;  // not null
update custrec;
```

```
Buffer: Jack Sprat      2016-06-150041.75
Null-byte map: 000
```

If you forget to set off the null-indicator

```
dueDate = curDate + %days(30);
update custrec;
```

```
Buffer: Jack Sprat      2016-06-150041.75
Null-byte map: 010
```

Oops!

After the update, DUEDATE is still null!

```
*...+....1....+....2....+....3.
Jack Sprat      0001-01-01004175
```

How to define null-capable fields in RPG

When ALWNULL(*USRCTL) is in effect …

If a field in a file is null-capable, the following RPG fields are also null-capable

- Fields from externally-described files
- Subfields in externally-described data structures

How to define null-capable fields in RPG

```
ctl-opt alwnull(*usrctl);
dcl-f testnull;
dcl-ds extds extname('TESTNULL') qualified end-ds;
```

Null-capable fields are indicated in the cross reference with "ALWNULL":

```
Global Field References:
   Field                Attributes
   DUEDATE              D(10*ISO-)      (From the file declaration)
                        ALWNULL
   EXTDS                DS(31)
     DUEDATE            D(10*ISO-)      (Data structure subfield)
                        ALWNULL
     NAME               A(15)           (Data structure subfield)
     PRVBAL             S(6,2)          (Data structure subfield)
                        ALWNULL
   NAME                 A(15)           (From the file declaration)
   PRVBAL               P(6,2)          (From the file declaration)
                        ALWNULL
```

Define your own null-capable fields

7.3 only

You can define your own null-capable fields starting in 7.3

Use the NULLIND keyword

```
dcl-s qty int(10) nullind;
```

- Field qty is null-capable
- The null-indicator is maintained internally by the RPG compiler, similar to null-capable fields related to externally-described files or data structures

Define your own null-capable fields and null-indicators

7.3 only

You can use your own indicator as the null-indicator for a field

Use NULLIND(*my_indicator*):

```
dcl-s qty_is_null ind;
dcl-s qty int(10) nullind (qty_is_null);
```

You can refer to the null indicator using its name, or using %NULLIND. These mean the same thing:

```
if qty_is_null;

if %nullind(qty);
```

Define your own null-capable subfields

7.3 only

You can associate an indicator subfield to be the null-indicator for another subfield in the same data structure

```
dcl-ds myds qualified;
   qty int(10) nullind(qty_is_null);
   qty_is_null ind(10);
end-ds;
```

As always, you can refer to the null indicator using its name, or using %NULLIND. These mean the same thing:

```
if myds.qty_is_null;

if %nullind(myds.qty);
```

Define your own null-byte map for a data structure

7.3 only

Use the NULLIND keyword to associate a data structure of null indicators with a data structure to represent whether the subfields are null.

The NULLIND data structure represents the null-byte map for the other data structure.

Use EXTNAME LIKEREC with *NULL to define the data structure of null indicators …

LIKEREC(rec:*NULL) and EXTNAME(file:*NULL)

7.3 only

*NULL defines a null-map data structure with indicator subfields instead of the actual types of the fields in the file.

The indicator subfields have the same names as the fields in the file.

```
dcl-ds cust_ds likerec(custrec) nullind(cust_null);
dcl-ds cust_null likerec(custrec : *null);

read custfile cust_ds;
if not cust_null.duedate; // duedate is not null
```

LIKEREC(rec:*NULL) and EXTNAME(file:*NULL)

7.3 only

If the main data structure has a specific extract type (*INPUT etc), define the NULLIND data structure the same way, adding *NULL.

```
dcl-ds cust_ds   likerec(rec : *output) nullind(cust_null);
dcl-ds cust_null likerec(rec : *output : *null);
```

Here, the two data structures represent the output record format.

*NULL – easier to work with trigger parameters

7.3 only

In trigger programs, there is a null byte map for the before and after record.

Before: Locate the null-indicator by its field number

```
dcl-s nullmap1 char(100) based(pNullmap1);

pNullmap1 = %addr(trigger_buffer) + null_offset1;
if %subst(nullmap1 : 2 : 1) = '1';
```

Now, less error-prone:

