Database - Bug #5085

uppercasing and string comparisons are incorrect for some character sets

01/15/2021 10:13 AM - Greg Shah

Status:	New	**Start date:**	
Priority:	Normal	**Due date:**	
Assignee:		**% Done:**	0%
Category:		**Estimated time:**	0.00 hour
Target version:			
billable:	No	**case_num:**	
vendor_id:	GCD	**version:**	

Description

Related issues:

Related to Database - Feature #1587: implement full support for word indexes	**Closed**
Related to Database - Support #3871: determine how to change codepages/locale...	**Closed**

History

#1 - 01/15/2021 10:23 AM - Greg Shah

During the work on the CONTAINS operator (starting at #1587-229), Igor encountered a problem in data import:

> In the import of a large customer database I've got 4 errors for a single table on populating the related word table:
>
> ```
> ERROR: character with byte sequence 0xce 0x9c in encoding "UTF8" has no equivalent in encoding "LATIN1"
> ```

Ovidiu noted this is something we've seen before:

> There are some very similar messages posted in #3871.
>
> However, this article has a short paragraph about *iso8859-1 Client to 1252* conversion, if this is the problem. Most likely, it is the € (euro) sign, which is not present in WIN 1252 CP.

Between Igor and Ovidiu, they determined:

> > BTW: in LATIN1 µ is called not "Greek mu", but "micro", I think this is the reason why there is not its uppercase version in the codepage.
>
> This is an interesting observation. The latin micro (µ) is U00B5. The Greek letters μ and Μ are U03BC and U039C respectively. Notice that they are not the very same glyph. I do not know who does this but converting U00B5 to uppercase U039C is wrong.

and this:

> > The problem is that we convert to uppercase using Java toUpperCase() method which works with Unicode strings and knows nothing about the codepage. Note the codepage is the problem. It seems to me like a flaw in Java's uppercase implementation. Here is why:
>
> ```
>       char micro = '\u00B5';
>       char mulc = '\u03BC';
>       char muuc = '\u039C';
> ```

```
      System.out.println(micro + "" + mulc + "" + muuc + "->" + Character.toUpperCase(micro) + Character
.toUpperCase(mulc) + Character.toUpperCase(muuc));
      System.out.println(Character.toUpperCase(micro) == Character.toUpperCase(mulc));
```

The output is:

```
µµM->MMM
true
```

As you can see there are no CP involved. Maybe we should add a filter for this micro character when uppercasing it? I have not wrote/run the same code in 4GL yet.

Question: are there other characters which behave the same?

and this:

> PostgreSQL UPPER function for the LATIN1 database converts µ to µ (doesn't change it).

BTW: 4GL UPPER function also leaves µ as-is, at least if SESSION:CHARSET is ISO8859-1.

and this:

I peeked at java.lang.CharacterDataLatin1, method int toUpperCase(int) which is called from Character.toUpperCase(int) and saw this (see line 152, it starts with else):

```
    int toUpperCase(int ch) {
        int mapChar = ch;
        int val = getProperties(ch);

        if ((val & 0x00010000) != 0) {
            if ((val & 0x07FC0000) != 0x07FC0000) {
                int offset = val  << 5 >> (5+18);
                mapChar =  ch - offset;
            } else if (ch == 0x00B5) {
                mapChar = 0x039C;
            }
        }
        return mapChar;
    }
```

Of all characters in Latin1 charset, this is the only one that is handled individually! The others are either bitwise offset or let unchanged.

This led Igor to ask:

However, it means the 4GL UPPER and Java toUpperCase are not compatible. Should we re-work FWD toUpperCase() functions implementation to match 4GL behavior?

#2 - 01/15/2021 10:34 AM - Greg Shah

It clearly is something that needs to be fixed in some way.

Looking at our implementation, it is more than just the implementation of the UPPER() built-in function. There are also:

- CAPS() (used in WHERE clauses)
- ExpressionConversionWorker.toUpperCase() used at conversion time
- Text.compareTo() where we uppercase before comparison
- hundreds (or thousands?) of locations in collections where we uppercase before inserting a key in a map, an item in a set or lookup via get(), contains() or containsKey()
- and more, I'm sure

On top of this, today these locations all operate using the default JVM encoding AND I think we have (in the distant past) implicitly encoded dependencies that assume that the default JVM encoding is UTF-8. Our approach to I18N assumes that inside the JVM we will process everything as Unicode but when we handle input or output we will honor sources or targets which have a different encoding by translating from that encoding into Unicode (when reading input) and to that encoding (when writing output). The 4GL has a similar concept with CPSTREAM , CPTERM, CPPRINT and CPINTERNAL. The CPINTERNAL is their equivalent of the default JVM encoding. The main deviation we have (conceptually) is that we have been assuming Unicode for CPINTERNAL while OE defaults to ISO-8859-1/LATIN1/Windows 1252.

I think we first must determine if our current approach can meet all of our compatibility requirements. If not, then we need to consider the alternative. If so, then we need to make a list of the changes needed to implement properly.

#3 - 01/15/2021 10:34 AM - Greg Shah

- Related to Feature #1587: implement full support for word indexes added

#4 - 01/15/2021 10:34 AM - Greg Shah

- Related to Support #3871: determine how to change codepages/locales during import added

#5 - 01/15/2021 11:34 AM - Ovidiu Maxiniuc

If we talk about comparing insensitive character data, there is a dirty workaround. I noticed that toLowerCase() in java.lang.CharacterDataLatin1 has no exceptions. So we can use LOWER instead of UPPER. However, one issue still remains: the CAPS() function. According to Igor's it will work correctly with inlined PSQL SQL functions, but for the rest of internal FWD of CAPS() and H2 SQL we need to re-implement it. Not sure about MSSQL dialect.

#6 - 01/15/2021 11:48 AM - Greg Shah

As noted in #1587-235, lowercasing has issues too:

> One thing here which is inconsistent in the 4GL is about using uppercase or lowercase in case-insensitive comparisons. Long ago, we found that (at least for character types inside the 4GL code) the case insensitive comparisons used UPPERCASE. This can be seen in our comments from Text.compareTo() in the case insensitive path:

```
      // DO NOT use String.compareToIgnoreCase() since this lowercases and yields different
      // results for >, <, >= and <= forms when [ \ ^ _ ' characters are included in the operands
      return s1.toUpperCase().compareTo(s2.toUpperCase());
```

I don't think we can implement a quick and dirty solution here. It is time to address the broader set of problems.