```
dcl-ds nullmap1 extname('CUSTFILE':*NULL)
                qualified based(pNullmap1) end-ds;

pNullmap1 = %addr(trigger_buffer) + before_null_offset;
if nullmap1.duedate;
```

Bonus: duedate is an indicator, so there is no need to compare it to '1' now.

When does RPG handle the null-indicator?

ALWNULL(*USRCTL) stands for "user controlled"

- The "user" is the RPG programmer

RPG sets or uses the null-indicator automatically in a few places

- When a null-capable field is read from a file (the null-indicator is set off if the field is not null-capable in that particular file)
- When a null-capable field is written or updated to a file
- When a null-capable field is used as a key in a list of key fields or %KDS
- During the EVAL-CORR opcode

When does RPG not handle the null-indicator?

RPG does not handle the null-indicator

- When a value is assigned to the null-capable field, the null-indicator is not set on

```
dueDate = curDate + %days(30);
%nullind(dueDate) = *off;  // Must be explicitly set
```

- When a null-capable field is used in a calculation, the null-indicator is ignored

```
%nullind(dueDate) = *on;  // DUEDATE is meaningless now
final_date = duedate + %days(30); // But RPG allows this
```

EVAL-CORR

EVAL-CORR assigns subfields with the same name and compatible data types

It also assigns null-indicators

- If both the source and target subfields are null-capable, both the value and the null-indicator are assigned
- If the target is null-capable and the source is not null-capable, the value is assigned and the target null-indicator is set off
- If the source is null-capable and the target is not null-capable, the value is assigned and the source null-indicator is ignored

EVAL-CORR

```
dcl-ds ds1 qualified;
   a char(10);
   b char(10) nullind;
   c char(10);
   d char(10) nullind;
end-ds;

dcl-ds ds2 qualified;
   a char(10);
   b char(10);
   c char(10) nullind;
   d char(10) nullind;
end-ds;
```

```
%nullind(ds2.c) = *on;
%nullind(ds1.d) = *on;
eval-corr ds1 = ds2;


// Equivalent to

ds1.a = ds2.a;

ds1.b = ds2.b;
%nullind(ds1.b) = '0';

ds1.c = ds2.c;

ds1.d = ds2.d;
%nullind(ds1.d) = %nullind(ds2.d);
```

EVAL-CORR

From the EVAL-CORR summary in the listing

```
          E V A L - C O R R   S u m m a r y
EVAL-CORR summary 1                          20
A    Assigned; exact match
B    Assigned; exact match
     Target subfield is null-capable; source subfield is not
C    Assigned; exact match
     Source subfield is null-capable; target subfield is not
D    Assigned; exact match
```

Checking/setting the null-indicator in the debugger

If the null-indicator is an internal field which can only be accessed by %NULLIND

- The name of the null-indicator in the debugger is

_QRNU_NULL_<name>

If PRVBAL is null-capable, the null-indicator is called _QRNU_NULL_PRVBAL

If CUST.DUEDATE is null-capable, the null-indicator is called _QRNU_NULL_CUST.DUEDATE

If you used NULLIND to associate your own indicator with the field, the _QRNU_NULL_ debug field does not exist

Agenda

- What is a null-capable field?
- Working with null-capable fields in RPG
- **Working with null-capable fields in embedded SQL**
- Trigger programs

Null-capable fields in embedded SQL

A normal host variable is specified with :varname

To specify the null-indicator for a host variable, you specify two names, the variable holding the value and the variable holding the null indicator

```
:fldname :nullindname
```

In the following example, a null-indicator is specified for fld1 and fld3, but not for fld2.

```
fld1null = -1;
exec sql insert into myfile
                values (:fld1 :fld1null,
                        :fld2,
                        :fld3 :fld3null);
```

Null-capable fields in embedded SQL

SQL null-indicators are two-byte integers (5i, or sometimes people use 2b or 4b)

- 0 means "not null"
- -1 means "null"

RPG null indicators have the same data type as other RPG indicators (single-byte character)

- '0' means "not null"
- '1' means "null"

SQL doesn't understand the %NULLIND relationship for RPG, and RPG doesn't have any concept of numeric null indicators

SQL null indicators vs RPG null indicators

If your SQL null indicators only contain 0 and -1, you can convert between RPG and SQL null-indicators like this:

```
    ind_sql = %int(ind_rpg) * -1;
OR
    ind_sql = - %int(ind_rpg);
    // '1'  -> -1
    // '0'  -> 0

    ind_rpg = %char(%abs(ind));
    // -1   -> '1'
    // 0    -> '0'
```

Special values for null indicators in embedded SQL

There are several other negative values that have special meanings in certain contexts. For example

- A null-indicator value of -5 means "use the default value" for an INSERT operation
- A null-indicator of -2 means there was some error in the field's value for a SELECT operation

See the "**References to host variables**" page in the knowledge center for more information about the special values for null indicators in embedded SQL

Note: These special values are only in effect if you compile with option *EXTIND (extended indicator support).

Agenda

- What is a null-capable field?
- Working with null-capable fields in RPG
- Working with null-capable fields in embedded SQL
- **Trigger programs**

Working with the null-byte maps in trigger programs

A trigger program parameter has four "buffer" sections

- The "before" buffer
- The "before" null-byte map
- The "after" buffer
- The "after" null-byte map

They are accessed using offsets in the first part of the parameter. From QSYSINC/QRPGLESRC TRGBUF:

```
D QDBORO          49     52I 0     Old Record Offset
D QDBORL          53     56I 0     Old Record Len
D QDBORNBM        57     60I 0     Old Record Null Byte Map
D QDBRNBML        61     64I 0     Old Record Null Byte Map Len
D QDBNRO          65     68I 0     New Record Offset
D QDBNRL          69     72I 0     New Record Len
D QDBNRNBM        73     76I 0     New Record Null Byte Map
D QDBRNBML00      77     80I 0     New Record Null Byte Map Len
```

Null-byte maps in trigger programs prior to 7.3

```
/copy qsysinc/qrpglesrc,trgbuf
dcl-pi *n;             // The parameter to this program
  parm likeds(QDBTB); // QDBTB defined in QSYSINC TRGBUF
end-pi;

// Define based data structures for file
dcl-ds beforeBuf  extname('TESTNULL' : *INPUT)
                  qualified based(pBeforeBuf) end-ds;
dcl-ds beforeNull qualified based(pBeforeNull);
   NAME ind;
   DUEDATE ind;
   PRVBAL ind;
end-ds;
dcl-ds afterBuf   extname('TESTNULL' : *INPUT)
                  qualified based(pAfterBuf) end-ds;
dcl-ds afterNull  likeds(beforeNull) based(pAfterNull);
```

For the null-byte map (beforeNull and afterNull), define one indicator for each field in the file

Null-byte maps in trigger programs prior to 7.3

```
// Set the basing pointers using the offsets in the parameter
pBeforeBuf  = %addr(parm) + parm.QDBORO;
pBeforeNull = %addr(parm) + parm.QDBORNBM;
pAfterBuf   = %addr(parm) + parm.QDBNRO;
pAfterNull  = %addr(parm) + parm.QDBNRNBM;


// Do some checking
if  afterNull.DUEDATE
and afterDs.PRVBAL <> *zero and not afterNull.PRVBAL;
   sndEscapeMsg ('PRVBAL must be zero if DUEDATE is null');
endif;
```

The null-byte maps in trigger programs, 7.3

```
ctl-opt alwnull(*usrctl);
/copy qsysinc/qrpglesrc,trgbuf

dcl-pi *n;              // The parameter to this program
  parm likeds(QDBTB); // QDBTB defined in QSYSINC TRGBUF
end-pi;

// Define based data structures for file TESTNULL
// - Link the null-byte map data structures to the
//   "ordinary" buffer structures using NULLIND
dcl-ds beforeBuf  extname('TESTNULL' : *INPUT)
                  qualified based(pBeforeBuf)
                  nullind(beforeNull) end-ds;
dcl-ds beforeNull extname('TESTNULL' : *INPUT : *NULL)
                  qualified based(pBeforeNull) end-ds;
dcl-ds afterBuf   extname('TESTNULL' : *INPUT)
                  qualified based(pAfterBuf)
                  nullind(afterNull) end-ds;
dcl-ds afterNull  extname('TESTNULL' : *INPUT : *NULL)
                  qualified based(pAferNull) end-ds;
```

7.3 only

The null-byte maps in trigger programs, 7.3

```
// Set the basing pointers using the offsets in the parameter
pBeforeBuf  = %addr(parm) + parm.QDBORO;
pBeforeNull = %addr(parm) + parm.QDBORNBM;
pAfterBuf   = %addr(parm) + parm.QDBNRO;
pAfterNull  = %addr(parm) + parm.QDBNRNBM;

// Do some checking
if  %nullind(afterDs.DUEDATE)
and afterDs.PRVBAL <> *zero and not %nullind(afterDs.PRVBAL);
   sndEscapeMsg ('PRVBAL must be zero if DUEDATE is null');
endif;
```

7.3 only

You could also code the "IF" like this, but the relationship between the fields and their null-indicators may not be as clear

```
// Do some checking
if  afterNull.DUEDATE
and afterDs.PRVBAL <> *zero and not afterNull.PRVBAL;
   sndEscapeMsg ('PRVBAL must be zero if DUEDATE is null');
endif;
```

How could this support be improved?

Ideally … in a perfect world …

- RPG would understand that when a value was assigned to a field, it's null-indicator should be set off
- RPG would understand that it is nonsense to use a field if its null-indicator is on
- RPG and embedded SQL would understand each other's null-indicators

Maybe some day …

RPG RFEs related to null-capable fields

There are some RFEs (Request for Enhancement) related to RPG and null-capable fields

- **Option for EXTNAME/LIKEREC(*NULL) to create SQL-type int(5) indicator subfields (97462)**
- **Consistent use for null values (97341)**
- **Full NULL support (90098)**

If you would like RPG's support for null-capable fields to be enhanced, vote for one or more of these RFEs

Use the "Comments" area to discuss the RFE

All RPG RFEs

Here a link that lists all the RFEs for RPG:

http://ibm.biz/rpg_rfe

Or

- Go to https://www.ibm.com/developerworks/rfe/
- Click "Search"
- Check the "I want to specify the brand, product family, and product" option.
- Select

  Product family: **Power Systems**
  Product: **IBM i**
  Component: **Languages - RPG**
  Brand: **Servers and System Software**

Creating an RPG RFE

If you don't find an existing RFE that describes what you want, open a new RFE:

- Go to https://www.ibm.com/developerworks/rfe/
- Click "Submit"
- Check the "I want to specify the brand, product family, and product" option.
- Select

  Product family: **Power Systems**
  Product: **IBM i**
  Component: **Languages - RPG**
  Brand: **Servers and System Software**

Make sure the headline is clear. You want to attract people to vote for your RFE.

Go to IBM

Special notices

This document was developed for IBM offerings in the United States as of the date of publication. IBM may not make these offerings available in other countries, and the information is subject to change without notice. Consult your local IBM business contact for information on the IBM offerings available in your area.

Information in this document concerning non-IBM products was obtained from the suppliers of these products or other public sources. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. Send license inquires, in writing, to IBM Director of Licensing, IBM Corporation, New Castle Drive, Armonk, NY 10504-1785 USA.

All statements regarding IBM future direction and intent are subject to change or withdrawal without notice, and represent goals and objectives only.

The information contained in this document has not been submitted to any formal IBM test and is provided "AS IS" with no warranties or guarantees either expressed or implied.

All examples cited or described in this document are presented as illustrations of the manner in which some IBM products can be used and the results that may be achieved. Actual environmental costs and performance characteristics will vary depending on individual client configurations and conditions.

IBM Global Financing offerings are provided through IBM Credit Corporation in the United States and other IBM subsidiaries and divisions worldwide to qualified commercial and government clients. Rates are based on a client's credit rating, financing terms, offering type, equipment type and options, and may vary by country. Other restrictions may apply. Rates and offerings are subject to change, extension or withdrawal without notice.

IBM is not responsible for printing errors in this document that result in pricing or information inaccuracies.

All prices shown are IBM's United States suggested list prices and are subject to change without notice; reseller prices may vary.

IBM hardware products are manufactured from new parts, or new and serviceable used parts. Regardless, our warranty terms apply.

Any performance data contained in this document was determined in a controlled environment. Actual results may vary significantly and are dependent on many factors including system hardware configuration and software design and configuration. Some measurements quoted in this document may have been made on development-level systems. There is no guarantee these measurements will be the same on generally-available systems. Some measurements quoted in this document may have been estimated through extrapolation. Users of this document should verify the applicable data for their specific environment.

Special notices